Online POMDP Planning For Vehicle Navigation In Densely Populated Area

Cai Shaojun

B.Sc.,South China University of Technology, 2011

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2014

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

CAI SHAOJUN Oct 13. 2014

Cai Shaojun

March, 2014

Acknowledgement

It would not have been possible to finish this work without the help and support of many people, to only some of whom it is possible to give particular mention here.

It is with immense gratitude that I acknowledge the support and help of my supervisor Professor David Hsu and co-supervisor Lee Wee Sun. Their continuous support constantly led me in the right direction.

I would also appreciate the help from my lab colleagues in my research. I would like to thank Ye Nan for his help in modifying the DESPOT solver for our experiment, and Bai Haoyu for his collaboration in the vehicle experiment. I would also like to thank the people in Singapore-MIT Alliance for their technical support to the engineering work.

Finally, I am deeply grateful to my parents, for their patient encouragement and support.

Contents

Declaration i

Acknowledgement i

Abstract vii

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Motivation 1

1.2 Contributions 3

1.3 Outline of This Thesis 4

2 Background 7

2.1 Related Work 7

2.1.1 Autonomous Navigation Systems 7

2.1.2 Background of Motion Planning 10

2.1.3 Reactive Planning 11

2.1.4 Planning Under Uncertainty 12

2.2 Preliminaries on POMDPs 13

2.2.1 Sequential Decision Problems and Markov Decision Process . . 14
2.2.2 Partially Observable MDP 14
2.2.3 POMDP Solvers 15
2.2.4 Comparison between Offline and Online POMDP 16

3 Environment Modeling 19
3.1 Environment 19
3.1.1 Maps and Static Obstacles 19
3.1.2 Pedestrians as Dynamic Obstacles 21
3.2 Robotic Motion 22
3.3 Intention-Aware POMDP Modeling 23
3.3.1 Global Space to Local Space 25
3.3.2 State and Observation 27
3.3.3 Action and Transitions 29
3.3.4 Reward Function 31

4 Planning Techniques 35
4.1 Online POMDP Planning 35
4.1.1 Belief Tracking 36
4.1.2 Online Tree Search 41
4.2 Other Planning Techniques 46
4.2.1 Reactive Planning 47
4.2.2 Offline Planning and its Limitations 48
4.2.3 Other Probabilistic Methods 51
4.3 Experiments in Simulation 52
4.3.1 Experimental Setup 52
4.3.2 Results 56

5 Experiments on the Autonomous Vehicle **59**

5.1 System Overview 59

5.1.1 Hardware 59

5.1.2 Software 60

5.2 Experiment Results 64

5.2.1 Environment Description 64

5.2.2 Case Analysis 66

5.2.3 Implementation Issues 72

6 Conclusion and Future Work **75**

6.1 Conclusion 75

6.2 Future Work 76

Bibliography **79**

Abstract

Technologies for autonomous vehicles have advanced dramatically in the last decade. The Google car and other autonomous vehicles have driven over long distances under difficult conditions. However, it remains a challenge for these vehicles to navigate safely, reliably, and smoothly among pedestrians and other human-driven vehicles in densely populated urban centers. One major difficulty is the uncertainties arising from unknown pedestrian intentions, unexpected changes in the environment, sensor noise, and imperfect vehicle control.

The partially observable Markov decision process (POMDP) is a principled general framework for decision making and planning under uncertainty. In this work, we develop a POMDP model that captures systematically the main modes of uncertainty when an autonomous vehicle navigates among pedestrians and exploit the model for effective autonomous driving. One limitation of the POMDP framework is high computational complexity. By leveraging a state-of-the-art online POMDP algorithm and constructing the model suitably to take advantage of its strengths, we demonstrate an successful application of the POMDP framework to autonomous vehicle navigation among pedestrians. In simulation, we analyze the performance of our POMDP approach in comparison with alternative models and algorithms in uncertain, dynamic environments. We further show that our autonomous golf-cart under the POMDP controller is able to navigate safely and smoothly in a dense crowd

on the U-Town Plaza at our university.

List of Figures

2.1 The MIT-Cornell collision 8
2.2 Comparison between offline and online approaches. $t_0, t_1, \ldots, t_n$ are the time steps during online execution. 15

3.1 Laser scan map of a campus environment 20
3.2 Demonstration of the concept of subgoal in the campus environment . 22
3.3 (a) is the global world model, in which the vehicle is moving from start to the end and the pedestrian is moving to its current subgoal G2. (b) is the corresponding local world model finally processed by the solver. The red grid cells represent the future positions of the vehicle. And the numbers show an example of pedestrian transition probability distribution based on its current subgoal. 24
3.4 The probabilistic network of our model. 26

4.1 Demonstration of the concept of 1-dimension virtual bumper [Schiller *et al.*, 1998] 47
4.2 Comparison of the pedestrian's transition probability with and without obstacles. 49
4.3 The difference in pedestrian's transition probability at the same local position 50

4.4 Straight lane environment 54
4.5 Straight lane environment 55
4.6 UTown Plaza environment 55

5.1 YAMAHA G22E golf cart 60
5.2 System architecture 61
5.3 (a) is the static map of the UTown Plaza environment. (b) is a runtime map of the environment with dynamically detected obstacles. (c) and (d) are the photos taken on spot. 65
5.4 Demonstration of the vehicle's behavior when a pedestrian stops besides its path. 68
5.5 Interacting with dense pedestrian crowd 70

List of Tables

4.1 Parameters 54
4.2 Result for straight lane 56
4.3 Result for roundabout 56
4.4 Result for UTown Plaza 57

Chapter 1

Introduction

1.1 Motivation

Autonomous vehicles have gained significant attention from all over the world. The DARPA Grand Challenge [Iagnemma and Buehler, 2006] has influenced many organizations over the past decade to design driverless cars that could win the race in different environments. Google self-driving car [Markoff, 2010] got the permission to drive on the road in the state of Nevada, USA in 2012. Autonomous vehicles have many potential advantages, such as reducing potential traffic accidents, increasing road efficiency, releasing manpower, and many more.

Equipped with high-performance sensing unit (e.g. Velodyne LIDAR) and localization unit (e.g. Applanix Intertial Navigation System), autonomous vehicles can ensure safety in most situations with short reaction time. For instance, Google cars and the vehicles in the DARPA Challenge have all passed the safety test of long distance driving.

There are strict traffic rules on the highway or urban roads, such as vehicles drive on their own lanes and pedestrians walk on the sides. Even though some more difficult scenarios such as lane changing and pedestrians crossing may happen from time to time, the behavior of the vehicles and the pedestrians still follow predictable patterns (pedestrians usually cross at the zebra crossing, *etc.*). However, driving on the highway is not enough. To successfully deliver the passengers to their destinations, the vehicle must drive through the last mile, which often contains densely populated areas such as campuses, parks, or narrow and crowded streets. In these places, collision avoidance becomes a significant challenge. Firstly, there are no obvious lanes to follow and the pedestrians' behavior become more subtle and seemingly irregular due to the variety of their intentions. Furthermore, dynamic obstacles such as temporarily parked cars may block pedestrians' paths, making their movements even more unpredictable. Secondly, Smoothness of vehicle motions also becomes a key criterion of a plausible touring experiment. We not only need to make decisions about whether to yield, overtake, turn, *etc.*, but also have to ensure the basic comfort of the passengers by avoiding sudden brakes, fast turns, or frequent stops.

To summarize the above, we want to provide a practical solution to the problem of autonomous vehicle navigation among pedestrians in densely populated environment. Below are some of the challenges:

- Uncertainties:

  In reality, our autonomous vehicle runs in an imperfect world full of uncertainties, for example, noise in imperfect vehicle control and localization need to be hedged against, and the noise in the pedestrians' motion make it difficult to determine their true intentions and predict their next positions.

- Dynamic Environments:

  The model should account for not only the static elements in the environment but also all the possible dynamic ones. For example, pedestrians are typical examples of dynamic elements which may affect the vehicle's decision during navigation. Furthermore, the pedestrians' motion model is not so straightforward, as they constantly interact with the environment and occasionally change their minds. Specifically, pedestrians' behavior may change as a result of unexpected obstacles and surrounding vehicles.

- Safety and Smoothness:
  Unlike indoor environment, outdoor navigation usually applies to larger-scale vehicles and may result in serious consequences when collision happens. On the other hand, overly-conservative policy may result in a jerky driving behavior especially when interacting with large crowds of people. Therefore, we need an approach to appropriately balance the two orthogonal requirements well.

With the above requirements, a general and automated approach is demanded, since it is difficult for a human designer to construct a policy for all the possible situations encountered during the navigation. Of all the approaches, Partially Observable Markov Decision Process (POMDP) can systematically accounts for the uncertainties in controlling and sensing. Once a successful POMDP model has been built, it can be easily extended to a new system by adjusting a few parameters. In addition, online POMDP solvers can avoid the state space limitation exists in the offline POMDP solvers by doing a local planning only for the near future.

1.2 Contributions

We develop an online POMDP approach for vehicle navigation in densely populated outdoor environments and successfully test it on an autonomous vehicle in the campus. Specifically our work makes the following contributions:

- We have designed a rich and effective probabilistic model that not only captures the dynamics of the environment, but also matches well with a powerful online solver. To tackle the problem of the solver's space limitation, we restrict planning to a local dynamically shifting window that supposedly only covers the most relevant areas. Another noteworthy point is that we use the concept "subgoal" to model the behavior of the pedestrians, which contributes greatly to our analysis of environmental dynamic.

- We leverage the state-of-the-art online POMDP planner to efficiently solve the model and generate the appropriate control action on-the-fly. Compared with simple reactive method, we take pedestrians' intentions into consideration and generate safer and smoother behavior; compared with traditional offline POMDP, we are greatly released from the limitation of the state space and the reliance on a-priori static model.

- We have successfully carried out the experiment in a campus, and the results indicate that our design has many advantages over the previous methods in terms of smoothness, safety and efficiency. In addition, this experiment allows us to identify the gap between computer simulation and real-world application, and thus making adjustments accordingly.

1.3 Outline of This Thesis

The rest of this document is structured as follows:

- Chapter 2 reviews the previous work and summarizes the background of our research.

- Chapter 3 presents the probabilistic model that we use. Firstly we describe the environment model and the general concept of our pedestrian modeling. Then we explain how we construct the POMDP model of the world.

- Chapter 4 describes the online planner we use to solve the problem of navigating in densely populated areas. We first explain the advantages of using online POMDP and compare with other methods. Then we validates with simulation experiments running in three example environment.

- Chapter 5 describes the real world experiment on an autonomous golf cart. We first describe system architecture. Then we elaborate the performance of our approach on the vehicle and how we tackle specific difficulties encountered.

- Chapter 6 concludes our work presented above and give some directions to future research.

Chapter 2

Background

2.1 Related Work

2.1.1 Autonomous Navigation Systems

Much work has been done on vehicle autonomous navigation. For example, the famous DARPA Urban Challenge (DUC) is a competition in which vehicles have to autonomously navigate in a complex urban environment populated with static and dynamic obstacles while obeying the traffic rules such as lane keeping and intersection precedence. Since the competition is funded by the Department of Defense, the intention of the competition is biased towards military application and all the vehicles involved are highly equipped. There occurred a few low-speed incidents during the competition, and a famous one was the MIT-Cornell collision in 2007 [Fletcher *et al.*, 2008] (Figure 2.1). According to the video of the accident and the analysis of the collision log, a leading cause of this collision was the failure to anticipate the true intention of the opponent vehicle [Fletcher *et al.*, 2008].

One of the most advanced examples in the area of autonomous navigation is Google car [Guizzo, 2011]. Google's autonomous navigation largely relies on their

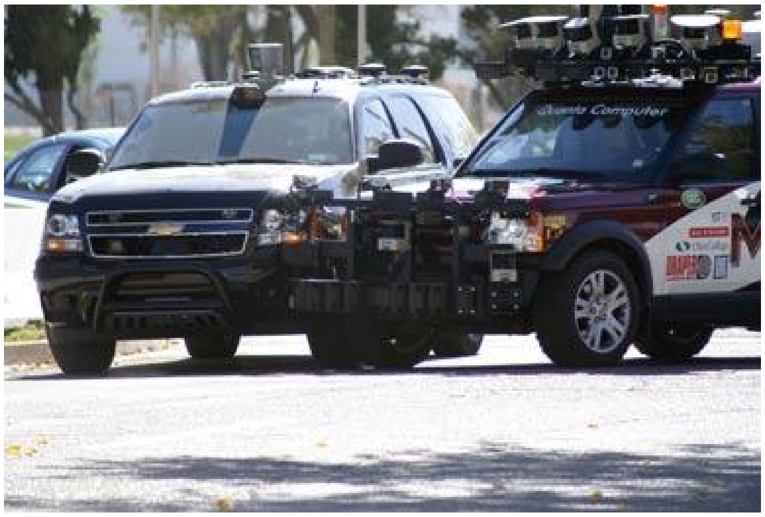


Figure 2.1: The MIT-Cornell collision

powerful Velodyne LIDAR sensor which can provide the planner with high awareness of the surrounding environment including adversarial and dynamic scenarios. However, Google Car's performance in a local densely populated environment is not clear. Dense crowd and dynamic obstacles may occlude the view of the sensor and therefore cause the sensor to lose its full awareness. Additionally, the high cost of high-performance sensor may also limit its use.

Singapore-MIT Alliance for Research and Technology (SMART) has also done some work on an adapted YAMAHA G22E golf cart. Instead of using high-performance sensor, this golf cart uses relatively low-cost 2D SICK laser scanner. The cart has done test navigation in the campus of National University of Singapore [Chong and others, 2011]. The navigation method used is Pure-Pursuit Control (PPC) [Kuwata *et al.*, 2009] as a path follower and Dynamic Virtual Bumper [Schiller *et al.*, 1998] as a collision avoidance system.

Besides the vehicles mentioned above, many major automotive manufactures such as General Motors, Ford and Volkswagen have also started to test their driverless car systems in recent years. Vislab [Bertozzi *et al.*, 2010] designed a challenging scenario in which vehicles had to drive from Italy to China autonomously. However, Vislab's vehicles all require a-prior maps for all the obstacles, and require a leader vehicle as a guide. In the past year, Induct Technology's Navia became the first commercialized autonomous self-driving vehicle [Maisto, 2014]. It can drive up to eight people at a maximum 20.1 km/h speed, providing shuttle services in city centers, parks and campuses. However, according to their website, the vehicle simply stops when it detects pedestrians in front, which is not very desirable in terms of efficiency and passenger experience.

On the other hand, some smaller-scale robots have managed to navigate in densely populated environments and interact with surrounding people. Among the earlier work, robot "RHINO" is a tour-guide robot deployed in a densely populated museum [Burgard *et al.*, 1999]. It gave tour to more than 2000 visitors during a six-day installation period while avoiding potential collision with surrounding tourists using Dynamic Window Approach [Fox *et al.*, 1997]. In Expo 2002, Robotx was installed at the spot performing tour-guiding and photo-taking tasks [Siegwart *et al.*, 2003]. It continuously operated for 159 days, up to 12 hours per day. More recently, an Interactive Behavior Operated Trolley (InBOT) [Goller *et al.*, 2010] could provide safe and reliable navigation to customers in the supermarket. All these robots could successfully function in a complex environment with dense pedestrian crowd. However, there are several key differences between the applications of these robots and our application

1. These robots are small and light compared to our golf cart testbed, and they run at a relatively low speed. Thus, safety is not a major issue. The worst accident for these robots would be "touching" someone while our golf cart can actually hurt or even kill people.

2. These robots do not carry passengers, while our golf cart can load several pedes-

trians. Therefore, in addition to safety, the touring experience of passengers has to be taken into consideration. Sudden acceleration, harsh brakes or long pauses are obviously unsatisfactory behavior.

2.1.2 Background of Motion Planning

Robotic motion planning has been an active area of research [Latombe, 1996] over the last few decades. Generally, the task of motion planning is to compute a sequence of intermediate states from a given start state to a specific goal state under certain kinds of constraints. In the earlier period, research typically focused on problems with simple constraints such as geometric constraints imposed by the static obstacles. Basically the planner need to compute a collision-free path from the initial position to the goal position. Most algorithms at that time worked under the concept framework of *configuration space* [Lozano-Perez, 1983], but the computation of the configuration space is durable only in spaces with low dimensions. When the degree of freedoms (DoF) increases, sampling-based approach emerged as a powerful tool to solve high-dimensional planning problems [Kavraki *et al.*, 1996]. To further address the issue of limited system mechanism ability, there emerged kino-dynamic motion planning [Donald *et al.*, 1993] which plans in the state space instead of configuration space.

The above work basically assumes that the model is perfect in a noiseless context. However, the real world is imperfect and contains all kinds of noises, which makes it hard to accurately predict the future state of the robot. Furthermore, the environment does not remain static as it contains all kinds of moving obstacles. There are two orthogonal ways to tackle this problem. One is to do reactive replanning once the environment changes; the other is to model the uncertainties and account for all possible future situations during the planning.

2.1.3 Reactive Planning

Generally, reactive planning techniques execute a policy which is computed based on the current world state or states can be reached in the near future. Usually these techniques make decisions based on simple criterion, such as the distance to nearby obstacles and the goal. There are several widely used reactive planning techniques. Trajectory rollout [Gerkey and Konolige, 2008] and Dynamic Window Approaches (DWA) [Fox *et al.*, 1997] are two approaches discretely sample velocities from the control space of the robot. Based on the robot's kinematic model, the algorithm generates trajectories by applying sampled velocity to the robot's current state with forward simulation that leads to the next state. The algorithm then scores each trajectory calculated by forward simulation, chooses the one with the highest score and executes the corresponding controlling command to the controller. Usually the optimization criterion contain the distance to the global path, the distance to the obstacles and the velocity of the robot. Dynamic window approach differs from the trajectory rollout in that dynamic window considers velocities that can be reached within a short time period(*e.g.*, one step of forward simulation). DWA has already been integrated as a standard local collision avoidance module in the ROS (Robot Operating System) [Quigley *et al.*, 2009], which is widely used on a large variety of robots [Quigley *et al.*, 2009].

Virtual Bumper [Schiller *et al.*, 1998] is a collision avoidance technique in charge of steering, braking and accelerating to ensure safe longitude and lateral control. The algorithm assumes a virtual spring and a damper in the personal space ahead of the vehicle and adjust the speed and the steering according to the assigned spring impedance when obstacles pop up in front.

Reactive planning techniques usually have fast reaction times but lack the ability to predict the future well, so that it breaks down easily, for example, trapped in local minima.

2.1.4 Planning Under Uncertainty

Motion planning can take into account uncertainty in sensing (the current state of the robot and workspace is not known with certainty), predictability (the future state of the robot and world cannot be deterministically predicted even when the current state and future actions are known) [LaValle, 2006]. Extensive work has explored uncertainty associated with robot sensing, including SLAM [Leonard and Durrant-Whyte, 1991] and POMDP [Kaelbling *et al.*, 1998] to represent uncertainty in the current state [Thrun *et al.*, 2005]. The predictability uncertainty can be further divided into the predictability of the robot's controlling and the predictability of the environment.

A lot of recent work explored the issue of uncertainty in robot's motion. Stochastic Road Map [Alterovitz *et al.*, 2007] builds a Markov decision process (MDP) model based on a Probabilistic Road Map (PRM) [Kavraki *et al.*, 1996] to handle the control uncertainty, but it does not take the observation uncertainty into consideration. Belief Road Map [Prentice and Roy, 2011] and LQG-MP [Van Den Berg *et al.*, 2011] handles the uncertainties from both controlling and sensing. However, they require the uncertainties to be Gaussian distributions which is improper for complex real world scenarios. For example, the distribution of pedestrian's intentions is not obviously Gaussian. In addition, LQG-MP only does local optimization and does not guarantee a solution that is globally optimal.

There are also much work studying the predictability of the dynamic environment, especially the pedestrians as an important component. Pedestrians do not move randomly in the environment, as there is certain rationale behind their motions which can be modeled in various ways. Tetsushi [Ikeda *et al.*, 2012] was the first to use the concept of subgoal to predict the pedestrian's future position. However, his work focuses more on the construction of the subgoal network based on the collected pedestrian trajectories data, as opposed to the navigation in the environment. Bennewitz's method [Bennewitz *et al.*, 2005] classifies pedestrians' motion trajectories into several representative motion patterns with EM algorithm, and a Hidden Markov Models (HMMs) is then derived from the resulting motion patterns to track the belief of the pedestrian positions. This method can utilize the prediction result to improve the path planning efficiency. However, it still considers prediction and control as two separate issues, and does not account for sensing and controlling noise.

As the recent development in the POMDP solving techniques, there are some recent works using POMDP-related approaches to model the sensing uncertainty and predicting uncertainty as a whole. Fern's work [Fern and Tadepalli, 2010] solves the problem of designing a computer assistant that provides helper action to a human agent by modeling it as a hidden-goal MDP (HGMDP), where the assistant can fully observe all the states of the system except for the human agent's goal. However, solving HGMDP is PSPACE-hard. To hedge against the difficulty, the paper introduces Helper Action Markov Decision Processes (HAMDP) which could solve a restricted version of the problem. Bandyopadhyay's work [Bandyopadhyay *et al.*, 2013] is more closely related to our work. It uses *Mixed Observability Markov Decision Process* (MOMDP), which is a structured variant of POMDP that makes controlling decisions under uncertainties. They model two kinds of uncertainties: the uncertainties from sensing and controlling, and the uncertainties from pedestrian's intention. However, due to the limitation of the off-line planner, only small scale problems can be solved, and the model dynamic has to be pre-defined.

2.2 Preliminaries on POMDPs

There are various approaches towards planning under uncertainties as mentioned in the related work, among which *Partially Observable Markov Decision Process* (POMDP) [Kaelbling *et al.*, 1998] is a systematic framework that can generate a globally optimized solution and balance exploration and exploitation.

2.2.1 Sequential Decision Problems and Markov Decision Process

Unlike the reactive planner which solves for the best action only for the current situation, *Sequential Decision Problem* tries to optimize the total utility of the entire action sequence from the beginning to the end. As uncertainties exist in the environment, the outcome of each action is a distribution over states. In a Markov Decision Model (MDP), the transition can be written as $P(s'|s,a)$ in which the outcome distribution only depends on the current state. Formally, a MDP consists of a tuple (S, A, T, R, γ) , where S, A are the system's state space and action space. T is the transition model in the form of $T(s,a,s') = P(s'|s,a)$. $R(s,a)$ is a reward function depending on current state s and action a. At each time step, the robot chooses an action, reaches the next state according to the transition probability, and gets the reward according to the reward function. The optimal policy maximizes the total expected reward $E(\sum_{t=0}^{\infty} R(s_t, a_t))$. After solving the MDP, we get a policy $\pi : S \rightarrow A$, which maps every state in the state space to an optimal action that maximizes its future expected reward.

2.2.2 Partially Observable MDP

In MDP, one action can lead to several future states. However, after the action is actually executed, the next state is uniquely determined. Equally speaking, the MDP assumes the sensor can perfectly sense the current state. POMDP has the same elements as MDP, but it assumes the sensor model to be is imperfect and thereby adding in a sensor model $Z(s', a, o) = p(o|s', a)$ that models the observation uncertainty. In every horizon of a POMDP problem , the robot is in a belief state $b(s)$ which is a probability distribution over the state space S. This distribution can be represented as a $|S|$ dimension vector. Now the optimal policy becomes a mapping from belief space to action space $\pi : B \rightarrow A$.

2.2.3 POMDP Solvers

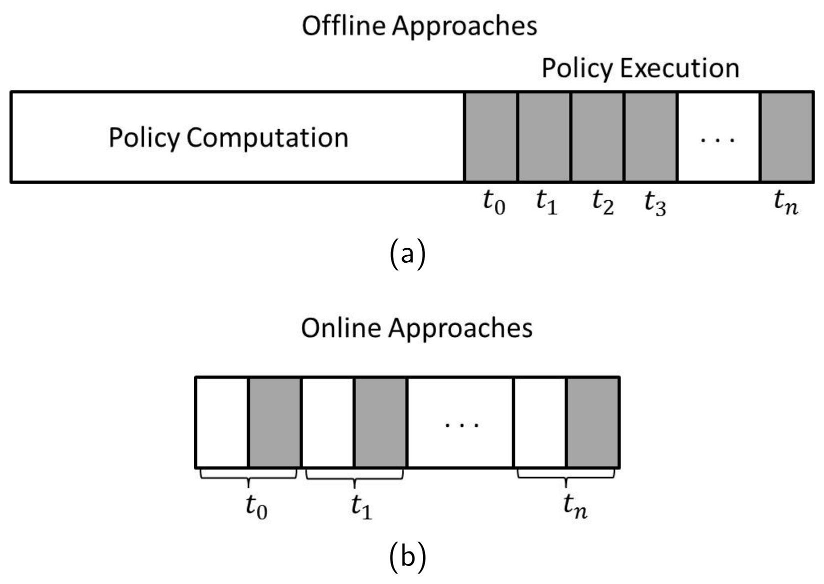


Figure 2.2: Comparison between offline and online approaches. $t_0, t_1, \ldots, t_n$ are the time steps during online execution.

POMDP solvers can be classified into offline POMDP solvers and online POMDP solvers. As in Figure 2.2a, offline solvers first compute the best action for all the possible situations (belief state), and then choose action according to the computed policy during execution. Since the computation time could be extremely long, many approximate algorithms have emerged during recent years to improve efficiency. Point-based algorithms [Pineau *et al.*, 2003; Spaan and Vlassis, 2005; Shani *et al.*, 2007] are currently among the most successful approaches. The key idea is to represent the continuous belief space B with a set of samples. HSVI [Smith and Simmons, 2004] and SARSOP [Kurniawati *et al.*, 2008] are among the state-of-the-art solvers that bias sampling towards the most promising belief space, in order to deal with large state space problems. MOMDP [Ong *et al.*, 2010] reduces the planning complexity through a factored model which separates the fully and partially observable state components so that it can greatly improve the efficiency under suitable conditions.

On the other hand, online solvers try to circumvent the complexity by planning for the current information state. As in Figure 2.2b, the policy computation and execution steps are interleaved. Online algorithms consume a short time period to compute the policy for a small set of beliefs reachable in the near future, execute according to the policy, and then recompute the whole policy for the next belief state. A recent survey [Ross *et al.*, 2008] lists three main categories of online planning algorithms: heuristic search, branch-and-bound pruning, and Monte-Carlo sampling. Heuristic search algorithms use a heuristic to help focus the search on the most relevant reachable beliefs. For example, AEMS2 [Ross *et al.*, 2007] explores the search tree by always expanding the fringe node with the highest expected error contributed to the current belief state. RTBSS algorithm [Paquet *et al.*, 2005] uses branch-and-bound pruning technique to prune branches that are known to be suboptimal. Other than the two categories, Monte Carlo sampling algorithms try to tackle the large observation space by sampling a subset observations and considering only beliefs reachable from the sampled observations. POMCP [Silver and Veness, 2010] and DESPOT [Somani *et al.*, 2013] are the state-of-the-art algorithms in this category. Unlike the other Monte Carlo tree search algorithms [McAllester and Singh, 1999; Asmuth and Littman, 2011], POMCP and DESPOT represent the belief in each node with particles, and perform particle filtering for belief update.

2.2.4 Comparison between Offline and Online POMDP

Offline POMDP planners pre-compute which action to execute for every possible belief state. By doing this, the planner can generate the optimal action without much additional computation during the execution stage. However, although the offline planning efficiency has been improved through point-based approximation in recent years, it is still difficult for offline planners to handle large scale problems [Ross *et al.*, 2008]. Besides, offline planners have to recompute the whole policy even when the environment makes the slightest change. Therefore it is difficult to apply offline POMDP to a frequently changing environment. On the other hand, online POMDP planners tackle the state space limitation of offline planners by only computing a local policy for the current belief state. In this way, online planners can significantly reduce the time for policy construction. And since online planners recomputes the partial policy every time after finishing executing an action, they can automatically handle the model change between two time steps. However, one limitation of online planning is the real time constraint: there might not be enough time to construct a good policy.

Chapter 3

Environment Modeling

This chapter consists of two parts. In the first part, we describe our modeling assumptions, including the environment, pedestrians and robot motion (Section 3.1 - 3.3). In the second part, we describe how we organize the world model in a POMDP framework (Section 3.4 and 3.5).

3.1 Environment

We consider two types of obstacles the vehicle faces during the navigation. The first is the static obstacles, which are known a-prior before the navigation. The other is the dynamic obstacles, which are encountered during the runtime.

3.1.1 Maps and Static Obstacles

The static obstacles are represented as an occupancy grid map. Each grid cell of the map is a boolean variable indicating whether this cell is occupied or not. The map can adopt different resolutions for each grid cell. In our case, 0.1m resolution for each grid cell is enough for accurately tracking the world state. This map can be built either from real laser scan or through manual input. A typical laser scan map is shown in Figure 3.1:

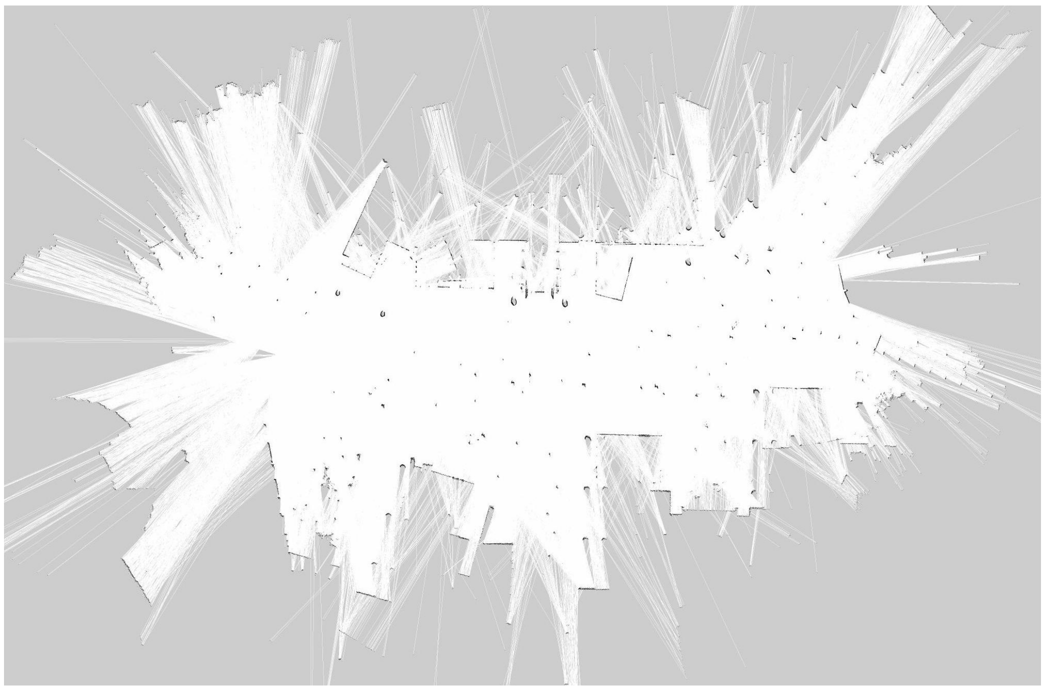


Figure 3.1: Laser scan map of a campus environment

Static obstacles are known before navigation. Those obstacles include walls, trees, curbs, *etc.*. During the task, the static obstacles do not change their positions once the map is built. To avoid those obstacles, we just need to calculate a shortest path in the free space with no obstacles on it. On the contrary, dynamic obstacles are much more difficult to deal with. Dynamic obstacles are moving objects detected when the vehicle is actually running on the road. Those obstacles include pedestrians, other vehicles, and so on. Generally to deal with dynamic obstacles, we may need to do online replanning. Usually pedestrians are the most frequently encountered dynamic obstacles, so they deserve the special attention.

3.1.2 Pedestrians as Dynamic Obstacles

If the environment is static, the problem turns into a simple path planning problem, and the vehicle just needs to maintain a certain speed and sticks to the pre-computed path. However, the real world environment is complex and filled with dynamic elements, most of which come from the pedestrians. Pedestrians' behavior is diverse and subtle yet rational and predictable. Our algorithm focuses on interaction with pedestrians, and we believe that knowledge of the pedestrians' behavior can greatly improve navigation efficiency.

There are several ways to model the pedestrians' behavior. Pattern-based approaches classify the pedestrians' trajectories into several pattern categories [Bennewitz *et al.*, 2005]. However, the walking patterns are actually quite distinctive for each pedestrian and discrete pattern space makes it hard to represent all pedestrians' walking habits.

From a higher level point of view, pedestrians are directed by their final destinations. However, due to the existence of obstacles, their destinations are not reachable through a straight path. According to the previous work [Ikeda *et al.*, 2012], pedestrian usually divide their paths in consecutive straight segments joining their actual positions with the goals. The joint points of those segments are the so-called "subgoals". These subgoals are mostly determined by the nature of the environment (entrance, turn, *etc.*) and common to all pedestrians. In Figure 3.2, there are six subgoals in the map. A pedestrian is walking from the garage to subgoal 1, but there is no straight collision-free path connecting directly to the final goal. Therefore, the pedestrian consecutively passes through intermediate subgoals 5 and 6 to reach the final subgoal 1, and the whole path is thereby divided into three segments. To find the subgoals in the environment, one way is to apply clustering algorithm on the collected pedestrians' trajectory data to retrieve the most likely subgoals in the environment [Ikeda *et al.*, 2012]. In our work, we simplify the process by manually specifying the subgoals after analyzing the structure of the environment and pedestrians' habits in the environment.

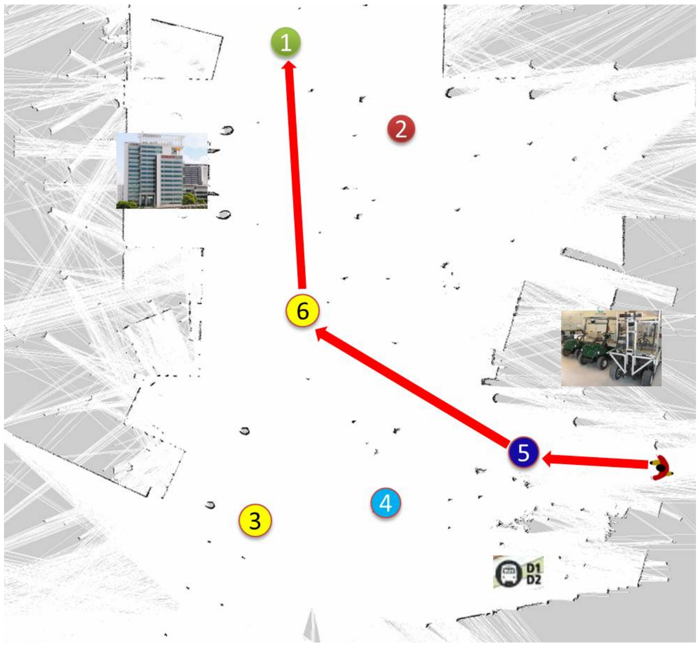


Figure 3.2: Demonstration of the concept of subgoal in the campus environment

On the other hand, Pedestrian usually deviates from a straight path towards their subgoal due to the impact from other pedestrians, vehicles and obstacles [Helbing and Molnar, 1995]. Our model form those possible deviations as noise in the pedestrian's transition model.

3.2 Robotic Motion

The robot model we are considering is a vehicle whose kinematic model is an abstraction of a real world car. The configuration of this robot can be represented as (x, y, w), in which x and y indicate the position on two axes and w shows its orientation. Under the assumption of no slipping, this robot model is a typical non-holonomic system which possesses two-degree control space (vel, str). vel can be controlled through accelerating and braking, and w can be controlled through steering. Nevertheless, it is difficult to do POMDP planning directly in 2-dimension grid. Firstly, either action space or observation space becomes large, making it difficult to solve. Secondly, an accurate motion model of the vehicle is needed to provide good prediction. Thirdly, in this case it is hard for the POMDP planner to form a smooth path, which greatly affects both the passengers and pedestrians' experience.

Due to the above concerns, we assume the car sticks to an a-priori path with no static obstacles on it (see Chapter 5 for extended version). This path can be either generated by a general path planner or through manual input.

Our motion model becomes :

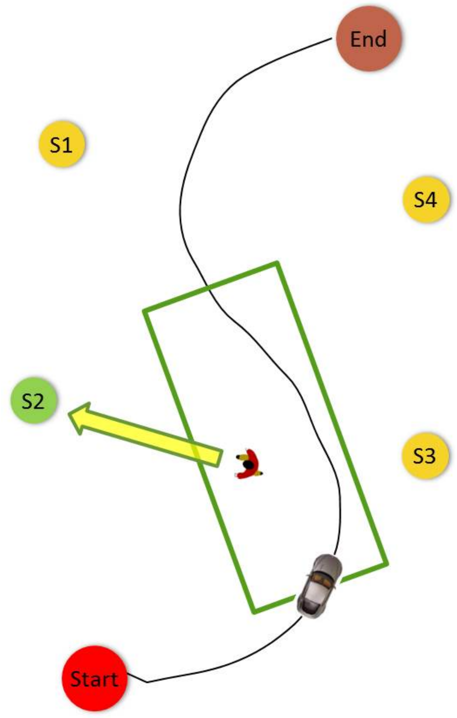
$$v_{t+1} = v_t + a\Delta t + \epsilon_1 \tag{3.1}$$

$$x_{t+1} = x_t + v_t\Delta t + \epsilon_2 \tag{3.2}$$

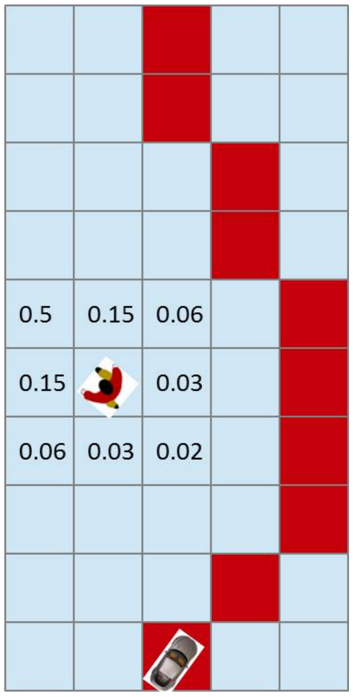
where v is the speed of the vehicle, x is the position index on the 1-dimension path (assuming the points on the path are equally spaced at certain resolution), a is the acceleration speed of the vehicle decided by the controlling action and the system characteristics, Δt is the control period and ϵ_1 and ϵ_2 are the controlling noise decided by the respective system. For our system, we reasonably assume the noise distributions are Gaussian.

3.3 Intention-Aware POMDP Modeling

Our work introduces Partially Observable Markov Decision Process (POMDP) as a model for this motion planning problem. POMDP is a rich probabilistic model which can capture the uncertainties in pedestrians' intentions as well as the uncertainties in controlling and sensing. In addition, POMDP model can easily adapt to the change



(a) global space



(b) local space

Figure 3.3: (a) is the global world model, in which the vehicle is moving from start to the end and the pedestrian is moving to its current subgoal G2. (b) is the corresponding local world model finally processed by the solver. The red grid cells represent the future positions of the vehicle. And the numbers show an example of pedestrian transition probability distribution based on its current subgoal.

in the system specifications by adjusting a few parameters.

3.3.1 Global Space to Local Space

As we described in the previous section, in our model, pedestrians are walking in a relatively large grid space. Large state space could be problematic for most POMDP solvers. Even state-of-the-art online planners can reduce this problem by planning for current belief state, their search could bias towards some irrelevant areas which leads to a useless policy (See Chapter 4 for details). To improve the planning efficiency, we confine the POMDP search within a local planning window that attempts to cover only the most relevant area. Specifically, our local planning window is a $W \times H$ rectangle in which we place the local path segment in the middle, and the vehicle at the bottom-center. The heading of the rectangle is the same as the vector starting from the position of the vehicle to the position m meters further on the path (Figure 3.3a). The window is dynamically shifting forward along with the vehicle every k time steps. Each time step, the pedestrians' positions inside the window will be updated, as well as the vehicle's position with respect to the window. For the current implementation, we do not model the pedestrians entering or exiting the window, therefore the planner assumes pedestrians stay static when they reach the border of the rectangle during the search.

The local path is the part of the global path inside the local planning window. Formally, the local path contains l points $(p_1, p_2, \ldots, p_l)$. Each point p_i is in a 2-dimensional space specifying the position in the local planning window. The local path is an approximation to the trace of the global path segment inside the window, as the red cells in Figure 3.3b. We choose the l points that are approximately equally spaced on the path.

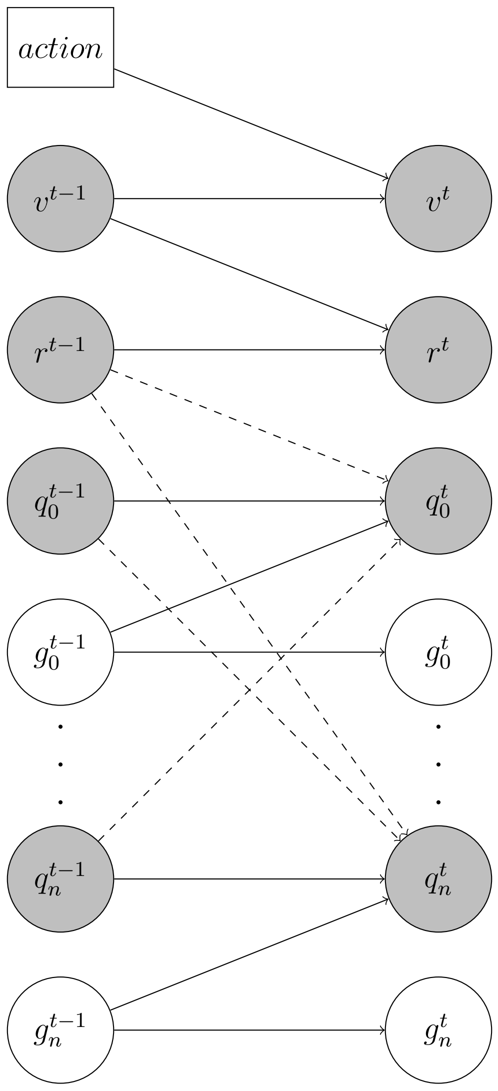


Figure 3.4: The probabilistic network of our model.

3.3.2 State and Observation

After we compute the local window, we discretize both the grid and the path inside the window to a desired resolution which forms the real POMDP state. As in Figure 3.3b, the actual positions are discretized into a $W \times H$ grid, each cell is a square with a side length lm. The grid cells on the local paths are marked as red cells in Figure 3.3b. The solver knows only the pedestrians falling within the window and ignores those outside the window. At the start of every control loop, the planning window will be updated according to the new position of the vehicle so that the visible pedestrians could also be updated accordingly.

The POMDP state consists of several state variables, specified as

$$s = (q_1, g_1, \dots, q_n, g_n, r, v)$$

The dependencies between these state variables are shown in Figure 5.2. We factorize the state into fully observable part $s_x = (q_1, \dots, q_n, r, v)$ which can be sensed accurately, and unobservable part $s_y = (g_1, \dots, g_n)$ which can not be sensed directly [Ong *et al.*, 2010]. Fully observable variables are marked in grey and unobservable variables are marked in white.

Let $(q_1, \dots, q_n)$ be the joint positions of N pedestrians. Each q_i is a 2-dimension integer coordinate (x, y) indicating the pedestrian's position in the planning window. The origin of the coordinate is located at the bottom-left corner of the local window, with the two sides of the window as the x and y axes.

Let $(g_1, \dots, g_n)$ be the intentions of N pedestrians. The value of each g_i can be chosen from the set of K subgoals $(S_1, S_2, \dots, S_K)$, where each subgoal is represented as one grid cell in the global space. Subgoals can be determined using the methods mentioned in Section 3.1.2. For example, in Figure 3.3(a), there are K=4 subgoals and the pedestrian's current subgoal is S_2. Intention g_i can not be observed directly;

it can only be inferred from other state variables.

r is an integer value denoting the vehicle's position on the local path. p_r is the rth point on the local path $(\rho_1, \rho_2, \ldots, \rho_l)$.

v is an integer value denoting the vehicle's speed. Each integer corresponds to a real vehicle speed, and the maximum value is decided by the speed limitation of the vehicle. We discretize the speed to reduce the search space of the planner, as well as avoid over-sensitive control. For example, it is difficult to drive a real vehicle at an accurate speed such as 3.14159m/s. Additionally, for comparison convenience, a discrete model is more general since some POMDP software does not support continuous state representation.

The POMDP observation consists of the following variables:

$$o = (o_{q_1}, \ldots, o_{q_n}, o_r, o_v)$$

where $(o_{q_1}, \ldots, o_{q_n})$ is the joint observation of the positions of N pedestrians, o_r is the observation of the vehicle's position and o_v is the speed of the vehicle. Since our sensors are accurate enough for our experiment (more details in Section 5.1), we can reasonably assume those observations are noiseless. In this case, we can simply generate the observations with an one-on-one mapping directly from the state.

However, joint observation modeling could result in a large observation space, which makes it difficult for the planner to find a good policy. As a rough estimation, the number of observation branches from the current state is $O(9^N)$, where N is the number of pedestrians, and 9 is the number of total transition directions of each pedestrian (see Section 3.3.3 for details). To reduce the growing speed of the number of observations, we can intentionally introduce some observation noises by grouping similar observations together. For example, if the actual sensor resolution of the pedestrian's position is $0.5m \times 0.5m$, the planner can retrieve the observation at

$1.0m \times 1.0m$ resolution. Consider an extreme case in which the sensor does not observe anything at all, so that the planner make decisions purely based on its belief of pedestrians' positions. As the planner is not certain about the pedestrians' positions, we expect it to take more conservative policies. Furthermore, we can dynamically adjust the observation resolution on-the-fly. Specifically, when there are fewer people on spot, we can use an accurate sensing model which gives a more optimal plan; when the number of pedestrians becomes large, we step back a bit with a noisy observation model that delivers a relatively conservative but safe plan.

3.3.3 Action and Transitions

Equations 3.3, 3.4 and 3.5 show the transitions of robot and pedestrian's states between two time steps. All the states are discretized at a reasonable resolution. The $\textsc{Sample}(x, \sigma)$ function samples an output according a Gaussian distribution $N(x, \sigma)$ and discretizes the output to form the next state.

1. *Velocity transition*

   There are three actions: ACCELERATE, DECELERATE, MAINTAIN. ACCELERATE increases one speed level, DECELERATE decreases one speed level, and MAINTAIN keeps the current speed.

$$v_t = \begin{cases} \textsc{Sample}(v_{t-1} + 1, \sigma_v) & \text{if } a = ACC \\ \textsc{Sample}(v_{t-1} - 1, \sigma_v) & \text{if } a = DEC \\ \textsc{Sample}(v_{t-1}, \sigma_v) & \text{if } a = CUR \end{cases} . \tag{3.3}$$

2. *Robot position transition*

The next position of the robot is calculated as:

$$r_t = \textsc{Sample}(r_{t-1} + \textsc{Real}(v_t)\Delta T, \sigma_r) \tag{3.4}$$

ΔT is the time period of the control loop. $\textsc{Real}(x)$ maps one-to-one from a discretized speed v to a real-world speed. The mapping function has to satisfy the acceleration constraint of the real vehicle $(\textsc{Real}(v_t) - \textsc{Real}(v_{t-1})) = \hat{v}\Delta T$ ($\hat{v}$ is the acceleration determined by the vehicle's controller characteristics) and the movement constraint $|p_{r_t} - p_{r_{t-1}}| = (\textsc{Real}(v_t) - \textsc{Real}(v_{t-1}))\Delta T$ ($|p_{r_t} - p_{r_{t-1}}|$ is the displacement on the path between two time steps).

3. *Pedestrian position transition*

The pedestrian's next position is calculated as following:

$$\begin{aligned} \theta_i &= \textsc{Sample}(\theta_0(g_i), \sigma_p) \\ q_{i,x}^t &= q_{i,x}^{t-1} + v_p \cos(\theta_i) \\ q_{i,y}^t &= q_{i,y}^{t-1} + v_p \sin(\theta_i) \end{aligned} \tag{3.5}$$

First, we sample the heading of the ith pedestrian from the Gaussian distribution $N(\theta_0(g_i), \sigma_p)$ with the mean movement direction towards the subgoal of ith pedestrian g_i, and σ_p represents the noise level of the pedestrian's movement. A small value of σ_p indicates a high probability that pedestrian is following the direction towards his subgoal in the next step, vice versa. After we get the sampled heading θ_i, we calculate the pedestrian's next coordinate $(q_{i,x}^t, q_{i,y}^t)$ based on θ_i and the walking speed v_p of the pedestrian. For simplicity, we assume pedestrian moves to adjacent eight grid cells or stay stationary in the next time step, so that its real velocity has to satisfy the constraint $v_p \leq C \times \Delta T$, where

ΔT is the control period, and each grid cell is a Cm $\times$ Cm square. We need to set proper parameters ΔT and C to match with the pedestrian's normal walking speed, which is around 1.0m/s - 1.5m/s.

An example is shown in Figure 3.3(b), the direction to the pedestrian's goal is indicated by the arrow in Figure 3.3(a). Among the eight adjacent cells, those with closer directions to the goal get higher transition probabilities.

3.3.4 Reward Function

In POMDP, a reward function is the utility function we want to optimize for this sequential decision problem modeled with POMDP. The output of this function is determined by various factors, such as goal incentive R_{goal}, crash penalty R_{crash}, action penalty R_{action} and horizon penalty H, as in equation 3.6.

$$R = R_{crash} + R_{goal} + R_{action} + H \tag{3.6}$$

Firstly, a penalty R_{crash} is imposed for crashing (equation 3.7). We set up two crash criterion. The first one is a higher penalty for a real crash situation when there exists pedestrian in a small window $w_1 \times h_1$, calculated by multiplying the penalty term $-D$ with the vehicle's speed v; the second one is a penalty $-D$ when there is pedestrian in a larger window $w_2 \times h_2$ and the vehicle has a speed more than Bm/s. The reason we add the second penalty term is that when there is pedestrian nearby, driving in a high speed could scare the pedestrian.

$$R_{crash} = \begin{cases} -D \times v & \text{if } \exists i,\ |q_{i,x} - p_{r,x}| < w_1 \text{ and } 0 < q_{i,y} - p_{r,y} < h_1 \\ -D & \text{if } \exists i,\ v > B\text{m/s and } |q_{i,x} - p_{r,x}| < w_2 \text{ and } 0 < q_{i,y} - p_{r,y} < h_2 \\ 0 & \text{else} \end{cases}. \tag{3.7}$$

Secondly, a goal incentive $+C$ is imposed to attract the vehicle towards its destination, and this could be done by assigning a positive value when vehicle reaches some point r_g on the local path, as in (3.8)

However, since our model assumes that pedestrians do not walk out of the local planning window, there will be pedestrians aggregated at the end of the window after several steps. If we place the goal at further part on the path, the attraction of the goal would be canceled by the risk of crashing pedestrians. On the other hand, if we place the goal at closer part on the path, the planning horizon becomes very short so that the vehicle just accelerates all the way.

$$R_{goal} = \begin{cases} +C & \text{if } r >= r_g \\ 0 & \text{else} \end{cases}. \tag{3.8}$$

$$R_{action} = \begin{cases} -A_1 & \text{if action}=ACC \\ -A_2 & \text{if action}=DEC \\ -A_3 & \text{if action}=CUR \end{cases}. \tag{3.9}$$

Besides the rewards and penalty above, we can add other components to further tune the behavior of the vehicle. We can impose a small negative value $-H$ on each horizon to push the vehicle to accelerate. Moreover, to take into account the smoothness requirement, we can impose penalties for acceleration and deceleration as in equation 3.9 (see further details in chapter 5).

Chapter 4

Planning Techniques

In the previous chapter, we described the world model and probabilistic model used for planning. In this chapter, we first describe the technical details of the state-of-the-art online POMDP planner we use. Then we are going to compare with several other frequently used methods to solve this navigation problem, including reactive planning, offline POMDP planning, and other probabilistic methods. In the last part, we present the experiment results in simulation to show the advantage of our approach.

4.1 Online POMDP Planning

As reviewed in chapter 2, online POMDP solver plans only for the current belief state and considers the horizons in the near future. Firstly, the planner starts on an initial belief, and it searches for a near-optimal action a at each time step based on the current belief state and world model. Secondly, the agent executes action a through actuators and receives an observation o through sensors. Afterwards, the belief update module updates the current belief based on the observation o received. The process then repeats.

In the following sections, we will address the two important components of a complete online planning process, which are *belief tracking* and *online tree search*. In the belief tracking section, we will introduce the general concept of belief update and present our approach to construct an appropriate particle filter for efficient belief update. In the online tree search section, we will explain the working mechanism of the state-of-the-art online POMDP solver DESPOT and elaborate how we apply it to solve our navigation problem.

4.1.1 Belief Tracking

The first component of online POMDP controller is *belief tracking*. As stated previously, POMDP maintains a belief distribution b over the state space S. The exact belief update process can be written as:

$$b'(s') = \eta Z(s', a, o) \sum_{s \in S} T(s, a, s') \tag{4.1}$$

where η is a normalizing factor over all possible observations. The majority of POMDP solvers use exact belief update process which needs to iterate through all the states to calculate the posterior probability on a single state. As in Section 3.2.2, our state is defined as a combination of robot vehicle's state and all the pedestrians' positions. The dimension of the joint state space is $2N + 2$, N being the number of pedestrians. As N becomes large, the total number of states exponentially increases, which makes it computationally expensive to execute exact belief update on the belief distribution over all the states.

An alternative belief update approach is to approximate the belief by a set of K particles:

$$B_t := (s_t^1, s_t^2, \ldots, s_t^K)$$

Each particle s_t^i (with $1 \leq i \leq K$) corresponds to a sampled POMDP state, which can be represented by a set of variables:

$$s_t^i = (q_1, g_1 \ldots, q_n, g_n, r, v)$$

Each state can be divided into fully observable part $s_{x,t}^i = (q_1, q_2, \ldots, q_n, r, v)$ which are the physical state of the pedestrians and vehicle, and unobservable part $s_{y,t}^i = (g_1, g_2, \ldots, g_n)$ which are the intentions of the pedestrians [Ong *et al.*, 2010]. K is a large number depending on the dimension of the belief space, in our case, roughly the number of pedestrians.

Particle filtering can be used to update the belief distribution represented by these particles. For large K, the likelihood for a state hypothesis s_{t+1} to be included in the particle set B_{t+1} shall be a reasonable approximation to its Bayes filter posterior:

$$s_{t+1}^i \sim p(s_{t+1}|s_t, a_{t+1}, o_{t+1}) \tag{4.2}$$

a_{t+1} is the control action taken for the current step, which can be chosen from ACCELERATION, DECELERATION, MAINTAIN. $o_t = (o_{q_1}, \ldots, o_{q_n}, o_r, o_v)$ is the real observation received for the current step, which can be mapped one-to-one to the fully observable part of the state $s_{x,t+1}^i$. As a consequence of equation 4.2, the denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region. According to equation 4.2, particle filter algorithm constructs the next belief B_{t+1} from B_t one time step earlier. The work of [Thrun *et al.*, 2005] provides a common framework of particle filter algorithm. Next we will describe the specific particle filtering algorithm we use with modifications for our navigation problem, as in Algorithm 1.

Algorithm 1 Particle Filtering

```
BeliefUpdateParticle($B_t, a_{t+1}, o_{t+1}$)
 1: $\bar{B}_{t+1} = B_{t+1} = \emptyset$
 2: $N_{\mathit{eff}} = 0$
 3: for $i = 1$ to $K$ do
 4:     $s^i_{y,t+1} = s^i_{y,t}$
 5:     $s^i_{x,t+1} = \text{ObsToState}(o_{t+1})$
 6:     $s^i_{t+1} = (s^i_{x,t+1}, s^i_{y,t+1})$
 7:     $w^i_{t+1} = p(o_{t+1}|s^i_{t+1})w^i_t$
 8:     $\bar{B}_{t+1} = \bar{B}_{t+1} + (s^i_{t+1}, w^i_{t+1})$
 9: end for
10: Normalize ($\bar{B}_{t+1}$)
11: for $i = 1$ to $K$ do
12:     $N_{\mathit{eff}} = N_{\mathit{eff}} + (w^i_{t+1})^2$
13: end for
14: if $\frac{1}{N_{\mathit{eff}}} < 0.1K$ then
15:     for $i = 1$ to $K$ do
16:         add randomly initialized new particle $(s^{K+i}_{t+1}, \frac{1}{K})$
17:     end for
18:     for $i = 1$ to $K$ do
19:         draw $m$ with probability $\propto w^m_{t+1}$
20:         add particle $(s^m_{t+1}, 1)$ to $B_{t+1}$
21:     end for
22: else
23:     $B_{t+1} = \bar{B}_{t+1}$
24: end if
25: return $B_{t+1}$
```

1. *Generating New Particles.*

   Lines 4-6 generate a new particle s^i_{t+1} from the ith particle in the current particle set. We copy the unobservable part $s^i_{x,t}$ directly to the new particle $s^i_{x,t+1}$ (line 4), since our model assumes the intention of the pedestrian does not change. In line 5, we generate the fully observable part by mapping from the current observation o_{t+1} using the function ObsToState. In line 6, we combine the two parts to form a new particle in time step $t + 1$.

2. *Importance Factor.*

   Line 7 calculates the *importance factor* w^i_{t+1} for each particle s^i_{t+1}. Importance factors are used to incorporate the measurement o_{t+1}, interpreted as the weight of each particle $w^i_{t+1} = p(o_{t+1}|s^i_{t+1})$. The weighted particle set together can approximate the Bayes filter posterior. In our problem, the importance factor can be calculated as $w^i_{t+1} = p(o_{t+1}|s^i_{t+1}) = p(o_{q_{t+1}}, o_{r_{t+1}}, o_{v_{t+1}}|q^i_{t+1}, r^i_{t+1}, v^i_{t+1})$. Since we assume each pedestrian's new position only depends on its previous position, the formula can be rewritten as

$$w^i_{t+1} = \prod_{j=1}^{n} p(q^i_{j,t+1}|q^i_{j,t}) \tag{4.3}$$

   However, our model assumes each pedestrian only moves to its adjacent grid cells. If the jth pedestrian moves to at an unexpected place, $p(q^i_{j,t+1}|q^i_{j,t})$ would become zero for all particles $1 \leq i \leq K$, so that the joint transition probabilities (importance weights) become zero too. In this case, we lost all the information previously tracked due to unexpected observation. To fix this problem, we can assign a small transition probability instead of zero for an unexpected pedestrian movement.

3. *Adding Diversity.*

Lines 11-22 is the tricky part of our particle filter. Since we model all the pedestrians' intentions in a joint state, the total number of possible states is $|g|^N$ where $|g|$ is the total number of subgoals and N is the total number of pedestrians. When N is large, it is difficult to accurately represent the belief distribution well with $K = 2000$ particles. Furthermore, since there is no intention transition in the particles, the number of particles representing each intention is fixed after initial sampling. Consider an extreme unlucky case that all the sampled states have subgoal 1 for all pedestrians, then there is no way for the approximate belief of intentions to converge to the true distribution afterwards. To deal with this problem, we add diversity to the particle set by adding random states and resampling. Line 10 normalizes the weight distribution of the particle set, after which the weights of all particles sum up to 1. Line 9 sums up the square of each particle's normalized weight as $N_{\mathit{eff}} = \sum_{i=1}^{K}(w_{t+1}^{i})^2$. After that, we adopt the method in [Doucet, 2001] which computes $\frac{1}{N_{\mathit{eff}}}$ as an indicator of the variance of the distribution. A relative small value of $\frac{1}{N_{\mathit{eff}}}$ indicates that most weight are concentrated on a few particles. The least value of this term is 1.0 when only one particle has weight 1.0 while all the other particles have zero weight. When $\frac{1}{N_{\mathit{eff}}} < 0.1K$ (lines 15-21), we add random states to the particle set followed by a resampling procedure.

Lines 15-17 adds K new particles to $\bar{B}_{t+1}$: the observable part is constructed according to the real observation; the unobservable intentions are randomly initialized. The weight of each particle is equal and all weights sum up to 1.0, equals to the total weights of old particles.

Lines 15-20 does *importance sampling* by drawing with replacement K particles from the temporary particle set $\bar{B}_{t+1}$, where the probability of drawing each particle is given by its importance weight w_{t+1}^{i}. Resampling can reduce the variance of the particle set by removing low-probability particles. However, it can cause the bias of the particle set as an estimator of the true belief increases. To tackle this problem, firstly we control the frequency of resampling by the measurement of effective particles (line 14). Secondly, we use low-variance sampling technique [Thrun *et al.*, 2005] which selects each sample in a sequential stochastic process instead of independent selecting.

4.1.2 Online Tree Search

The common way of online POMDP planning is to construct a *belief tree*, with the current belief b_0 at the root of the tree, and performs tree search for a policy π that maximizes the expected total discounted reward $V_\pi(b_0)$. Each node of the tree represents a belief, which has $|A|$ action edges. Each action edge further branches into $|O|$ observation edges. Every node and its child satisfies $b' = \tau(b, a, o)$, where b' is the updated belief of b after executing an action $a \in A$ and receiving an observation $o \in O$. At each internal node, the best action is chosen by computing the maximum value of action branches and the average value of observation branches. The result tree forms an approximately optimal value $V_{\pi^*}(b_0)$ and its policy π^*.

When the number of action and observation branches becomes large, the size of the belief tree grows exponentially making the tree search infeasible. State-of-the-art online planners such as POMCP [Silver and Veness, 2010] and DESPOT [Somani *et al.*, 2013] use *sampling* technique to approximate the full search tree. And instead of requiring explicit representation of a large transition and observation table, they only require a generative model which samples a successor state, observation and reward given the current state and action. For example, POMCP uses a Monte-Carlo sampling approach and utilizes UCT algorithm [Kocsis and Szepesvári, 2006] to improve the action selection in the tree. However, UCT is sometimes overly greedy and suffers from the worst-case $\Omega(\exp(\exp(\ldots\exp(1)\ldots)))$[1] bound.

[1]Composition of $D - 1$ exponential functions.

Instead of doing Monte-Carlo simulation like POMCP, DESPOT runs a *deterministic simulation* from a small set of sampled *scenarios*. Each scenario contains the combination of a sampled state and random number sequence, represented as $\phi = (s, \phi 1, \phi 2, \dots)$. A deterministic simulative model is a function $S \times A \times R \rightarrow S \times Z$, such that if a random number ϕ is distributed uniformly over [0,1], then $(s', o') = g(s, a, \phi)$ is distributed according to $p(s', z'|s, a)$. When we simulate this model for an action sequence $(a_1, a_2, a_3, \dots)$ under a scenario $(s_0, \phi_1, \phi_2, \dots)$, the simulation generates a trajectory $(s_0, a_1, s_1, z_1, a_2, s_2, z_2, \dots)$, where $(s_t, z_t) = g(s_{t-1}, a_t, \phi_t)$ for $t = 1, 2, \dots$. The simulation trajectory traces out a path from the root of the standard belief tree.

The structure of the DESPOT search is shown in Algorithm (2). More details can be found in [Somani *et al.*, 2013]. The algorithm contains the following steps.

1. *Tree Initialization.*

   In the SEARCH(B_t) function, the algorithm first samples K scenarios from the current particle belief set B_t. Then the search tree is constructed with a single node b_0 at the root, containing the initial K scenarios Φ_{b_0}.

2. *Action Selection.*

   The tree search process contains a sequence of trials and backups before reaching time limit. The TRIAL(b) function traces a path from the root node b_0 to a leaf node. The algorithm chooses the successive nodes following the similar idea as the HSVI selection heuristic [Smith and Simmons, 2004]. For every belief node b in T, DESPOT maintains an upper bound $U(b)$ and a lower bound $L(b)$ on $\hat{V}_{\pi^*}(b)$, which bounds the value of the optimal policy π^* for b under the set of scenarios $\mathbf{\Phi}_b$. Similarly DESPOT maintains bounds $U(b, a)$ and $L(b, a)$ on the Q-value $Q_{\pi^*}(b, a) = \frac{1}{|\mathbf{\Phi}_b|} \sum_{\phi \in \mathbf{\Phi}_b} R(s_\phi, a) + \gamma \sum_{b' \in \text{Children}(b,a)} \frac{|\mathbf{\Phi}_{b'}|}{|\mathbf{\Phi}_b|} \hat{V}_{\pi^*}(b')$. Here we estimate the probability of reaching one observation branch b' after executing action a by the number of scenarios in that particular branch b' divided by

Algorithm 2 Online DESPOT Tree Search

```
SEARCH($B_t$)
 1: construct $K$ random scenarios $\Phi_{b_0}$ for $b_0$
 2: Initial tree $T$ with only one node $b_0$ at the root
 3: while time remaining do
 4:     b= TRIAL($b_0$)
 5:     Backup lower and upper bounds for nodes on the path from b to b0
 6: end while
 7: Compute a regularized policy $\pi^*$ from T
TRIAL($b$)
 1: if $depth(b) > D$ then
 2:     return b
 3: end if
 4: if b is a leaf node then
 5:     Expand $b$ one level deeper
 6:     Construct initial upper and lower bounds on $b$
 7: end if
 8: $a^* = \arg\max_{a \in A} U(b, a)$.:
 9: $z^* = \arg\max_{z \in Z_{b,a^*}} \text{WEU}(\tau(b, a^*, z))$.
10: $b = \tau(b, a^*, z^*)$.
11: if $\text{WEU}(b) \geq 0$ then
12:     return TRIAL ($b$)
13: else
14:     return $b$
15: end if
16:
```

the total number of scenarios in node b as $\frac{|\mathbf{\Phi}_{b'}|}{|\mathbf{\Phi}_b|}$. The same estimation is used for calculating weighted excess uncertainty and upper and lower bound. The algorithm chooses the action branch a^* that maximizes $U(b, a)$ for the current node b and then chooses the observation branch z^* that maximizes the *weighted excess uncertainty* at the child node $b' = \tau(b, a^*, z)$: $\text{WEU}(b') = \frac{|\mathbf{\Phi}_{b'}|}{|\mathbf{\Phi}_b|}$ where $\text{excess}(b') = U(b') - L(b') - \epsilon\gamma^{-depth(b')}$ [Smith and Simmons, 2004] and ϵ specifies the desired gap between the upper and lower bounds at the root b_0. In our case, we set ϵ to be $k(U(b_0) - L(b_0))$ where $k \in (0, 1)$. The trial ends when the weighted uncertainty $\text{excess}(b) < 0$, therefore a larger k generally increases the length of each trial.

3. *Expand Fringe Node*

   When the trial reaches a leave node b in the tree, the node will be expanded one step further. The algorithm executes every action in the action space from each scenario in $\Phi(b)$. For a scenario $\phi_i = (s_i, \phi_d, \phi_{d+1}, \dots)$, after executing action a, a new state s'_i and observation o' will be generated according to the deterministic simulation model $(s'_i, o') = g(s_i, a, \phi)$. Then a new branch will be created from that observation branch, containing the new state and remaining random numbers $(s'_i, \phi_{d+1}, \phi_{d+2}, \dots)$.

4. *Initialize Upper and Lower Bounds*

   Initial upper and lower bounds are calculated on the leave nodes. The algorithm calculates the initial upper bound of a node by averaging the *fringe upper bound* on each scenario in the node. The fringe upper bound is the maximum reward that can be obtained on a scenario by following any policy with respect to its random number sequence. In our vehicle navigation problem, we calculate an estimation of the fringe upper bound value by assuming the vehicle is driving at its highest speed with no pedestrians around, formally as:

$$UB_{fringe}(s) = R_{goal}\gamma^{(p_g - p_r)/v_{max}} \tag{4.4}$$

where R_{goal} is the goal incentive specified in the reward function, $p_g - p_r$ is the distance from the vehicle's current position to the local goal, v_{max} is robot's maximum speed, and γ is the discount factor.

To calculate the initial lower bound at the leave node, the algorithm simulates a *default policy* for l steps under the scenarios Φ_b and compute the average discounted return. *default policy* π_0 is constructed as a mapping from the belief which is approximated by the particles to action space : $B \to A$. In our vehicle navigation problem, we use a reactive default policy. The policy maintains two safety windows. If there is pedestrian in the larger window, the vehicle will decelerate to a low speed and then cruise with this speed. If there is pedestrian in the smaller window, the vehicle will decelerate until it stops. In all the other cases, the vehicle will simply accelerate. Since DESPOT keeps improving on the lower bound policy through tree search, we can expect the resulting tree policy to be better than a purely reactive policy.

5. *Backup*

Finally the algorithm traces the path backward to the root and performs backup on both the upper and lower bounds at each node along the way. For the lower-bound backup,

$$L(b) = \max_{a \in A}\left\{ \frac{1}{|\boldsymbol{\Phi}_b|} \sum_{\phi \in \boldsymbol{\Phi}_b} R(s_\phi, a) + \gamma \sum_{z \in Z_{b,a}} \frac{|\boldsymbol{\Phi}_{\tau(b,a,z)}|}{|\boldsymbol{\Phi}_b|} L(\tau(b,a,z)) \right\}. \tag{4.5}$$

Basically the algorithm first calculates the lower bound reward on each action a by averaging over $Z_{b,a}$, which is the set of observations can be received on all scenarios in $\mathbf{\Phi}_b$. Then we choose the maximum lower bound reward between all the actions as the lower bound value for current node b. The upper bound backup is calculated in the same way by substituting upper bound for lower bound in equation (4.5).

6. *Regularization*

   After enough trials and backups, the search tree contains a near-optimal policy for the K scenarios. However, the chosen policy computed on the K scenarios may not perform as well on the other scenarios. The potential overfitting issue can be reduced by *regularization* technique which takes the size of the policy tree into account during the policy evaluation process: the smaller the policy size is, the higher probability it is adopted. Regularization is implemented with a tree pruning procedure after tree search (line 7), which computes a regularized policy from the policy tree in linear time of the tree size.

In conclusion, the key idea of DESPOT algorithm is to approximate the *regularized optimal policy* on a small set of sampled *scenarios*. According to the proof in [Somani *et al.*, 2013], it only requires a number of scenarios equal to the size of the optimal policy tree to find a good policy, which is a significant improvement than POMCP's worst case convergence time.

4.2 Other Planning Techniques

Our framework provides a systematic approach of a vehicle navigating in a dynamic environment with uncertainties. In addition, our model dynamics can be specified on-the-fly during the navigation process without requiring a-priori planning. In the following, we compare our approach with three other classes of planning techniques, which are reactive planning, offline POMDP planning and other probabilistic methods.

4.2.1 Reactive Planning

Consider the most intuitive way of driving, which is just to stop when there are obstacles nearby and go when clear. A better way is to predict a few steps ahead and get the best action by forward search based on the vehicle's state and the kinematic model of the vehicle. Dynamic Window Approach (DWA) [Fox *et al.*, 1997] and Virtual Bumper [Schiller *et al.*, 1998] are two commonly-used algorithms in this category.

As a more technical recap of the previous review, DWA basically wants to optimize an objective function written as:

$$G(v, w) = \sigma(\alpha \cdot heading(v, w) + \beta \cdot dist(v, w) + \gamma \cdot vel(v, w)). \tag{4.6}$$

v and w are the sampled linear and angular velocities. *heading(v,w)*, *dist(v,w)* and *vel(v,w)* represent the distance to the goal, the distance to the obstacles and the future vehicle speed simulated from the current state with sampled velocity v and w.

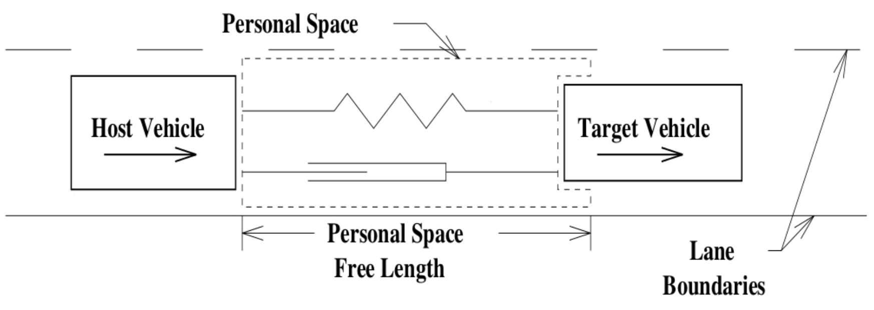


Figure 4.1: Demonstration of the concept of 1-dimension virtual bumper [Schiller *et al.*, 1998]

Virtual Bumper [Schiller *et al.*, 1998] assumes that there is a virtual spring between the vehicle and obstacles. A virtual spring force is computed based on the distance and relative velocity between the vehicle and obstacles. Then it calculates the result speed from the force computed. One practical implementation is like this:

$$V_x = \alpha(P_x - X_{\mathit{off}}) \tag{4.7}$$

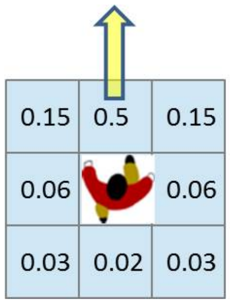
$$V_y = \beta(P_y - Y_{\mathit{off}} - \gamma * A * V) \tag{4.8}$$

A, V are the vehicle's current heading angle and speed. X_{off} and Y_{off} are the respective buffering zones in two directions. P_x and P_y are the obstacles' positions relative to the vehicle in two directions. V_x and V_y are the result speed in two directions, where V_x is the velocity in the same direction as the car's heading, and V_y is the orthogonal direction.

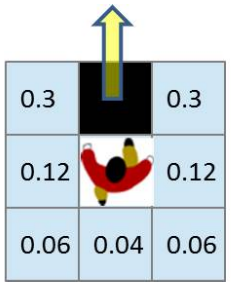
Reactive Planning is usually simple and easy to implement. However, this kind of planning only works well when the agent is omniscient of the environment. In our problem scenario, an important issue is that the intention of the pedestrian is not fully observable to the agent. To deal with this partial observability, we need to model it with our knowledge and infer the unobservable part through the model every step. Thus, we move to a more powerful probabilistic modeling technique POMDP.

4.2.2 Offline Planning and its Limitations

As stated in Chapter 2, offline POMDP planners generally consume much longer time for the policy construction than online planners as they compute the policy for every belief state. In this section, we will explain the difficulty applying offline planning methods to our problem. Basically the difficulty comes from the possible change in the environment which consequently changes the POMDP transition model, so that the



(a) no obstacle



(b) obstacle ahead

Figure 4.2: Comparison of the pedestrian's transition probability with and without obstacles.

expired policy may not be optimal anymore for the new model. From an orthogonal aspect, it would also become infeasible to solve if we extend our model to take into account the possible changes (*e.g.* model the changes as uncertainties). Additionally, offline POMDP planners also have difficulty dealing with the large state space caused by a large pedestrian crowd, and therefore approximations are necessary.

4.2.2.1 Environment change

The environmental change can be caused by the appearance of unexpected obstacles and the shift of the planning window.

- *Unexpected obstacles.* Unexpected obstacles could affect the pedestrians' behavior. As shown in Figure 4.2, the pedestrian's intention is moving straight ahead, so the grid cell in front has the largest transition probability of 0.5. However, after that cell is blocked by some obstacle, the pedestrian is unable to move to that cell and the transition probability becomes 0 instead. As a result, the reduced transition probability will be evenly distributed to other grid cells, and the change in the pedestrian's transition model can further affects the decision of the planner.

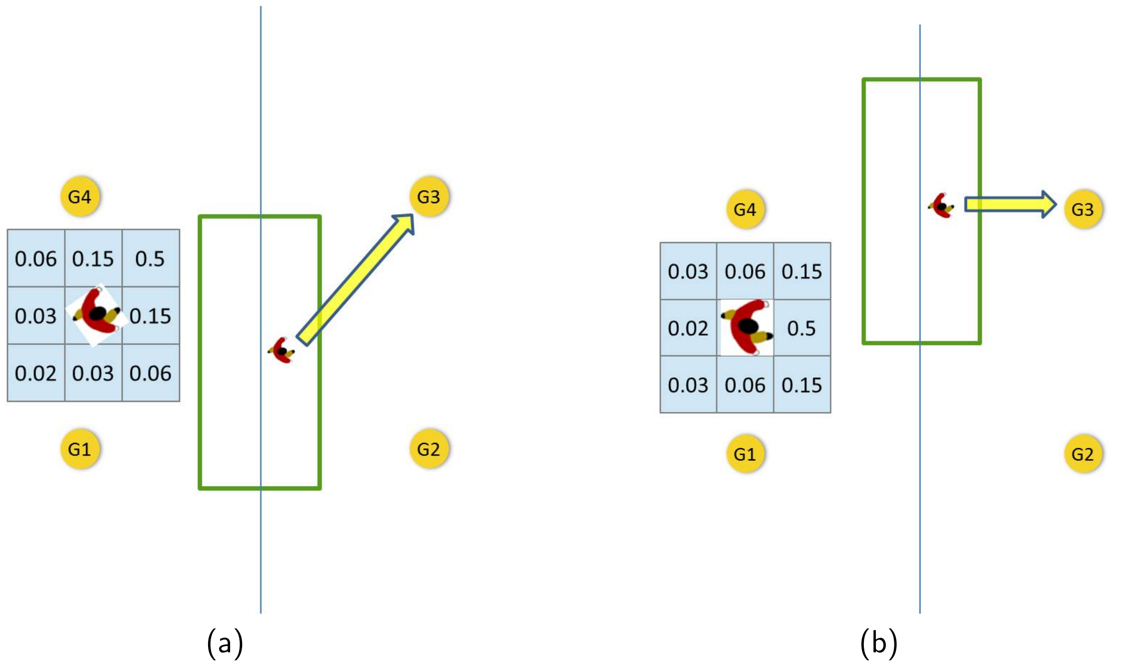


Figure 4.3: The difference in pedestrian's transition probability at the same local position

- *Planning window shifting.* As described in the previous part, we map the subgoals and path from the global map to the local planning window for calculation of the transition model. Therefore, different positions of the planning window result in different transition models. As in the Figure 4.3, even though the pedestrian's local position with respect to the window remains the same, the transition probabilities for the pedestrian changes due to the change of his direction to the goal. In a same way, the transition model of the vehicle can be affected by the update of the local path segment.

For the above problems, it is also unlikely to model the changes a-priori with an enlarged state space. As a rough estimation, there are 2^{WH} different configurations of unexpected obstacles, where WH is the number of grid cells in the window. Multiplied by this factor alone would result in a huge state space infeasible to solve. Therefore, to still run an offline planner, we have to sacrifice model accuracy and use a pre-defined model. In the experiment in Section 4.4, we make the assumptions for the offline planner that there is no obstacle in the window, goals are located at the four corners of the window, and the local path is always a straight lane.

4.2.2.2 Multiple Pedestrians

As described in section 3.3, we model all the pedestrians in a joint state. This joint modeling results in a state space of size $(W \times H \times H \times K)^N \times |v|$ where N is the number of pedestrians, W, H are the two side length of the planning window, K is the number of subgoals, and $|v|$ is the number of speed levels. This is definitely infeasible for any offline planner to solve when the number of pedestrians gets large. A quick hack is to approximate by solving single pedestrian problem and initiate a controller for each pedestrian. Then the planner combines each controller output and chooses the action that generates the lowest speed, sorted as DECELERATE, MAINTAIN, and ACCELERATE, from slow to fast. Yet, it is worth noting that the "slowest" action is not necessarily the safest one. When a large crowd of pedestrians approach, sometimes the best policy is to speed up and overtake rather than move slowly and wait for the pedestrians to approach. In addition, since the DECELERATE action cannot stop the vehicle immediately, the vehicle may end up crashing as well. Besides the safety concerns, this approximation tends to generate too many DECELERATE actions when dealing with large pedestrian crowd in the real world experiment, making the ride inefficient and unsmooth.

4.2.3 Other Probabilistic Methods

Recent state-of-the-art probabilistic models such as Belief Roadmap [Prentice and Roy, 2011] and LQG-MP [Van Den Berg *et al.*, 2011] also account for uncertainties in controlling and sensing, and they model uncertainties as Gaussian distributions.

However, in the real world some of the uncertainties such as pedestrian's intentions do not obviously follow Gaussian distributions. Furthermore, these models assume that the environment is static and thus failing to account for dynamic elements such as pedestrians and the traffic. On the contrary, our model makes no assumption of the distribution types of the uncertainties' and consider for the dynamicity of the environment. Specifically, we model pedestrians' intentions as subgoals and take into consideration the pedestrians' interaction with the surrounding environment.

4.3 Experiments in Simulation

In the previous sections, we have introduced our technique approach to the problem of vehicle navigating and interacting with pedestrians in a densely populated environment, from the modeling aspect and the planning aspect. In this section, we are going to apply our approach on three typical environments in simulation, and compare the performance between our approach and other commonly used approaches.

4.3.1 Experimental Setup

We present three commonly encountered navigation environments, from simple to complex, shown in Figure 4.4, Figure 4.5 and Figure 4.6. The circles on the three maps are the subgoals of the pedestrians. During one simulation run, each pedestrian is initialized on a random position with a random subgoal. If one pedestrian reaches its subgoal, it will be randomly reinitialized on a new position with a new subgoal, so that there are constantly N pedestrians on the map. The first environment is a *straight lane*, where the navigation path is straight in the middle with six subgoals placed on two sides. This environment intends to explore the change of the model dynamics affected by the change of relative goal positions when the vehicle is moving forward. The second environment is a *roundabout*, where there is a curve path from bottom to top with five goals placed on the borders. This experiment environment shows the ability of the online algorithm to adapt to an arbitrarily curved path, while offline planners have to approximate by computing different policies for different lanes. The third environment is *UTown plaza*, which is a realistic environment constructed from information gathered from the LIDAR, and the subgoals are all meaningful real world spots such as bus stops and restaurant entrance.

The planning window is a 7m×15m grid, each grid cell being a 1m×1m square. The size of the window is chosen to allow enough space for the vehicle to brake in time while not consume too much computation time. The control frequency is set to 1hz, as the vehicle is driving in a relatively low speed. The vehicle speed is discretized into three levels: $0m/s$,$1m/s$,$2m/s$. Other detailed parameters are listed in Table 4.1.

We make comparisons between two online algorithms DESPOT and POMCP and two offline algorithms MOMDP and QMDP. QMDP [Littman *et al.*, 1995] is an approximation of POMDP by assuming that the states become fully observable after one step of control. In our simulation experiments, QMDP computes its MDP policy on the same model as MOMDP. Additionally a reactive planner and a random planner are provided as baseline comparison algorithms. The random planner chooses one action randomly each time step to provide a lower bound estimation of the performance. The reactive planner is constructed consistently with the intuition behind human driver's behavior and similar to the method used in Singapore-MIT Alliance's autonomous golf cart which has been tested in the NUS campus for a long time [Chong and others, 2011]. Basically this planner maintains two safety windows. If the pedestrian is inside the smaller window, the vehicle will brake. If the pedestrian is inside the larger window, the vehicle will maintain a low speed. DESPOT and POMCP are each given 1 second per action online computation time which is the same as the control period 1m/s. MOMDP and QMDP are given enough time to compute their polices offline until the policies converges.

Table 4.1: Parameters

Pedestrian noise	0.5
Robot motion noise	0.6
Robot speed noise	0.6
Robot motion noise	0.6
Goal incentive	500
Crash penalty	-1000$\times$speed
High speed penalty	-1000
Horizon penalty	-1
Speed change penalty	-10
Discount factor	0.95

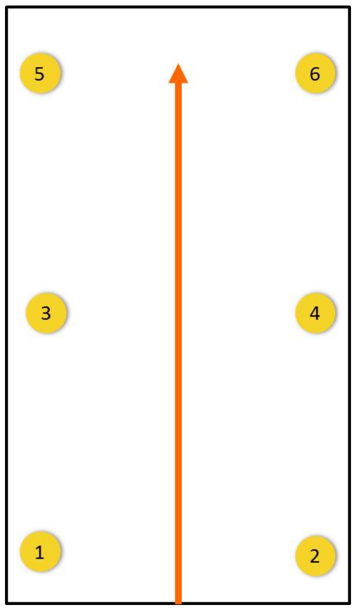


Figure 4.4: Straight lane environment

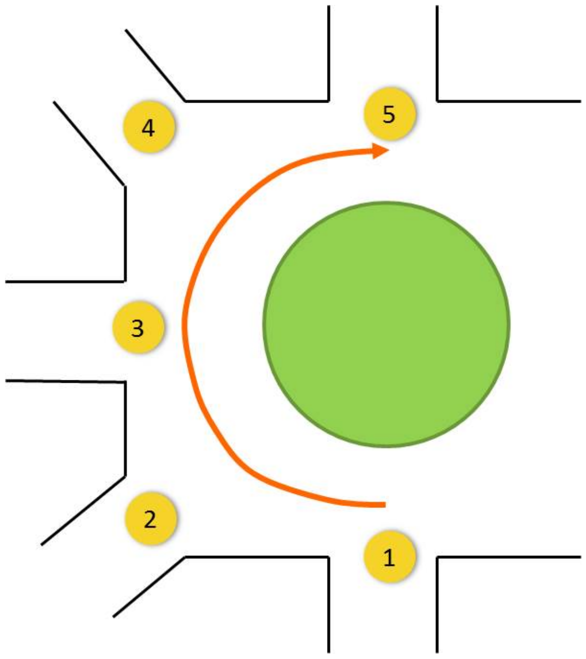


Figure 4.5: Straight lane environment

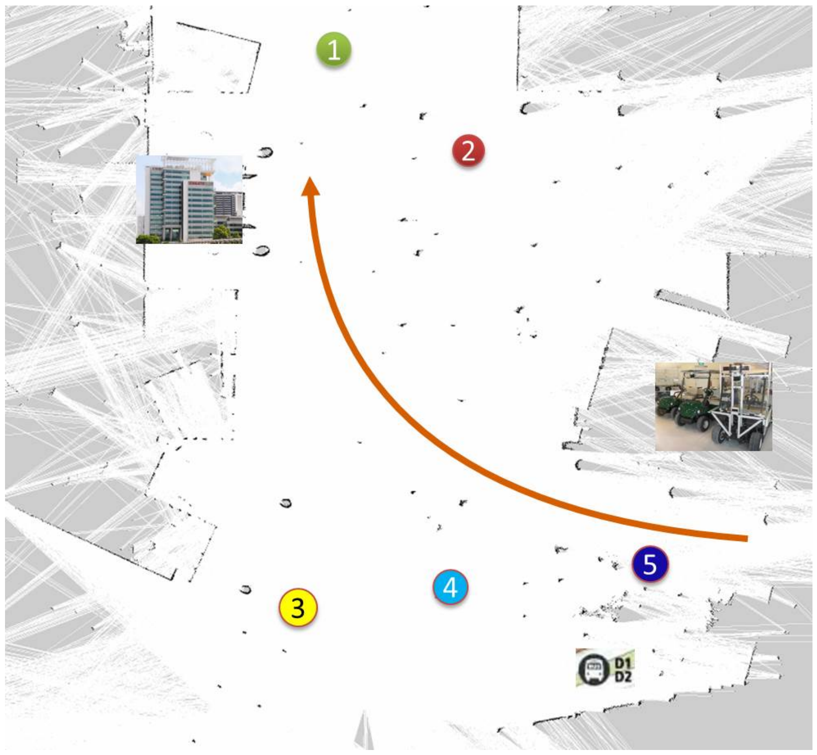


Figure 4.6: UTown Plaza environment

4.3.2 Results

Table 4.2: Result for straight lane

	N=3		N=7	
	Time	Accident	Time	Accident
DESPOT	17.84(0.08)	0.0074	21.33(0.13)	0.030
POMCP	17.15(0.07)	0.019	22.34(0.12)	0.051
MOMDP	16.53(0.03)	0.036	20.73(0.14)	0.063
QMDP	16.99(0.077)	0.023	21.75(0.13)	0.055
Reactive	16.41(0.06)	0.030	19.87(0.09)	0.0941
Random Action	19.02(0.10)	0.046	29.03(0.21)	0.087

Table 4.3: Result for roundabout

	N=7		N=15	
	Time	Accident	Time	Accident
DESPOT	22.67(0.10)	0.040	28.42(0.15)	0.098
POMCP	21.96(0.10)	0.042	28.14(0.18)	0.105
MOMDP	22.12(0.13)	0.077	28.12(0.23)	0.146
QMDP	22.88(0.14)	0.069	28.93(0.22)	0.139
Reactive	22.02(0.09)	0.061	26.56(0.11)	0.133
Random Action	26.86(0.17)	0.071	42.03(0.34)	0.167

All the experiment results are averaged over 4000 simulation runs. Each experiment is running with two different number of pedestrians N. As in Table 4.2, Table 4.3 and Table 4.4, Columns 2 and 4 report the average time and standard error for the vehicle to reach the final goal, and columns 3 and 5 report the accident rate which happens when pedestrian and vehicle are close within certain distance. For comparison, we tune the reward function of the first four algorithms so that the difference in their navigation time is less than 1 second.

Firstly, let us look at the comparison between DESPOT and offline algorithms. As the result shows in Table 4.2, Table 4.3, and Table 4.4, DESPOT produces consistently lower accident rate than both MOMDP and QMDP at all levels of pedestrian numbers.

Table 4.4: Result for UTown Plaza

	N=20		N=50	
	Time	Accident	Time	Accident
DESPOT	23.19(0.09)	0.021	29.52(0.16)	0.078
POMCP	22.45(0.08)	0.025	29.28(0.14)	0.082
MOMDP	22.9(0.1)	0.034	29.44(0.24)	0.112
QMDP	22.9(0.1)	0.033	29.55(0.22)	0.108
Reactive	21.54(0.06)	0.055	25.34(0.09)	0.145
Random Action	24.82(0.14)	0.059	37.62(0.33)	0.186

This is due to the inherent limitation of offline planning, as explained in section 4.2. Since offline POMDP planners cannot dynamically adapt to the environmental change and have to make approximations towards multiple pedestrians, the gap between the model used by the offline planner and the real environment model is always large than online planners, therefore lead to the worse performance.

Now let us move to the comparison between two online planners, DESPOT and POMCP. In the Roundabout and UTown Plaza problem, the accident rate of DESPOT is only slightly lower than that of POMCP (Table 4.3 and Table 4.4), while in the Straight lane problem, DESPOT result shows a significantly lower accident rate than POMCP (Table 4.2). The performance discrepancy here could be explained by POMCP's worst case exponential convergence time. Since our problem has a large branching factor, it may be hard for POMCP to converge under the planning time constraint. Also the standard implementation of POMCP is using unweighted particles, which means the particle set can easily go empty during belief update as it simply rejects all the particles inconsistent with the current observation (see more details in section 4.3.1).

Compared with the first four POMDP algorithms, reactive algorithm has a shorter running time but a higher accident rate in most cases as shown in Table 4.2, Table 4.3 and Table 4.4. This is because reactive planner does not decelerate until it gets very close to the pedestrian.

The collision rates in the simulation experiments are much higher than that in reality. In our experiments on the robot golf cart (Chapter 5), we have never encountered a single collision in all our experiments. The possible reasons for a high collision rate in simulation could be explained by the following. Firstly, since we adopt a discrete model similar to the previous offline POMDP work for the purpose of comparison, the discretization may cause error to collision detection. In addition, we check collision by assuming that the vehicle is disk-shaped, which could further increase the error rate. Secondly, currently we do not take into consideration pedestrians interaction with the vehicle, thus some collision may be caused by unrealistic scenarios such as pedestrians crashing into the vehicle. Lastly, since the original purpose of the simulation is to compare performances of vehicle navigation between DESPOT and other algorithms, we generate simulation results in a higher collision level for the sake of computational source and time and focus less on controlling the absolute collision rate value.

To make a more realistic simulation, we can improve the model from the following aspects. Firstly, we can replace the original model with a continuous one (which is already being used in recent work) that has more realistic collision checking criteria and models the shape of the vehicle's geometry more accurately. Secondly, we can model the pedestrian's interaction with the vehicle, *e.g.*, Social Force Model [Helbing and Molnar, 1995] takes into account the deviations in pedestrian's path by calculating the impact from the vehicle and surrounding obstacles.

Chapter 5

Experiments on the Autonomous Vehicle

5.1 System Overview

5.1.1 Hardware

Our test vehicle is a modified YAMAHA G22E golf cart (Figure 5.1) mounted with various sensors and actuators to achieve autonomy. The brake pedal and steering wheel are controlled by two motors, and the throttle is controlled directly by DC voltage. The motors for braking and steering behave as stepper motors receiving appropriate number of pulses to reach the desired position. The throttle receives the PWM signal from the controller in its voltage range. Computers control those actuators through BeagleBone controller.

The sensors most relevant to our experiment are the LIDAR, wheel encoders and Inertia Measurement Unit (IMU). The LIDAR we use is SICK LMS291 laser range finder, which has a range of 30 meters (max range of 80 meters) with an error of about 10mm and a field-of-view (FoV) of 180°. The wheel encoders and IMU are used to

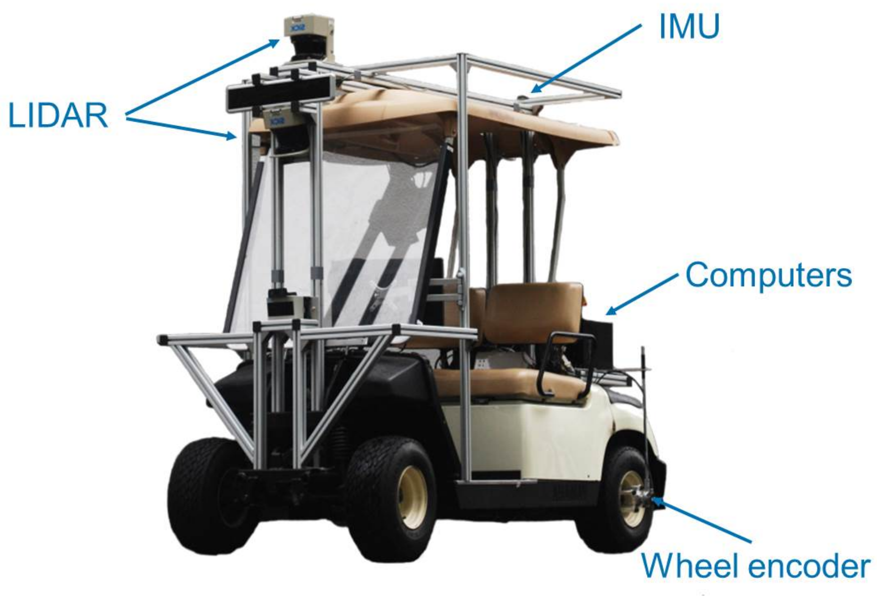


Figure 5.1: YAMAHA G22E golf cart

estimate the state of the vehicle. We integrate the input from wheel encoder tick counts and IMU gyro information with Kalman Filter to give a relatively accurate dead-reckoning estimate of the vehicle's pose and speed.

5.1.2 Software

Our code is written based on the Robot Operating System (ROS) [Quigley *et al.*, 2009], which is a popular platform providing a rich variety of frequently used navigation software packages. The framework of our software is shown in Figure 5.2, with various software modules (shown in rectangles) working together.

Vehicle State Estimation. The vehicle state estimation module contains two components which are localization and speed estimation. The localization module uses Adaptive Monte-Carlo Localization Algorithm (AMCL) [Thrun *et al.*, 2005], taking as input the sensor data from LIDAR and IMU. Same as general particle filtering algorithm, AMCL maintains a set of particles to represent the probability

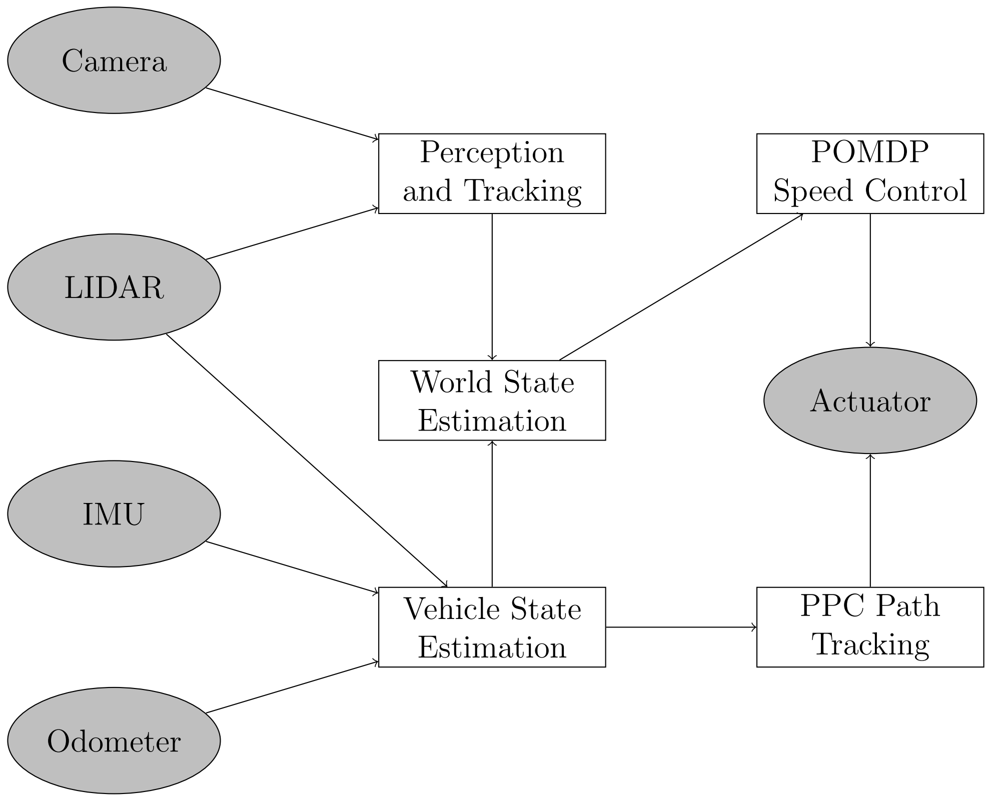


Figure 5.2: System architecture

distribution of the pedestrians. At each time step, the algorithm first samples new particles from the old particles combining dead-reckoning pose estimation. Then it does importance resampling based on the perception received from LIDAR. Specifically, it adopts a likelihood field perception model [Thrun *et al.*, 2005], which firstly maps the scanned points into the global coordinate system, then calculates the joint likelihood of each point in the scan. The likelihood of a point is calculated from a zero-centered Gaussian, taking the distance from the point to the nearest obstacle as the variable.

PPC Path Tracking. The Pure-Pursuit Controller (PPC) [Kuwata *et al.*, 2009] module handles path tracking task based on the output of the vehicle state estimation module. The path tracking module reads a sequence of way points from start to end at the beginning of navigation. During runtime, path tracking algorithm computes the steering command heading towards the next way point based on the estimation of the vehicle's position and speed.

Perception and Tracking. Perception and tracking module are the key components of our system, as they provide the information of the pedestrians' positions needed by the POMDP planner. We use LIDAR as the main source of perception input. LIDAR returns an array of detected points, and we cluster the points into groups according to their spatial proximity. After that, we filter out candidate clusters for pedestrians based on their sizes and velocities [Chong *et al.*, 2011]. In the current implementation, we use only a simple linear velocity model for pedestrians which may have difficulty distinguishing pedestrians from other similar static and moving obstacles. To correct those false positive errors, we pre-specify all the static obstacles in the map and program LIDAR to ignore these static obstacles during detection. Furthermore, in our experiment venue, most moving obstacles are pedestrians so that detection errors are limited, and even some other moving obstacles mistakenly detected can still be avoided as generic dynamic obstacles. Besides LIDAR, there is also an alternative source of input called on-board camera that is less suitable to our model but worth mentioning, which returns a sequence of images and uses a HoG classifier [Dalal and Triggs, 2005] to label the pedestrians in each image. On-board cameras are good at classifying different kinds of obstacles with image processing approaches, but it has a limited field-of-view and lacks robustness in pedestrian labeling. As a result of the innate features of the two perception sources, on-board camera is prone to generate dangerous false negatives in pedestrian detection, while LIDAR could only cause limited false positives which can still guarantee safety. Therefore, LIDAR is considered more suitable for our experiment.

Once LIDAR detects the pedestrians, our tracking module associates each pedestrian with a unique ID. However, we observed some false associations when facing dense pedestrian crowd. For instance, if two pedestrians stand too close, the tracker will recognize them as one object. This is actually acceptable as nearby pedestrians are considered to have similar future behaviors. Another more notable issue is that if two close pedestrians switch positions, their IDs may also be swapped, resulting in wrong belief tracking. However, after about three to five time steps, belief update module can automatically correct the error based on their new observations.

POMDP State Estimation. After we get the vehicle's state information from the vehicle state estimation module and the pedestrians' information from perception module, a world estimation module comes in to combine them into POMDP state and observation ready for the POMDP solver to use. Basically there are two main tasks of this module: the first task is to filter out errors such as duplicated detection and false detection caused by an imperfect perception module; the second task is to maintain the local planning window and transform all the global coordinates to local coordinates.

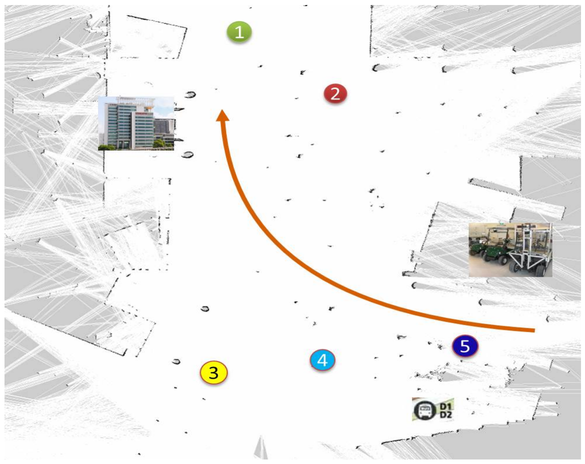
POMDP Speed Control. POMDP speed control module has two parts, namely a POMDP solver and a speed publisher. POMDP solver uses a DESPOT solver initialized with random belief of pedestrians' intentions. The solver then updates the belief when a new observation is passed in from the POMDP state estimator and issues a control command out of *accelerate*, *decelerate* and *maintain*. Then we run a ROS speed publisher routine in a separate process that sends speed to the low-level controller at higher frequency. The speed publisher smoothly increases (decreases) to the desired speed according to a user-specified acceleration.

5.2 Experiment Results

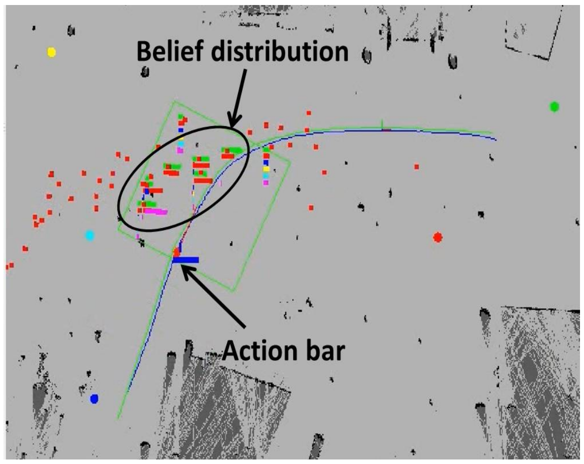
5.2.1 Environment Description

We choose UTown Plaza as our experiment venue. It is a popular spot on campus, and large crowd of people pass by throughout the day, especially upon the arrival of school buses. In Figure 5.3, we show a brief description of the environment. Figure 5.3(a) is the a-priori laser-scan map constructed before running the vehicle. Those parts with dark color represent obstacles or unknown areas, while the light color represent free area. Our path starts from the golf cart garage to the CREATE tower, as denoted by the orange curve in Figure 5.3(a). There are no obvious lanes for the vehicle and pedestrians to follow, and the space is filled with different kinds of obstacles.

In the previous chapter, we described our intention representation as geometry coordinates of subgoals. However, intention could be a more general concept. For example, we can also define the intentions to be the pedestrians' walking directions, or their decisions to whether stop or move. Based on our observation during the experiment, sometimes pedestrians just stay stationary talking or making phone calls rather than moving somewhere. Therefore, we add an additional stop intention to model this kind of scenario, and our definition of intention includes five subgoals along the path which are the intersections of pedestrian flow, plus an additional stop

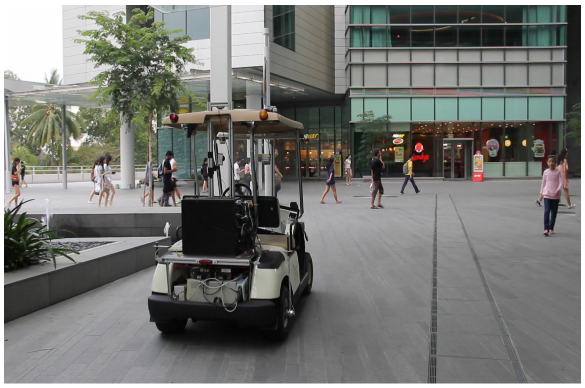


(a)



(b)

(c)



(d)

Figure 5.3: (a) is the static map of the UTown Plaza environment. (b) is a runtime map of the environment with dynamically detected obstacles. (c) and (d) are the photos taken on spot.

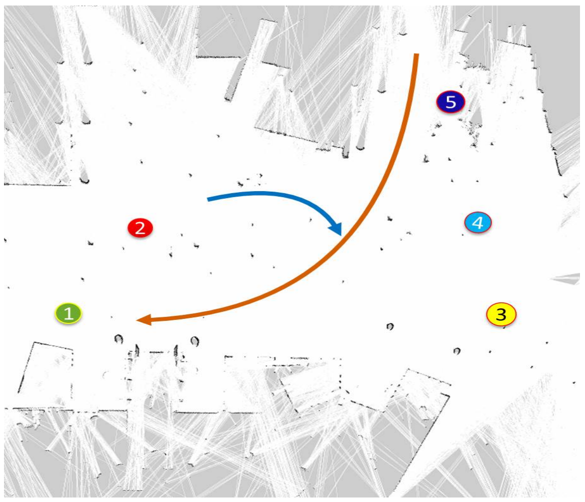
intention.

Figure 5.3(b) is a runtime map with dynamically detected obstacles on top. Small red squares represent the pedestrians and the green rectangle represents the current planning window. Inside the window there are some color bars representing the current probability distribution of each pedestrian's intention. Each bar corresponds to a subgoal marked with the same color, and the length of the bar indicates the probability of the pedestrian heading for that subgoal. During this specific run, pedestrians are mostly going to subgoal 1 and 2 so that there are many red and green bars on the map. Additionally, there's a similar looking bar at the bottom-center of the planning window indicating the current action of the vehicle. Green indicates ACCELERATE, red indicates DECELERATE and blue indicates MAINTAIN.

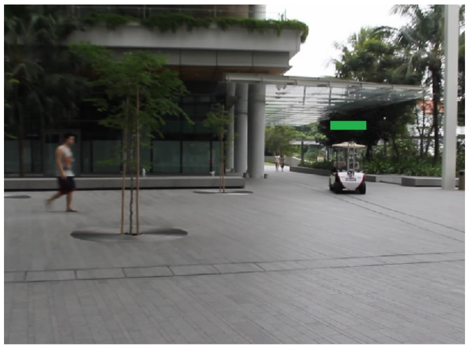
5.2.2 Case Analysis

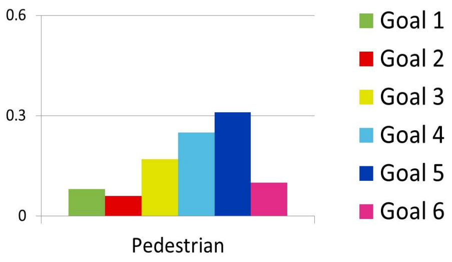
In the previous chapter, we described the advantage of our new approach and compared with other methods quantitatively. In this section, we will show experimental results on the real vehicle for a better understanding of our approach, and compare with the performance of a reactive dynamic virtual bumper method. In the following, goal 1 - 5 correspond to the five subgoal locations marked in Figure 5.4a, and goal 6 is the stop intention. The group of colored bars in Figure 5.4 and Figure 5.5 represent the probability distribution of the pedestrian walking towards each subgoal. And the colored bars above the vehicle in Figure 5.4 represent the action chosen at that time step, where green represents acceleration, red represents deceleration, and blue represents cruising.

We first show the vehicle dealing with a single pedestrian and see how the policy and belief changes at different time steps. In Figure 5.4a, the orange arrow represents the path of the vehicle, and the blue arrow represents the path of the pedestrian. From t=0s to t=5s, a pedestrian walks from left to right (Figure 5.4b, Figure 5.4c) so



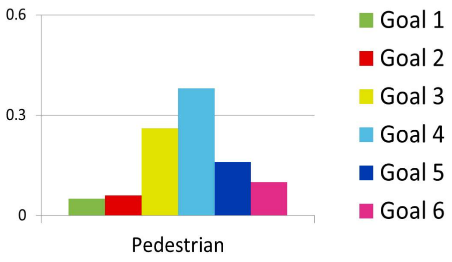
(a) top-down view of the goals and the paths of the vehicle and pedestrian





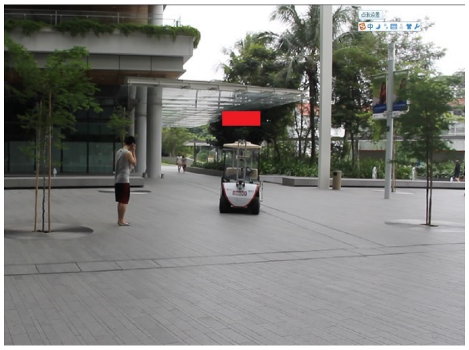
(b1) (b2)

(b) t=0s, vehicle starts to move, action=ACCELERATE

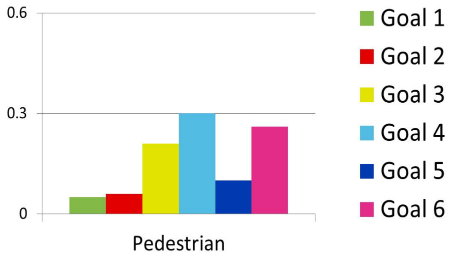


(c1) (c2)

(c) t=3s, vehicle approaches, action =MAINTAIN

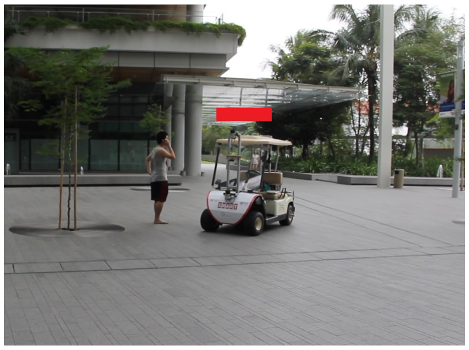


(d1)

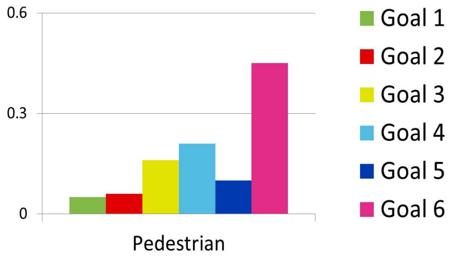


(d2)

(d) t=6s, vehicle slows down, action=DECELERATE

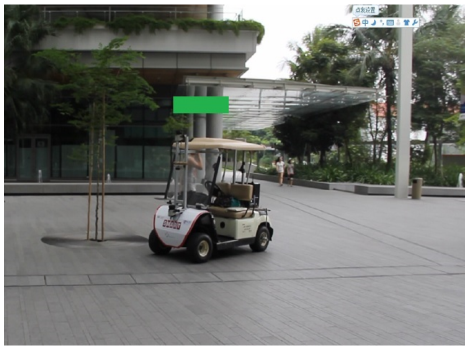


(e1)

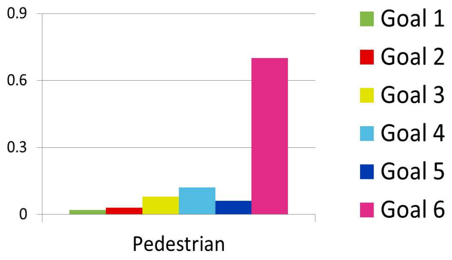


(e2)

(e) t=9s, vehicle stops, action=DECELERATE



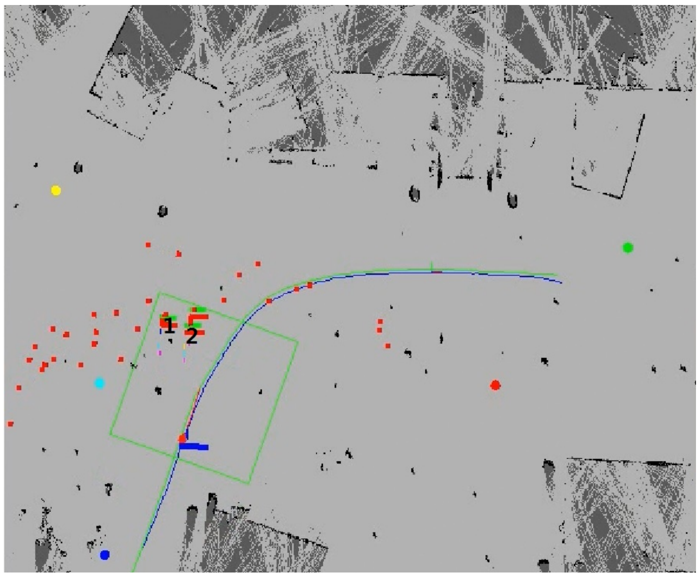
(f1)



(f2)

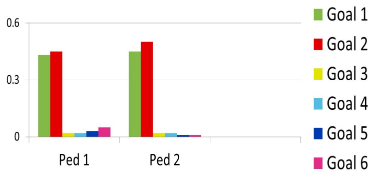
(f) t=12s, vehicle overtakes, action=ACCELERATE

Figure 5.4: Demonstration of the vehicle's behavior when a pedestrian stops besides its path.



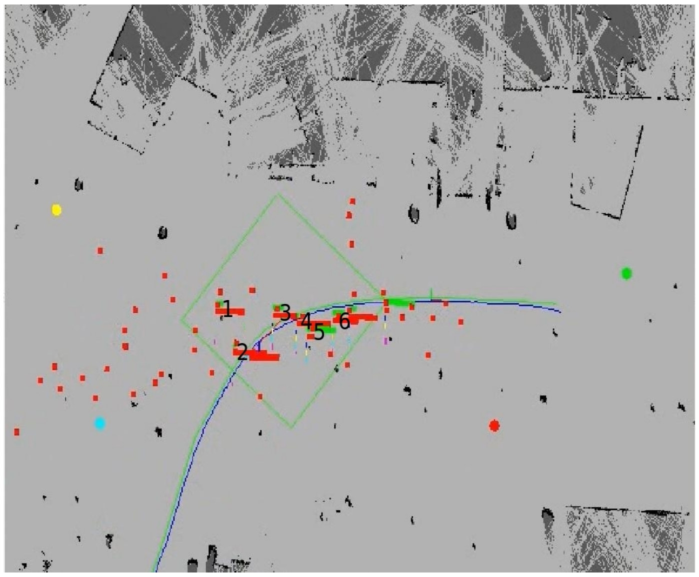
(a1)

(a2)

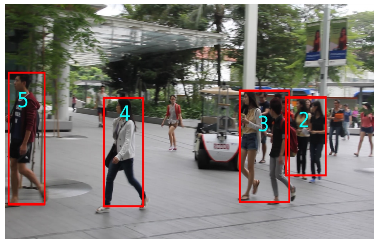


(a3)

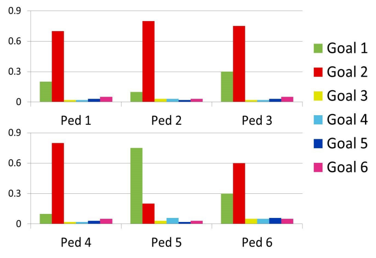
(a) t=3s, vehicle maintains a high speed 2m/s when no pedestrians around



(b1)

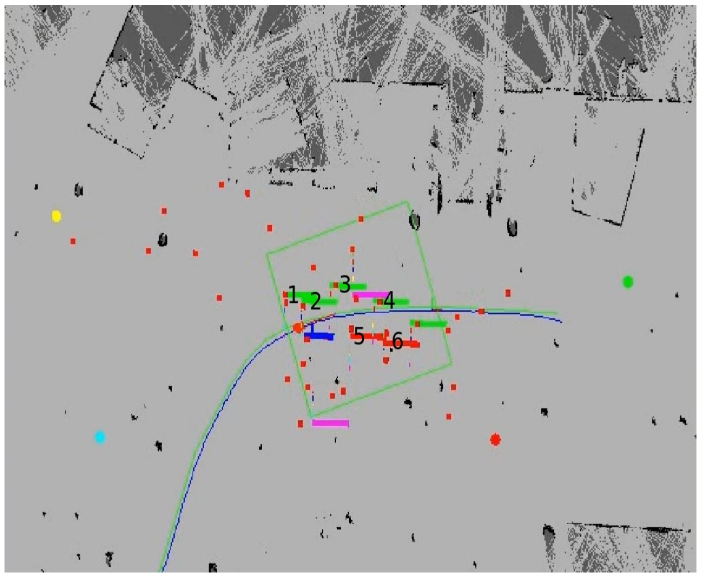


(b2)

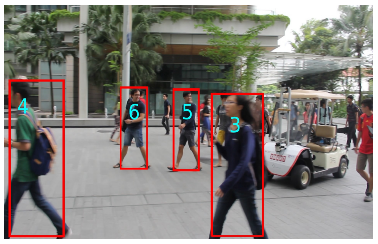


(b3)

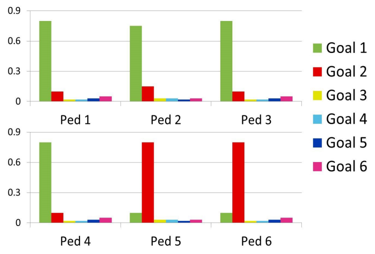
(b) t=10s, vehicle brakes to avoid collision



(c1)



(c2)



(c3)

(c) t=15s, vehicle maintains a low speed 1m/s

Figure 5.5: Interacting with dense pedestrian crowd

that the intention distribution bias towards the goals on the right side, as in Figure 5.4(b2) and Figure 5.4(c2), the blue and yellow bars are longer. The planner then chooses DECELERATE action at t=6s to slow down the vehicle. However, after the pedestrian stops moving due to a sudden phone call, his intention of stopping starts to grow (Figure 5.4d). After about 2 seconds, his intention distribution converges to stop intention (goal 6) and therefore the vehicle confidently overtakes him(Figure 5.4e).

The second case is the vehicle interacting with many pedestrians during peak hour. Figure 5.5a, Figure 5.5b and Figure 5.5c show three different time steps during a single navigation process. In Figure 5.5a, the vehicle maintains a high speed of 2.0m/s when no nearby pedestrians. In Figure 5.5(b1), the vehicle's current speed is 2.0m/s and there are pedestrian with id 2, 3 and 4 on the left-hand side. In Figure 5.5(b3), we can see the intention distributions of the three pedestrians converge to subgoal 2 which indicates they tend to cross the path in the near future, so that the vehicle takes an DECELERATE action (red action bar). We can also see from the camera view (Figure 5.5(b2)) that pedestrians marked with 2, 3 and 4 (close pedestrians are grouped as one detection) are on the way of crossing so that deceleration is appropriate. In Figure 5.5c, after deceleration, the vehicle's speed decreased to 0.8m/s. Although there are still some pedestrians nearby, the vehicle chooses a MAINTAIN action to slowly cruise along with the crowd. The reason for choosing this policy can be find in Figure 5.5(c1) and Figure 5.5(c3), where the pedestrians' intention distributions on the left-hand side are almost converged to subgoal 1, while the intention distributions on the right-hand side are mostly converged to subgoal 2. Therefore, the planner infers that among the nearby pedestrians, it is unlikely that someone is going to cross the path and it's better to cruise behind with a low speed rather than stop completely. The corresponding camera view is shown in Figure 5.5(c2), in which we can see pedestrian 3 and 4 are on the left-hand side, and pedestrian 5 and 6 are on the right-hand side.

In contrast, a reactive controller may not handle well the above scenarios, as it does not take pedestrians' intentions into account and only plan for the current observable state. The reactive controller decides to stop or go based on the distance to each pedestrian. If there is a pedestrian with a distance smaller than k meters to the vehicle, the controller will execute a brake action. The navigation is safe but jerky with a large k value. For example, in the scenario of Figure 5.4, the vehicle might just stop behind the pedestrian until he moves away. In the scenario of Figure 5.5b, the vehicle could also stop although pedestrians have low probability to cross. On the other hand, a small k value could lead to dangerous situations. In the scenario of Figure 5.4, the vehicle might just pass through without stopping, which is too aggressive since the pedestrian actually intends to cross at first.

The video including the above two scenarios and more runs is available in the attached CD-ROM and also on Youtube (http://youtu.be/UHKULAtzaFk).

5.2.3 Implementation Issues

In the experiment in section 4.3, our performance measurement only includes the accident rate. However, in the real world scenario, smoothness is another important concern besides safety. If the vehicle generates sudden acceleration and braking frequently, the passenger might feel extremely dizzy, as well as the pedestrians passing by are scared. To avoid sudden and frequent acceleration and deceleration, one way is to bias the action towards MAINTAIN by adding a proper penalty term to ACCELERATE and DECELERATE (see section 3.3.4). It is worth noting that if the penalty term is larger than the discounted goal incentive, the vehicle will not move at all. The parameters we use is shown in Table 4.1, the ACCELERATE and DECELERATE get -10 reward while MAINTAIN gets -1 reward. Based on our observation, this reward model makes the vehicle tend to maintain its speed rather than accelerate or decelerate frequently.

We have also tried different sets of noise parameters on the vehicle. In one set of parameters, we assume pedestrian's behavior is noisy in the sense that the probability to the most probable direction is about 10 times than that of the least probable direction. In this case the vehicle's behavior is conservative and tends to keep large distances from the surrounding pedestrians, hedging against the probability that the pedestrian walks to its front. In the other set of parameters, we set the noise parameter to be very small, and the two most probable directions get more than 80% chance to be visited. In this case, the vehicle behaves more like a normal driver. However, based on our experience during the experiment, sometimes curious pedestrians attempt to play with the vehicle such as jumping to its front suddenly. In this case, the less noisy model does not perform well and may result in dangerous outcomes.

In our model description in Chapter 3, we still adopt a discrete model. This is mainly for the purpose of comparison, as some of the previous work used discrete offline POMDP. We want to control for the other parts of the model and focus our comparison on the real differences between offline and online POMDP. In our recent work, we are already trying out continuous model and test it on the real vehicle.

Previously our approach only does speed control on a single pre-defined path. In order to make our approach more flexible, later on we modified our approach to be able to accept a dynamically generated path during runtime. Basically we decompose the planning problem into two layers: the path-planning layer and the POMDP speed control layer. The POMDP speed control layer does online POMDP planning on the path generated by the path-planning layer. The path planning layer runs in a separate thread using hybrid A* search algorithm used by the Stanford "Junior" autonomous vehicle in the DARPA urban challenge [Montemerlo *et al.*, 2008]. Hybrid A* represents the vehicle's state as (x, y, θ), where (x, y) is the vehicle's location, and θ is the vehicle's heading direction. This planner can also take the vehicle's control ability such as speed and steering limitations into account.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have studied the problem of navigating in a densely populated environment. The key challenge here is interacting with dynamic obstacles, such as pedestrians, which the vehicle encounters during navigation. Pedestrian's behavior is subtle and uncertain, but mostly rational and can be predicted. Our work presents a practical online POMDP-based approach by taking pedestrians' intentions into account. Our work can be summarized from the following three aspects.

1. We design an online POMDP-based probabilistic model capturing the transition of the world especially the behavior of the pedestrians. We use the idea of local planning window to concentrate planning on the most relevant areas. It reduces the planning space from the global map to a local area. Another noteworthy point is we use the subgoal concept as the main motivation of the pedestrian's movement. In this way, we can better predict the future positions of the pedestrians and make efficient planning.

2. We carry out several experiments to make comparisons between three categories

of planners: online POMDP planners, offline POMDP planners and reactive planners. Based on the result, we find that the DESPOT online planner overcomes the limitations observed in the other two categories and is most suited for our task.

3. We successfully run the real world experiment with an autonomous golf cart running the DESPOT planner in a crowded campus area. We analyze the real-time captured data and images in two experiment runs, and compare with the reactive controller. Afterwards we describe several implementation issues for the experiment.

6.2 Future Work

1. In the model description part we present our pedestrian model which assumes pedestrian is only attracted by the subgoal. However, our current code implementation contains an alternative model which also considers impact from vehicle and other pedestrians. The reason we did not include this part in the thesis is we have not found appropriate experiment scenario to test this model. In the future we can keep working on this issue and see how it affects the navigation process.

2. Another direction for future work is to deal with large pedestrian crowd more efficiently. Right now our model considers each pedestrian as equally important, while in reality we care more about the pedestrians closer to the vehicle and expect a more accurate prediction about their behaviors. Hierarchical observation can be used to address this issue by assigning more accurate observations to the closer pedestrians and noisier observations to the far-away pedestrians. In this way, the number of observation branch can be bounded without losing

much prediction accuracy on the most important pedestrians.

3. We can also try continuous modeling in the future in order to reduce the discretization error. Furthermore, this approach would also help with simplifying the model construction process, which shall become crucial once we move to a more complicated pedestrian modeling.

Bibliography

[Alterovitz *et al.*, 2007] R. Alterovitz, T. Siméon, and K. Y. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and Systems*, pages 246–253. Citeseer, 2007.

[Asmuth and Littman, 2011] J. Asmuth and M. L. Littman. Approaching bayes-optimalilty using monte-carlo tree search. In *In Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011.

[Bandyopadhyay *et al.*, 2013] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, and D. Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

[Bennewitz *et al.*, 2005] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *The International Journal of Robotics Research*, 24(1):31–48, 2005.

[Bertozzi *et al.*, 2010] M. Bertozzi, L. Bombini, A. Broggi, M. Buzzoni, E. Cardarelli, S. Cattani, P. Cerri, S. Debattisti, R. Fedriga, M. Felisa, et al. The vislab intercontinental autonomous challenge: 13,000 km, 3 months, no driver. In *Proceedings of the 17th World Congress on ITS, Busan, South Korea*, 2010.

[Burgard *et al.*, 1999] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial intelligence*, 114(1):3–55, 1999.

[Chong and others, 2011] Z. Chong et al. Autonomous personal vehicle in crowded campus environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots & Systems, Workshop on Perception and Navigation for Autonomous Vehicles in Human Environment*, 2011.

[Chong *et al.*, 2011] Z. Chong, B. Qin, T. Bandyopadhyay, T. Wongpiromsarn, E. Rankin, M. Ang, E. Frazzoli, D. Rus, D. Hsu, and K. Low. Autonomous personal vehicle for the first-and last-mile transportation services. In *Cybernetics and Intelligent Systems (CIS), 2011 IEEE 5th International Conference on*, pages 253–260. IEEE, 2011.

[Dalal and Triggs, 2005] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[Donald *et al.*, 1993] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[Doucet, 2001] A. Doucet. *Sequential monte carlo methods*. Wiley Online Library, 2001.

[Fern and Tadepalli, 2010] A. Fern and P. Tadepalli. A computational decision theory for interactive assistants. In *Interactive Decision Theory and Game Theory*, 2010.

[Fletcher *et al.*, 2008] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, et al. The MIT–Cornell collision and why it happened. *Journal of Field Robotics*, 25(10):775–807, 2008.

[Fox *et al.*, 1997] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33, 1997.

[Gerkey and Konolige, 2008] B. P. Gerkey and K. Konolige. Planning and control in unstructured terrain. In *International Conference on Robotics and Automation Workshop on Path Planning on Costmaps*, 2008.

[Goller *et al.*, 2010] M. Goller, F. Steinhardt, T. Kerscher, J. Marius Zollner, and R. Dillmann. Proactive avoidance of moving obstacles for a service robot utilizing a behavior-based control. In *Intelligent Robots and Systems, 2010 IEEE/RSJ International Conference on*, pages 5984–5989. IEEE, 2010.

[Guizzo, 2011] E. Guizzo. How googles self-driving car works. *IEEE Spectrum Online, October*, 18, 2011.

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

[Iagnemma and Buehler, 2006] K. Iagnemma and M. Buehler. Editorial for Journal of Field Roboticsspecial issue on the DARPA grand challenge. *Journal of Field Robotics*, 23(9):655–656, 2006.

[Ikeda *et al.*, 2012] T. Ikeda, Y. Chigodo, D. Rea, F. Zanlungo, M. Shiomi, and T. Kanda. Modeling and prediction of pedestrian behavior based on the sub-goal concept. In *Robotics: Science and Systems*, 2012.

[Kaelbling *et al.*, 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[Kavraki *et al.*, 1996] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European Machine Learning and Data Mining Conference*, pages 282–293. Springer, 2006.

[Kurniawati *et al.*, 2008] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient Point-Based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, pages 65–72, 2008.

[Kuwata *et al.*, 2009] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore. Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on*, 17(5):1105–1118, 2009.

[Latombe, 1996] J.-C. Latombe. *Robot Motion Planning*. Springer, 1996.

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

[Leonard and Durrant-Whyte, 1991] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[Littman *et al.*, 1995] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, volume 95, pages 362–370. Citeseer, 1995.

[Lozano-Perez, 1983] T. Lozano-Perez. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, 100(2):108–120, 1983.

[Maisto, 2014] M. Maisto. Induct now selling Navia, first self-driving commercial vehicle. *eWeek*, 2014.

[Markoff, 2010] J. Markoff. Google cars drive themselves, in traffic. *The New York Times*, 10:A1, 2010.

[McAllester and Singh, 1999] D. A. McAllester and S. Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416. Morgan Kaufmann Publishers Inc., 1999.

[Montemerlo *et al.*, 2008] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[Ong *et al.*, 2010] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[Paquet *et al.*, 2005] S. Paquet, L. Tobin, and B. Chaib-Draa. An online POMDP algorithm for complex multiagent environments. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 970–977. ACM, 2005.

[Pineau *et al.*, 2003] J. Pineau, G. Gordon, S. Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 3, pages 1025–1032, 2003.

[Prentice and Roy, 2011] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Robotics Research*, pages 293–305. Springer, 2011.

[Quigley *et al.*, 2009] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *International Conference on Robotics and Automation Workshop on Open Source Software*, volume 3, 2009.

[Ross *et al.*, 2007] S. Ross, B. Chaib-Draa, et al. AEMS: an anytime online search algorithm for approximate policy refinement in large POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 2592–2598, 2007.

[Ross *et al.*, 2008] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[Schiller *et al.*, 1998] B. Schiller, V. Morellas, and M. Donath. Collision avoidance for highway vehicles using the virtual bumper controller. In *Proceedings of the IEEE International Symposium on Intelligent Vehicles*, 1998.

[Shani *et al.*, 2007] G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 2619–2624, 2007.

[Siegwart *et al.*, 2003] R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, et al. Robox at Expo. 02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3):203–222, 2003.

[Silver and Veness, 2010] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 23, pages 2164–2172, 2010.

[Smith and Simmons, 2004] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 520–527. AUAI Press, 2004.

[Somani *et al.*, 2013] A. Somani, N. Ye, D. Hsu, and W. S. Lee. DESPOT: online POMDP planning with regularization. In *Advances in Neural Information Processing Systems*, pages 1772–1780, 2013.

[Spaan and Vlassis, 2005] M. T. Spaan and N. A. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[Thrun *et al.*, 2005] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[Van Den Berg *et al.*, 2011] J. Van Den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.