Autonomous Driving at Intersections: A Critical-Turning-Point Approach for Planning and Decision Making

by

Keqi Shu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2020

Author's Declaration

Abstract

Left-turning at unsignalized intersection is one of the most challenging tasks for urban automated driving, due to the various shapes of different intersections, and rapidly changing nature of the driving scenarios. Many algorithms including rule-based approach, graph-based approach, optimization-based approach, etc. have been developed to overcome the problems. However, most algorithms implemented were difficult to guarantee the safety at intersection scenarios in real time due to the large uncertainty of the intersections. Other algorithms that aim to always keep a safe distance in all cases often become overly conservative, which might also be dangerous and inefficient.

This thesis addresses this challenge by proposing a generalized critical turning point (CTP) based hierarchical decision making and planning method, which enables safe and efficient planning and decision making of autonomous vehicles. The high-level candidate-paths planner takes the road map information and generates CTPs using a parameterized CTP extraction model which is proposed and verified by naturalistic driving data. CTP is a novel concept and the corresponding CTP model is used to generate behavior-oriented paths that adapt to various intersections. These modifications help to assure the high searching efficiency of the planning process, and in the meanwhile, enable human-like driving behavior of the autonomous vehicle. The low-level planner formulates the decision-making task to a POMDP problem which considers the uncertainties of the agent in the intersections. The POMDP problem is then solved with a Monte Carlo tree search (MCTS)-based framework to select proper candidate paths and decide the actions on that path.

The proposed framework that uses CTPs is tested in several critical scenarios and has out-performed the methods of not using CTPs. The framework has shown the ability to adapt to various shapes of intersections with different numbers of road lanes and different width of median strips, and finishes the left turns while keeping proper safety distances. The uses of the CTP concept which is proposed through human-driving left-turning behaviors, enables the framework to perform human-like behaviors that is easier to be speculated by the other agents of the intersection, which improves the safety of the ego vehicle too. The framework is also capable of personalized modification of the desired real-time performance and the corresponding stability. The use of the POMDP model which considers the unknown intentions of the surrounding vehicles has also enabled the framework to provide commute-efficient two-dimensional planning and decision-making. In all, the proposed framework enables the ego vehicle to perform less conservative and human-like actions while considering the potential of crashes in real-time, which not only improves the commute-efficiency, but also enables urban driving autonomous vehicles to naturally integrate into scenarios with human-driven vehicles in a friendly manner

Acknowledgements

I would like to thank my supervisor Prof. Dongpu Cao for giving me the opportunity to study and doing research on my field of interest in the Cogdrive lab. His patience and understanding and the kindly encouragements helped me navigate the toughest challenges in this journey. I am very lucky to have such a mentor both in research and life.

I would like to thank Dr. Huilong Yu, who had provided profound insights, ideas which made the project fun and exciting. The knowledge you passed on through the heated discussion is embedded deep in my heart.

I would also like to thank all my lab mates, without whom my life in Waterloo would be a lot different, the fun we had and the pleasure of research we enjoyed together would be my beautiful memories as a graduate student.

Thanks to my figure skating varsity teammates and coach Mrs. Kim Biddiscombe, who made my life a lot more exciting and fun after school.

Finally, thanks to my family for always supporting me and encouraging me throughout the toughest times, this thesis would not have been possible without them.

Dedication

This is dedicated to my family.

Table of Contents

List of Figures ix

List of Tables xi

List of Abbreviations xii

1 Introduction 1

1.1 Background 1

1.2 Remaining Challenges 2

1.3 Thesis Objectives 4

1.4 Outline 4

2 Literature Review 6

2.1 Introduction 6

2.2 Rule Based Approach 6

2.3 Graph Based Approach 7

2.4 Optimization Based Approach 9

2.5 Reinforcement Learning Based Approach 11

2.6 POMDP Based Approach 11

2.6.1 Exact Solution 13

2.6.2 Offline Approximate Solution 15

2.6.3 Online Approximate Solution 19

2.7 Summary 24

3 Left-turning Human Driving Behavior Analysis 25

3.1 Introduction . . . 25

3.2 Left-Turning Behavior Analysis . . . 25

3.3 Critical Turning Point Model . . . 29

3.4 CTP Model Identification and Validation . . . 31

3.5 Summary . . . 37

4 CTP-Based Left-Turning Planning and Decision-Making Framework 38

4.1 Introduction . . . 38

4.2 A Hierarchical Planning and Decision Making Framework . . . 38

4.2.1 High-level Behavior-Oriented Paths Generation . . . 40

4.2.2 Candidate Path Selection and Speed planning . . . 42

4.3 CTP-based Candidate Paths Generation . . . 42

4.4 POMDP Problem Formulation . . . 46

4.4.1 State Space . . . 47

4.4.2 Observation Space and Observation Model . . . 48

4.4.3 Action Space . . . 48

4.4.4 Reward . . . 49

4.4.5 Transition Model . . . 50

4.5 Solving the Decision-Making Problem . . . 51

4.5.1 Representation of the Policy . . . 51

4.5.2 Policy Generation and Update . . . 54

4.6 Summary . . . 56

5 Simulation and Results 58

5.1 Introduction . . . 58

5.2 Simulation under Typical Scenarios . . . 58

5.3 Adaption to Various Road Geometries . . . 61

5.4 Optimization between Tree Size and Performance Stability . . . 63

5.5 Summary . . . 65

6 Conclusions and Future Work **66**

6.1 Conclusions 66

6.2 Future Work 67

References **68**

List of Figures

2.1 Value function representation by hyperplanes in a continuous belif state [1] 13
2.2 Procedure of value iteration [2] 14
2.3 Tree constructed by MCTS 23
2.4 Outline of Monte Carlo Tree Search [3] 24

3.1 Real-life driving behavior of left turns 26
3.2 Selected scenarios in the INTERACTION dataset [4] 27
3.3 Extracted left-turning trajectories from naturalistic dataset 28
3.4 Yaw rate and speed along the global coordinates and left-turning paths . . 29
3.5 Critical turning point model 30
3.6 CTP location distribution in the local frame 32
3.7 CTPs extracted from regular intersection dataset 34
3.8 Driving style evaluation 35
3.9 CTP extracted from irregular intersection dataset 36

4.1 CTP-Based left-turning planning and decision-making framework overview 39
4.2 Simplified nonholonomic vehicle model for path generation [5] 43
4.3 Interpolating of the 2-D curve [5] 44
4.4 Illustration of the relation between the episode and the path in the belief tree [6] 53

5.1 Simulation results of three typical scenarios with different oncoming vehicle's intention 59

5.2 Vehicle distance and speed profile with respect to time in two critical scenarios 60
5.3 Change of actions due to different beliefs 61
5.4 Speed profile and marching ratio with respect to time 62
5.5 Commuting efficiency with different number of CTPs 64

List of Tables

4.1 Recommended parameters for path generation of four typical intersections 46

5.1 Performance evaluated in different intersections 63

5.2 Different tree sizes with according performances 65

List of Abbreviations

ABT Adaptive Belief Tree 51, 54, 55

ADAS Automotive Driver Assistance Systems 1

AV Autonomous Vehicles 1, 6, 11, 12

CL-RRT Closed-loop Rapidly-exploring random tree 7

CTP crtical turning point 25, 30, 31, 33, 35, 37, 38, 40, 42, 48, 50, 56, 59, 61–67

DARPA Defense Advanced Research Projects Agency 1

MCTS Monte Carlo tree search 11, 22, 23, 42, 51, 66

MDP Markov decision process 11, 16, 21

MPC Model predictive control 9, 10

PBVI Point-based value iteration 17–19

POMDP partially observable Markov decision process 4, 6, 11–17, 19, 22, 24, 42, 46, 51, 54–57, 60, 63, 65, 66

RL Reinforcement learning 11

RRT Rapidly-exploring random tree 7, 8

SAE Society of Automotive Engineers 1

UCB upper confidence bound 54

UCB1 upper confidence bound, version 1 54, 56

Chapter 1

Introduction

1.1 Background

The development of the Automotive Driver Assistance Systems (ADAS) has grown rapidly due to its increasing potential to improve the efficiency of traffic, and meanwhile, ensure the safety of the passengers and drivers [7]. After the Defense Advanced Research Projects Agency (DARPA) Grand Challenge [8] where many benchmark works have been established and implemented, the ADAS system progresses rapidly and has been implemented by many car manufacturers. Up to now, a lot of manufacturers and research labs are aiming to realized level-4 or level-5 Society of Automotive Engineers (SAE) level [9] autonomous driving.

One of the hot spots in developing a highly autonomous ADAS system is to enable safe and efficient planning and decision-making at intersections. Even for human drivers, 40% of all injury accidents that happen on road occur at intersections in the United States every year [10]. As for autonomous vehicles, most of the algorithms implemented had shown that they were difficult to guarantee safety [11] due to the complexity and dangerous nature of intersections. However, algorithms that avoid accidents in an extensive manner often become overly conservative [12], this is also very dangerous. Because overly conservative systems could lead to a counterproductive effect on the commuting efficient, and could lead to collisions, congestion and even deadlock [13].

Therefore, decision making and planning of Autonomous Vehicles (AV)s under unsignalized intersections is a challenging task [14]. Thus, it is well worth to develop planning and decision-making algorithms that significantly minimize the potential of collisions with other agents in the environments and provide efficient and comfort for both the ego vehicle and surrounding vehicles.

1.2 Remaining Challenges

Among all planning and decision-making tasks at intersections, left turning when the oncoming vehicles are not using turn signals is one of the most common and challenging tasks. The challenges remained in these research domain include 1) the exact future paths of the surrounding vehicles that are hard to predict; 2) the unknown intention of the surrounding vehicles and pedestrians; 3) the various shapes of different intersections; 4) the rapidly changing nature of the intersection scenarios; 5) the perception uncertainties of the surrounding environment.

If the exact future paths of the surrounding vehicles are certain, the ego vehicle could plan paths that increase the commute efficiency while guaranteeing safety. However, the exact long-term future trajectories of surrounding vehicles can not be 100% accurate for the following reasons. The training data need to be labeled which is very time-consuming [15]. The early predictions of surrounding vehicles are hard due to the lack of the observed traveled trajectories, the insufficient knowledge of the historic trajectories made it hard to find the specific model to fit. Long-term prediction is also changeling because the prediction error would accumulate and the predicted paths would be very unreliable [16]. most common approaches either can not always provide accurate predictions due to the consideration of the dependencies between the ego vehicle and the environment, or suffers from poor real-time performances because of computational complexity [17].

The unknown intention of the surrounding driver and pedestrian is a higher level of prediction compare to exact trajectories prediction, which is also critical to ensure the safety of the autonomous vehicles driving at intersections, especially when the real-time trajectories prediction is infeasible. However, this problem still remained to be unsolved for self-driving cars [18]. More specifically, the driving style various from people to people and a general model to fit all the driver is nearly impossible. Conservative drivers would decrease speed to very slow or even came to a full stop before making a left turn, an aggressive driver, on the other hand, might make the turns with relatively high speed or some times did not even brake before making the turns. Also, different behaviors would be performed at different road intersections and scenarios, the capabilities of adaption to different intersection scenarios is a challenging task too. For example, people driving at multi-lane intersections might be more cautious due to the increasing surrounding vehicles and the higher probability of collision compare to single lane intersections. Formulating the intention estimator with a single model would lead to errors of the estimation, which is very dangerous for the passenger on the ego vehicle. What's more, as same as future paths prediction, the longer prediction horizon is chosen, the lower confidence of the estimated intention is preserved. Finally, the balance of the weights of the labeled class for model training is also a formidable task [19].

The wide variety of different road shapes including different numbers of road lanes for regular intersections, and different road angles or T-junctions for irregular intersections would be a burden for generalized model and real-world world application of planning and decision-making at intersections. Common approaches include formulating the problem as an optimization problem and explore all the possible paths of the whole intersection, due to the number of dynamic objects and interactive nature of the intersection, this could lead to heavy computation load, which influence real-time performance. Another common approach is generating a pre-defined path and generate speed profiles on that path. However, in some irregular intersections, the pre-defined path could be challenging to generate automatically. Also, planning on a single path might lead to sub-optimal solutions.

The intersection scenarios evolve heavy interaction from the ego vehicle with the environment in a high frequency, the pedestrians and surrounding vehicles might make rapid changes in their behavior, for example, the front car might decelerate at any time to yield to the street-crossing pedestrians, which need the ego vehicle to respond in time. Moreover, there were also scenarios where occlusion occur, that the vision of left-turn vehicles is partially blocked by the left-turning vehicles in the opposite direction, especially the case where the left-turning vehicles were trucks. In this case, the so-called "ghost vehicle", which is the oncoming vehicles that were out of the $field\ of\ view$ of the ego vehicle, would drive out of the blocked area, and collide with the ego vehicle [20]. Not until the ego vehicle observed the dangerous action of the environment, could they respond in time to avoid collisions. Therefore, the frequency of decision-making needed to be realized as high as possible. However, due to the limitation of computational power, in general, good decisions usually acquire a though exploration of all the possible policies, which limit the rate of decision-making. As a result, the compromise between generating better policies and improving real-time performance becomes a challenge for the left-turning vehicle.

Finally, although many perception algorithms had been developed to obtain good preception results for autonomous vehicles [21], there still lies the problem that fatal perception error might occur due to bad perception sensor layout, occlusion at the intersections, poor sensor resolution, etc. [22], the uncertainties in perception would lead to serious collisions. As a result, it would be ideal if the planner of the ego vehicle could take the uncertainties into consideration.

1.3 Thesis Objectives

The main goal of this thesis is to achieve safe and efficient planning and decision making in urban scenarios, more specifically, the scope of this thesis focused on unsignalized left-turning scenarios. On the basis of approaches in the literature and the existing problems, the goal for the proposed framework is:

- the framework should be able to deal with complex intersection scenarios;
- the framework should enable ego vehicle to always keep a safety distance with other agents in the intersection;
- the framework should avoid overly conservative actions;
- the framework should have good generalization character and should be able to be adapted to various shapes of intersections;
- the framework should be operated online to overcome the rapidly changing nature of the intersection.

1.4 Outline

Chapter 2 first gives an overview of the popular approaches for solving planning and decision-making problems at intersections in literature, and evaluates the advantages and disadvantages of every approach. After that, a detailed review of solving partially observable Markov decision process (POMDP) problem is presented, including solving an exact solution of POMDP problem and approximate solutions.

Chapter 3 analyses the left-turning driving behavior through naturalistic driving datasets and from the observed phenomena, a CTP model is proposed. The generalized and parametric model is then verified through further evaluation of the naturalistic driving datasets.

Chapter 4 proposed a CTP-based left-turning hierarchical planning and decision-making framework. First, the structure of the framework is presented, then the high-level candidate path generation method is presented. Finally, the low-level path selection and speed profile generation method is proposed, which includes both POMDP problem formulation and POMDP problem-solving method.

Chapter 5 describes the simulation results of the proposed framework. The framework is first tested in several typical scenarios in simulation and the commute efficiency is evaluated. Then, the ability of adaption to different shapes of road intersections are also tested. Finally, the optimization between a more optimal solution and performance stability is discussed.

Chapter 6 summarized this thesis, highlights the major contributions, and discussed about possible future works that would be done.

Chapter 2

Literature Review

2.1 Introduction

Decision making and planning of AVs under unsignalized intersections is a challenging task [14] due to the rapidly changing nature and unknown intentions of the surrounding human-driven vehicles. Left turning when the oncoming vehicles are not using turn signals is one of the most common and challenging tasks for planning at unsignalized intersections [23]. On solving such a complicated problem, balancing between safety and commuting efficiency is critical. There are various approaches that mitigate this contradiction. The approaches include rule-based, graphs based, optimization-based and machine-learning-based methods. These methods all have their advantages and drawbacks. Through comparison among the approaches, the POMDP based methods have shown the most satisfying advantages and drawbacks that could be weakened. Therefore, the major and popular approaches for solving POMDP problems are reviewed. The advantage and disadvantages of the methods are also analyzed.

2.2 Rule Based Approach

The rule-based approach is comparatively easier to implement, which is widely used for planning at intersections [24–26]. Klingelschmitt *et.al.* [25] proposed a scene representation for traffic scene modeling semantically, in which semantic state space was used to realize transition in model states. Klingelschmitt *et.al.* [24] presented a generalized and efficient situation hypotheses selection system framework, which combines a probabilistic situation recognition and a fast risk assessment algorithm. The framework was able to notably reduce the total unnecessarily considered situation hypotheses. Hubmann *et.al.* [27] proposed a longitudinal driving strategy that analyzed the Inevitable Collision States and used it as guidance to avoid collision with both static and dynamic objects. The algorithm had shown good real-time performance and the capability to be implemented on an autonomous vehicle. [28] introduced a discriminative maneuver estimator that could work in various scenarios, the approach used pairwise probability coupling to combine the reusable and partial classifiers to enable the proposed model to be able to estimate a huge amount of intersection scenarios. The result showed that the reduction of the complexity of the partial classifiers outperformed the models that are particularly designed for some scenarios. These approaches are quick to be implemented, however, the performance of the planner is closely related to the completeness of modeling every scenario, which is nearly impossible for cases as intersections.

2.3 Graph Based Approach

Another popular approach plans the trajectory and speeds only considering the current state of the environment, but plans in a high frequency at intersection scenarios using graph theory, such as the Rapidly-exploring random tree (RRT) algorithm [29]. This algorithm was first introduced by LaValle [30], after that, the algorithm had been developed to many variants and is widely used for the planning of autonomous vehicles. The tree generation process of the general RRT algorithm is presented in Algorithm 1 [30]. This algorithm uses a randomly sampled method to generate a tree-like path that explores the search space. The main procedure for generating the search tree given the initial point is: first, a point is sampled randomly in the free space, then, the distance is calculated from each of the tree nodes to the newly sampled point and the closest tree node is obtained. Then in the direction from the nearest point to the sampled point, a predefined distance extends from the nearest point to obtain a new point (n-point). Finally, if the line that connects the nearest point and n-point located inside the free space, the n-point and the connecting line is added to the search tree.

After the RRT was proposed, many variants based on this algorithm have been proposed. Few classic variations include the RRT* algorithm [31] and Closed-loop Rapidly-exploring random tree (CL-RRT) algorithm [32] which are often compared with other planning algorithms in literature as benchmarks. RRT* had shown to provide asymptotic optimal paths by adding a rewire process to the search tree growing process. CL-RRT uses the closed-loop simulation while generating the tree to provide better feasibility of the

Algorithm 1 Rapidly-exploring random tree generation

```
1: function Generate RRT ($x_{init}$, K, $\Delta t$)
2:     $\mathcal{T}$ . init($x_{init}$)
3:     for $k = 1$ to $K$ do
4:         $x_{rand} \leftarrow$ RANDOM_STATE()
5:         $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}$,$\mathcal{T}$)
6:         $u \leftarrow$ SELECT_INPUT($x_{rand}$,$x_{near}$)
7:         $x_{new} \leftarrow$ SELECT_INPUT($x_{rand}$,$x_{near}$)
8:         $\mathcal{T}$. add_vertex($x_{new}$)
9:         $\mathcal{T}$. add_edge($x_{near}$,$x_{new}$,$u$)
       return $\mathcal{T}$
```

generated paths. This method could operate in complex and constrained scenarios. With the emerging development of the RRT family, RRT-based algorithms had shown to satisfying performances and are widely applied in autonomous driving including intersection scenarios.

Ma *et.al.* [33] used a sampling-based RRT method to generate paths at a intersection. It takes advantage of the road geometry and generates splines accordingly. Using the spline as the initial tree, RRT method was used to generate obstacle avoidance trajectories. This method can quickly generate collision-free paths at high frequency. Chen *et.al.* [34] formed the problem in to a hierarchical model. The planning algorithm can be divided into three layers, route planning layer, task planning layer and motion planning layer. Route planning layer generates way-points from the road geometry. The task planner received the way-points as milestones and generate tasks accordingly. The motion planner then generate real-time control commands with each task. The prediction-planning approach can be implemented in various scenarios that benefit from the plentiful prediction and planning algorithms. Yoon *et.al.* [35] introduced a spline-based RRT* algorithm that combined cubic Bezier curves and RRT* to achieve continuous and efficient collision avoidance at intersections. This kind of method is applicable in all kinds of intersections using various prediction and planning algorithms. Mehta *et.al.* [36] brought a human interface into the RRT planner to enable to add experienced-drivers knowledge to assist the process of replanning.

However, RRT algorithms are mostly implemented in mazes and other static environments, with the complexity and fast-changing nature of intersections leads to conservative decisions and the refresh rate of the planner may not be fast enough for some critical scenarios.

Only considers and plans on the current state may be insufficient to guarantee safety. Therefore, planning with predicted future states of the surrounding vehicles has become a popular way of solving left-turn problems, benefiting from the plentiful prediction algorithms [15–19,37,38]. Huang *et.al.* [15] predicted the conditional variational distribution of future paths of the ego vehicle using multiple sensors, and realized the approach by training a variational neural network. The proposed algorithm is able to deal with inherently uncertainties and coordination between different sensors, it is also able to be enhanced by the physics-based model to increase the prediction accuracy and other performances. [37] introduced an intention prediction RNN-based method using Lidar-tracking information. The recurrent neural network was trained by a big amount of unsignalized intersection naturalistic driving dataset. The dataset was also used to evaluate the good prediction results, that the predictor was able to predict potential collision 1.3 seconds in advance. Using these prediction algorithms, the performance of the intersection planner could be improved significantly. Jeong *et.al.* [39] used SVM to predict the intention of surrounding vehicles, based on the predicted state, the ego vehicle was able to perform human-like planning behaviors at uncontrolled intersections. Jeong *et.al.* [40] trained an LSTM-based RNN to predict the motion of surrounding vehicles, with the prediction results, the planner was able to increase safety. Wang *et.al.* [41] predicted the probabilistic occupancy of surrounding vehicles with Monte Carlo simulation, the prediction model was trained off-line to achieve a good real-time performance of the online planner. If the prediction accuracy is good, the methods could provide good safety performance and could apply to different scenarios. However, to make precise predictions on all the vehicles at intersections is a formidable task.

2.4 Optimization Based Approach

Another popular approach is modeling the problem into an optimization problem, by properly modifying the reward function and constraints, planning and decision-making could be operated in a heuristic manner. One typical optimization method used for autonomous vehicles in intersections is the Model predictive control (MPC) method. The key elements in MPC include a vehicle model, an obstacle model, and a proper optimization method.

The vehicle model includes point-mass model which considers the vehicle as a point with mass, and does not consider the tire model or vehicle geometry. The kinematics model on the other hand, considers vehicle geometry which is often used in path planning. The dynamic model are developed by considering the tire model and thus are nonlinear, and are often used in control problems and high-speed scenarios with large lateral forces. The obstacle model which are used for obstacle avoidance includes two major approaches. One approach is through setting a virtual repulsive force in the cost function to add resistance from the obstacle to the vehicle, also known as the potential field methods, slack variables could be added to the constraints to obtain better acceptable feasibility. This modification leads to nonconvex and nonlinear cost functions. Another approach is through setting the constrains to allow the vehicle to operate inside the obstacle-free area, however, the constraints are most likely nonconvex. Solving nonconvex problems could cause high computation power, therefore, the model is often simulated to a linear programming problem, quadratic programming problems, etc [42]. Then the problem could be solved with the corresponding solver.

Over the years MPC has been broadly developed and used in solving planning and decision making problems for autonomous vehicles including intersection planning problems. Schildbach *et.al.* [43] designed a Robust Model Predictive Control planning algorithm to ensure safe gaps at intersection scenarios. The method maximized efficiency and driving comfort and also improved the model with "Affine Disturbance Feedback". [41] first used road geometry to generate initial trajectories, then MPC was used to refine the coarse initial trajectories and generate action and the corresponding speed profile along the smoothed path. During this process, the constraints of the optimization were easy to be implemented through coordinate transformation. Finally, probabilistic-occupancy-based predictions of surrounding vehicles were calculated offline to provide better real-time performances. The simulation results showed that the improvement of the planned path and speed, and also demonstrated the efficiency of the proposed approach.

Hult *et.al.* [44] presented a bi-level (coordination level and vehicle level) MPC for intersection planning. Which allot occupancy time slots during the intersection and generated vehicle direct control inputs separately. This approach had shown feasibility and stability with certain assumptions. Nilsson *et.al.* [45] used 2 loosely coupled MPC to make longitudinal and lateral planning accordingly. Liu *et.al.* [46] introduced a MPC model that could automatically select appropriate parameters for different maneuvers to improve the generality of the planner. To achieve better driving comfort, a lane-associated potential field was used. Also, the surrounding vehicles are modeled as polygons and are converted into constraints in MPC to ensure safety. [47] planed speed of the ego vehicle on a fixed path generated from a set of waypoints by using temporal optimization. The non-convex optimization problem was simplified by a set of quadratic programs and was solved by the slack convex feasible set (SCFS) algorithm to improve the real-time performance of the proposed method. However, this method requires accurate modeling of the vehicle and the environment that will lead to heavier computational load, which influences the real-time performance.

2.5 Reinforcement Learning Based Approach

Reinforcement learning (RL) is also a common approach for intersection decision making and planning for AVs, these approaches are commonly combined with neural networks for better generalization and feasibility. Bouton *et.al.* [48] introduced a safe RL algorithm based on the model-checker to improve safety, and a learning-based belief update method was used to allow the proposed method to coop with occlusions and perception errors. [49] enabled high-level decision making of ego vehicles at occluded multi-lane intersections. A risk-based reward function was used to combine risk assessment to the NN-based Q network, which enabled the decision-maker to perform less cautions and safe actions. [50] learned typical behaviors of other vehicles to enable the ego vehicle to coop with surrounding vehicles at intersections. The speed of the surrounding vehicles and the distance from the surrounding vehicles to the ego vehicle was taken into consideration in the model to perform different driving behaviors. The proposed approach was able to crossover the intersection and avoid collision almost all the time, which outperformed the state-of-the-art models. [51] introduced an intersection planner that considered the low-level control of the ego vehicle. The planner considered the dynamic and kinematic constraints of the ego vehicle in the deep-RL model, this improves the feasibility of the high-level planning results in real-life intersection scenarios, and the feasibility was verified by simulations. Isele *et.al.* [52] combined reinforcement learning and neural network to learn active sensing behaviors to grantee safety in a congestion environment. Qiao *et.al.* [53] trained the policies in an intersection scenario with an automatically generated curriculum (AGC) to increase the training speed. However the performance of the deep reinforcement learning approach is closely related to the training data, and it would be very hard to obtain data for all the dangerous cases, which sometimes would lead to unsafe performances.

2.6 POMDP Based Approach

In a Markov decision process (MDP) model, all the states are considered to be definite. However, in the real-world, the states can be partially observable. Therefore, to represent this uncertainty, a POMDP based approach was introduced and had become an emerging technique for solving intersection planning problems in autonomous driving. Bouton *et.al.* [54] navigated the ego vehicle at intersection scenarios where rapid changes occurred by modeling the problem into a POMDP problem and solved the problem with Monte Carlo tree search (MCTS) method, the results showed that the planner was able to outperform a threshold-based heuristic method in safety and efficiency. [55] also formulated the intersection planning and decision-making problem into POMDP problem, the paper modeled the unknown intention of the surrounding vehicles with a probabilistic model and merged that into the observation model of the POMDP model. By solving the POMDP problem, the ego vehicle was able to make continues decisions on a fixed path at intersections on-line. As an extension from [55], [56] extended the intersection planning and decision model so that the model could coop with more than one surrounding vehicles and was capable to perform better results with more computation time, also, the proposed POMDP model was able to operate in continuous state space. Hubmann *et.al.* [20] considered occlusion caused by both static and dynamic objects, and represented the operation of the vehicles in the observation filed and the phantom vehicles as reachable sets. This enabled the ego vehicle to navigate in urban intersections and perform less conservative yet safe actions compared to the baseline method. Brechtel *et.al.* [57] considered several aspects of uncertainty and predicted the results for different actions due to different assumptions in the proposed approach. The authors did not use symbolic representation to properly represent the state space for all the scenarios, but they used a continuous solver to represent specific situations. [58] enabled decision making at generalized congested intersections which avoided both static and dynamic obstacles. The objects in the observable area were predicted and objects in the occluded area were modified with extra attention to avoid severe collisions. The proposed model was able to perform human-like behaviors at unsignalized intersections.

POMDP model takes uncertainties into consideration, which could increase the commuting efficiency of AVs. Solving such problems online using the Monte Carlo tree search is effective, however, it would cost a great amount of computational power. To ensure real-time performance, most approaches use a one-dimensional and highly-discrete action set to reduce the search space but that limits the exploration space. However, using a two-dimensional action set (e.g. vertically and horizontally) would lead to poor searching efficiency.

POMDP is a decision making framework for the agent in a designed scenario. The model embedded the ability of cooperating with uncertainties, for decision-making of autonomous driving vehicles, theses uncertainties include perception uncertainties, surrounding drivers' intention uncertainties and control uncertainties. The agent would receive feedbacks from the reward after a decision had been made at each time step [1]. The major classes of solving the POMDP problems could be divided into exact solution methods and approximate solution methods.

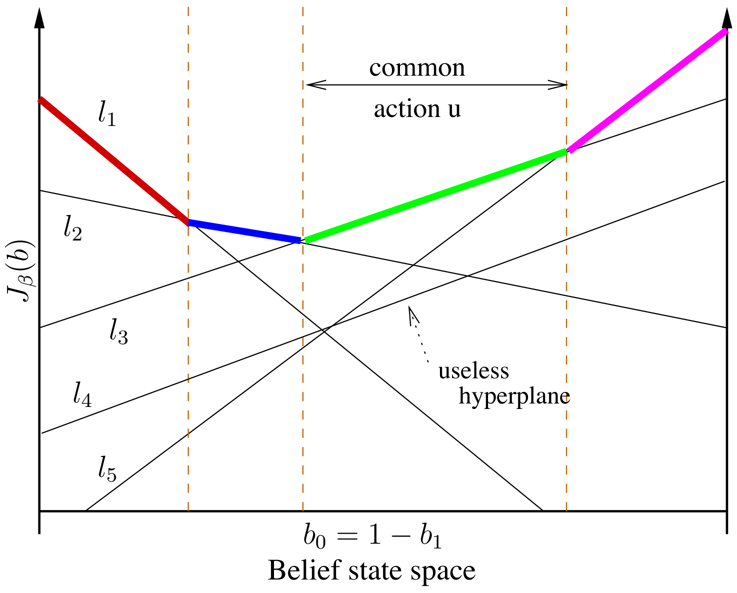


Figure 2.1: Value function representation by hyperplanes in a continuous belif state [1]

2.6.1 Exact Solution

Representation of the Value Functions

The approaches of finding exact solutions of a POMDP model was first introduced by [59] and [60], and were then greatly developed in academia [61]. For finite horizon problems, the value functions of every belief sates could be denoted as piece-wise linear function which is convex. Figure 2.4 is an example of the value function of a two states system with continuous belief state. The figure is a cross-section of a three-dimensional plot defined by available points in the two-dimensional simplex ($b_0 = 1 - b_1$). All the hyperplanes l that represents the value function formed the set $\mathcal{L}$, also, the maximum value of all the hyperplanes along the belief state space formed $\bar{J}_\beta(b)$ in Figure 2.4. Alpha vectors stand as a common representation of the final policy, due to their ability to represent the policy in a compact format. [62].

Value Iteration

One fundamental idea for finding an exact solution is through value iteration. The basic procedure is presented in Figure 2.2, where V stands for the value function, b stands for

```
**Value iteration** ($POMDP$, $\epsilon$)
    **initialize** $V$ for all $b \in \mathcal{I}$;
    **repeat**
        $V' \leftarrow V$;
        update $V \leftarrow HV'$ for all $b \in \mathcal{I}$;
    **until** $\sup_b \mid V(b) - V'(b) \mid \leq \epsilon$
    **return** V;
```

Figure 2.2: Procedure of value iteration [2]

the belief states and H is the value functions mapping. More specifically, the method first traversed all the plans for only one step, then all the corresponding results are evaluated and the plans that are not good enough for any initial belief state are discarded, linear programming could be used to locate the dominating plan at the desired belief state. From the rest of the one-step plans, two-step plans are generated, the results are evaluated, and again, the corresponding results that are not good enough for any belief states are discarded. This international steps stop until the designated horizon is reached or if the so-called Bellman error is smaller than the desired threshold which is written as ϵ in Figure 2.2. Bellman error is the difference between two value functions generated from consecutive steps, which is represented as $V(b) - V'$ in Figure 2.2. In most cases, a huge component of the potential plans are dominated by other plans, tossing the plans that are dominating could limit the load of computation to find the optimal policy. Other methods were also developed for the exact solution of POMDP problems, [63] described an enumeration algorithm that enumerated all the possible hyperplanes to denote the value function and used dynamic programming to update the hyperplanes. Other approaches include, One-pass algorithm [64], Linear support algorithm [65], Witness algorithm [66] and Incremental Pruning algorithm [1, 67].

Weak Point of Exact Solutions

However, for POMDP problems with more states, the complexity of the problem would grow exponentially accordingly. Since belief states needed to be updated when using dynamic programming (DP), the grown states made it nearly impossible to be monitored [68]. Also, with longer horizon length, the problem of representation of value function after operating several steps of dynamic programming increased exponentially. Finally, the process of finding the belief state for which the hyperplane is dominating would cost huge computational effort solving complicated linear programming problems. Even in the case of solving simplified POMDP problem with finite horizon, finding the most optimal policy is still a PSPACE-hard problem [69], and it is an NP-hard problem to find the optimal policy tree using limited nodes [70]. For POMDP problems with an infinite horizon, the number of belief states could be infinite, so does the corresponding hyperplanes. This would lead to undecidable in determining convergence [71]. Considered the burdens of finding exact solutions, many approaches focused on finding an approximate answer, experimental results showed that there exist possibilities of the solver converging to an acceptable but fixed policy in a limited amount of time for very complicated problems. Thus, finding the sub-optimal policy is one of the feasible solutions for very complex POMDP problems [1]. Finding approximate solution could be divided into on-line methods and off-line methods, which will be discussed in the rest of the section.

2.6.2 Offline Approximate Solution

Common offline methods do all of the computation before implementing the policies and represent the policies in either finite-state controllers or alpha vectors. Some other approaches do most of the computation by finding the upper and lower bounds for setting the optimal value function, which increases the efficiency of the online search by directing the search to the appropriate direction and help to estimate the long-term reward.

Blind Policy

The blind policy is an approach for obtaining lower bound for a POMDP online solver, in this method, no matter how the belief states changes, the planner would select the same action. Since each static policy must be equal or less optimal than the best policy, thus, resulting value functions (usually specified by an alpha-vectors) of these policies stand as lower bounds of the solution. Where these alpha-vectors could be represented as:

$$\alpha_{t+1}^{a}(s) = R(s,a) + \gamma \sum_{s' \in S} T\left(s,a,s'\right) \alpha_{t}^{a}(s), \tag{2.1}$$

where $\alpha_0^a = \min_{s \in S} R(s,a)/(1-\gamma)$. The alpha-vectors could be used to find the value of the lower bound of a belief state. This method requires much less computation power. However, the lower bounds obtained are usually far from optimal, thus it does not imply much information [2, 72].

MDP and QMDP

The MDP based approach approximates the value function of a POMDP problem into its underlying MDP [73], the approximated value function can be calculated with Bellman's equation under the MDP framework

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T\left(s'|s,a\right) V^*\left(s'\right) \right). \tag{2.2}$$

Similar to the MDP approach, QMDP [73] approximates the POMDP model to a MDP model in some extent. The major concept of the algorithm is that it assumed that all the uncertainties (partial observability) disappear after the first step. Then a set of alpha-vectors corresponding to each actions are being created. Based on the value function of a MDP problem $Q(s,a)$, alpha-vectors could be computed through value iteration as:

$$\alpha_a^{(t+1)}(s) = R(s,a) + \gamma \sum_{s'} T\left(s'|s,a\right) \max_{a'} \alpha_{a'}^{(t)}\left(s'\right), \tag{2.3}$$

where $\alpha_a^{(0)}(s) = 0$. If enough iteration has been done, the set of alpha-vectors could be able to approximate the value function. This approximation formed a upper-bound on the value function that is tighter than the MDP approach. This approach could encounter limitation in information-gathering actions, but has shown to be applicable in many real life scenarios where different choices of actions could not significantly reduce the uncertainty of the states [62].

Fast Information Bound

Fast information bound algorithm [2] considers some level of uncertainties of the POMDP model, a alpha vector would be generated for every action and instead of using iteration method as QMDP (Equation 2.3) to obtain the alpha vectors, the method calculates the alpha vectors by:

$$\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{o} \max_{a'} \sum_{s'} O\left(o|s',a\right) T\left(s'|s,a\right) \alpha_{a'}^{(k)}\left(s'\right). \tag{2.4}$$

Though fast information bound is a little more complicated than QMDP, this approach provides a upper bound that is tighter or equally tight upper bound compare to the QMDP method, where the complexity is usually manageable.

Point-based value iteration (PBVI)

PBVI method family are very popular approaches for solving POMDP problems with a finite set of belief points. The general PBVI approaches include two major parts, that are limiting the size of the value function through representing the value of the problem at the finite and reachable subset of the beliefs, and also using point-based algorithms to optimize the value function. In other words, the general procedure of solving point-based value iteration is to collect belief subset then update the value over the select beliefs until the stopping criterion is reached. Different approaches in this PBVI family have different ways of achieving the two major parts, and the stopping criterion is relevant to the different choices of the two major parts.

The first point-based value iteration method was introduced by Pineau *et.el.* [74]. The algorithm starts by initiating the belief state ($B_0 = \{b_0\}$), after that, the algorithm updates the value function using a point-based Bellman backup for each point of the current belief set b_i. Which could be represented as

$$V_B^{j+1} = \left\{ \text{ backup } \left(b, V_B^j\right) : b \in B \right\}. \tag{2.5}$$

The Bellman backup is operated in a random order and this procedure stopped when$V_B^j = V_B^{j+1}$, or the predefined rounds of iteration have been reached.

Then to achieve better results, the algorithm selects a successor $b^{'}$ from each belief state b which is furthest from the set B. Let L represents the selected distance metric, we have:

$$|b' - B|_L = \min_{b \in B} |b - b'|_L \,, \tag{2.6}$$

and forward simulation is used to generate the candidate successors:

$$b' = \max_{a,o} |b^{a,o} - B|_L \,. \tag{2.7}$$

In this way, the successor points (a single $b^{'}$ for every $b \in B_i$) are added into B_i, the newly added points and the original points in B_i formed the new set B_{i+1}. The detailed procedure is written in Algorithm 2 [75].

PBVI intended to choose belief points in the reachable belief space in an even manner, and tried to fully span the reachable space in the designated horizon. However, the approach searches greedily in the local space, thus it may not be able to provide a full span for the current belief. But still, the error between the value functions is limited by the linear function over the belief subset [74]. Thus, the value converged to optimal value under certain circumstances [75].

Algorithm 2 PBVI

```
1: function PBVI
2:     while $V$ has not converged to $V^*$ do
3:         Improve($V, B$)
4:         $B \leftarrow$ Expand ($B$)

1: function Improve(V, B)
2:     repeat
3:         for each $b \in B$ do
4:             $\alpha \leftarrow$ backup($b, V$)
5:             //execute a backup operation on all points in $B$ in arbitrary order
6:             $V \leftarrow V \cup \{\alpha\}$
7:     until $V$ has not converged,
8:   //repeat the above until  V  stops improving for all points in  B

1: function Expand(V, B)
2:     $B_{new} \leftarrow B$
3:     for each $b \in B$ do
4:         Successors($b$) $\leftarrow \{b^{a,o} | \Pr(o|b, a) > 0\}$
5:         $B_{new} \leftarrow B_{new} \cup \text{argmax}_{b' \in S_{uccec\,\text{sarc}(b)}} \|B, b'\|_L$ //add the furthest successor of $b$
       return $B$ new
```

Perseus

The PBVI algorithms family focuses on the proper expansion of the belief state set B since the expansion is relatively hard, also the set B decides the size of the value function V. If a huge set of beliefs could be collected easily and the set lead to a compact V, the efficiency of the point-based Bellman backups would be monitored without having complicated belief selection procedures. The Perseus algorithm first takes an initial belief set and does exploration to expand the set until a sufficient number of points have been expanded and collected. after exploration in the belief space, the algorithm computes the value function iteratively. At the start of each iteration, the set is $B' = B$ and the value function is empty, then a belief point from the belief set is sampled randomly and is used to compute a new alpha-vector α_{new}. If $\alpha_{new} \cdot b' > V(b')$, then the new alpha-vector is added to the empty value function. Then all the belief points that the value is improved by the alpha-vector are deleted, thus would not be backed up at that iteration cycle. If $\alpha_{new} \cdot b \leq V(b)$ then $\alpha_{old} = \text{argmax}_{\alpha \in V} \alpha \cdot b$ is added to V'. The iteration ends until $B' = \phi$ and V is set to V'. The algorithm is also presented in Algorithm 3 [75].

This algorithm has two major advantages, the belief expansion and collection is relatively fast. Also, the value function is created by a set of alpha-vectors that are much smaller than $|B|$ in the value update process. This could enable the algorithm to collect the belief points much bigger than PBVI methods, therefore, shows the potential to cover the reachable belief space better. However, this method also suffers from some disadvantages. The pure random selection strategy to back up the beliefs means the exploration is uni-directional, thus, there usually exists unneeded backups that did not help the value function to converge. Also, for more complicated problems, the algorithm needs to explore a large set of beliefs, thus would be hard to operate [75].

2.6.3 Online Approximate Solution

Offline methods pre-calculates the policies covering any possible belief states, this is usually applicable for problems with small or medium complexity. For large POMDPs, offline methods would take a great amount of time to generate policies, and the approximation of the value function performs badly in very large domains. Therefore, online methods became a good alternative for solving large POMDP problems. Instead of considering finding the alpha-vector that is optimal for the whole process, online methods only focus on finding the local optimal, that is, generating the optimal solution in a fixed horizon based on the current belief. This allows the solver to only deal with a small set of beliefs, thus reduces the computation effort and decreases the time for policy generation compared to offline methods. Since the solver has to operate in real-time, the solver has very limited time for policy construction, this remains a major concern for most online solvers.

The general online planning framework includes a planning process and an execution process that are operated successively and repeatedly. During the planning process, a tree representing the policy for beliefs on each time frame is generated. The tree represents the policy by including possible sequences of actions and observations from the current belief, it represents the beliefs with OR-nodes which had actions as their children, where the action nodes are included between every layer of belief nodes, which are presented as AND-nodes. The major procedure is as follows. The algorithm first initializes the root of the tree with the initial belief, then it selects the following fringe node, after that, the next reachable beliefs node are generated and the value for this node is calculated, finally, the value is backpropagated to its ancestors. This cycle continues until some termination condition is reached (e.g computation time for policy tree construction). Values of beliefs are estimated by backpropagation from fringe nodes all the way back to the root nodes. Upper bounds and lower bounds are sometimes pre-calculated as the initial value of the beliefs. The execution phase then selects the action with the biggest value from the current

Algorithm 3 Perseus

```
1: function Perseus
2:     $B \leftarrow$ Random Explore($n$)
3:     $V \leftarrow$ PerseusUpdate($B, \phi$)

1: function RandomExplore(n)
2:     $B \leftarrow \phi$
3:     $b \leftarrow b_0$
4:     repeat
5:         choose a random successor of $b$
6:         Choose $a \in A$ randomly
7:         Choose $o \in \Omega$ following the $\Pr(o|b, a)$ distribution
8:         Add $b^{a,o}$ to $B$
9:         $b \leftarrow b^{a,o}$
10:        $V \leftarrow V \cup \{\alpha\}$
11:    until $|B| = n$

1: function PerseusUpdate(V, B)
2:     repeat
3:         $B' \leftarrow B$
4:         $V' \leftarrow \phi$
5:         while $B' \neq \phi$ do
6:              // Choose an arbitrary point in $B$ to improve
7:              Choose $b \in B'$
8:             $\alpha \leftarrow$ backup($b, V$)
9:             if $\alpha \cdot b \geq V(b)$ then
10:                 // Remove all points from $B$ that was already improved by the new $\alpha$
11:                $B' \leftarrow \{b \in B' : \alpha \cdot b < V(b)\}$
12:                $\alpha_h \leftarrow \alpha$
13:            else
14:                $B' \leftarrow B' - \{b\}$
15:                $\alpha_b \leftarrow \text{argmax}_{\alpha \in V}\, \alpha \cdot b$
16:            $V' \leftarrow V' \cup \{\alpha_b\}$
17:        $V \leftarrow V'$
18:    until $V$ has converged
```

root belief node and executes the action, the tree will then updates the root belief with the obtained observation.

Most approaches improve the performance of the algorithms by reducing the number of beliefs of the policy tree. The major difference between the different approaches include ways of expanding and constructing the next belief nodes and selecting the next fringe node from the current node. The major online planning approach includes Branch-and-Bound Pruning, Heuristic Search, and Monte Carlo Sampling.

Branch-and-Bound Pruning

Brach-and-Bound pruning is an approach that reads the information from the upper and lower bounds of the value function to trim the search tree. With pre-computes lower bound and upper bound, the tree only grows necessary lower nodes and prune the sub-optimal nodes. Some popular offline methods mentioned in Section 2.6.2 could be used for finding the bounds. Blind policy selects the same action for all cases which could be used form the lower bound, while the MDP, QMDP and the fast informed bound could work as the upper bound. These bounds are maintained on the value of actions of each belief nodes in the tree. If the upper bound of an action a in a belief b is lower than another action $\tilde{a}$ on the same belief, then value of that action $\tilde{a}$ on that node must be greater than the value of action a ($Q^*(b, \tilde{a}) \geq Q^*(b, a)$). Therefore, the a is suboptimal on belief b, thus, this branch will be deleted and the children belief node of the action a in belief b will not be considered. First, the bounds are applied on the fringe nodes then through backpropagation, the bounds are applied to the parent nodes. It can be easily concluded that the tighter the bounds are chosen, the more portion of the tree would be pruned, thus the less computation effort is needed to select an action. The detailed algorithm is presented as Algorithm 4 [62]. Where $\underline{U}$ represents the lower bound, $\bar{U}$ denote the upper bound, b represents the belief and d represents the depth of the tree.

Monte Carlo Tree Search

Monte Carlo methods are a popular approach that is based on repeated random sampling. Instead of setting up heuristics straight away and search and pruning based on the heuristics, which is very effective, it obtains and assesses the states through sufficient random simulations. Because the main idea of the method is, with more and more random simulation, the strategy will be derived closer and closer to a successful strategy. In problems that have large state space, precise representation of the probability function of the transition

Algorithm 4 Branch and bound online policy

```
1:  function SelectAction($(b, d)$)
2:      if $d = 0$ then return $(\text{NIL}, \underline{U}(b))$
3:      $(a^*, \underline{u}) \leftarrow (\text{NIL}, -\infty)$
4:      for $a \in A$ do
5:          if $\bar{U}(b, a) \leq \underline{u}$ then return $(a^*, \underline{u})$
6:          $u \leftarrow R(b, a)$
7:          for $o \in O$ do
8:              $b' \leftarrow \text{UpdateBelief}(b, a, o)$
9:              $(a', \underline{u}') \leftarrow \text{SelectAction}\,(b', d - 1)$
10:             $u \leftarrow u + \gamma P(o|b, a)\underline{u}'$
11:         if $u > \underline{u}$ then
12:             $(a^*, \underline{u}) \leftarrow (a, u)$
        return $a^*, \underline{u})$
```

model $\mathbb{T}$ and observation model $\mathbb{O}$ can be hard. Thus, a generative model $\mathcal{G}$ also known as the black box simulator is used to overcome this problem. The model takes in the states and actions in the current time frame, and output a sample of the received reward, the observation and the state in the next time frame, which applies similar dynamics as the POMDP problem. The Monte Carlo methods includes simple roll-out methods that dose pure random sampling where no search tree is constructed [76] and fixed horizon depth-first search [77] and also MCTS methods [78]. MCTS is a very popular approach for solving online POMDP problems, the method is possible to be applied to any problem that could be represented by state-action pairs, the tree generated by this method is represented in Figure 2.6.3. Where a represents the actions, o represents the beliefs and b is the belief. The tree nodes of a MCTS represents the states of the problem. Each state is associate with a action formulate the visitation count of the node $N(s, a)$, and the cumulative value $Q(s, a)$ of all the simulation that passed that node. The empty history started from the initial belief is represented by the root node, where a history is a sequence of past observations and actions [62]. The tree grows with sequences of layers, different layers alternate between action node layer and observation layer.

The method builds the search tree by making random simulations from the initial node of the tree until termination condition is reached. The simulation follows four major steps, that is selection, expansion, simulation and backpropagation, which is also presented in Figure 2.6.3. At the selection phase, the algorithm starts from the root of the tree and executes actions base on that node to reach another node, then from the new node, it

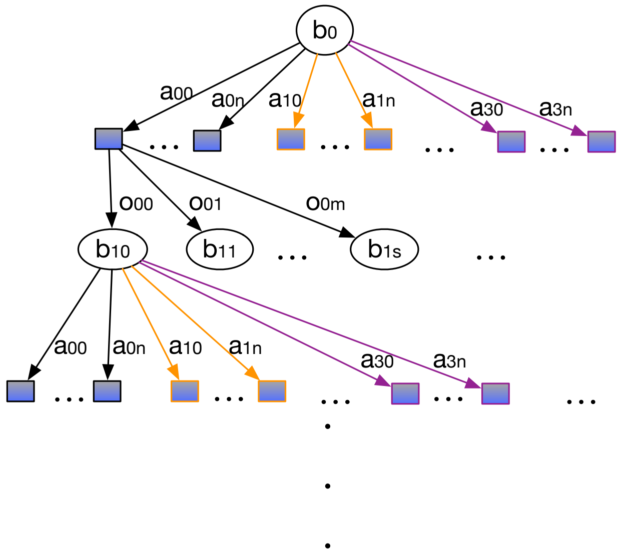


Figure 2.3: Tree constructed by MCTS

executes actions again to reach another node, this goes on until the algorithm reached the leaf node. Different heuristic selection strategy can be applied during this process, some approaches focus on balancing between exploitation and exploration. After that, the expansion phase, one or more new children is generated from the leaf node. Then, at the simulation phase, if the previously generated leaf node(s) does not indicate the terminal state, a simulation would be operated with sequential actions until the terminate state is reached. Finally, the value of the previously generated node(s) obtained from the simulation phase will be backpropagated to all the nodes in the history from the root node to the leaf node. Then this process is repeated X times to generate the desired policy. At the next time step, the children of the root node corresponding with the selected action and actual observation becomes the new root node. The MCTS is an anytime algorithm and has shown that with a sufficient number of iterations, this method could converge very close to the optimal solution or obtain an optimal solution. Also, prior knowledge could be applied to the algorithm through setting initialized visitation count N_0 and cumulative value Q_0 in of the node as well as the simulation phase where rollout policy could be applied.

MCTS has three major advantages. First, the tree could be searched and grew in the direction where it is worth more computation effort to increase the search efficiency, states that have the potential to lead the tree to an optimal policy are visited more often. Second, the algorithm could be stopped at any time, if computational time is limited, the algorithm

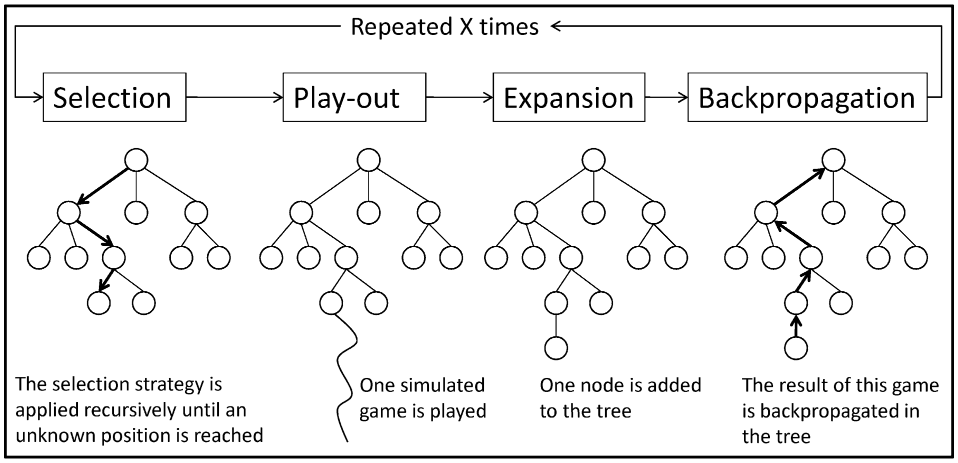

Repeated X times
Selection
Play-out
Expansion
Backpropagation
The selection strategy is applied recursively until an unknown position is reached
One simulated game is played
One node is added to the tree
The result of this game is backpropagated in the tree

Figure 2.4: Outline of Monte Carlo Tree Search [3]

could be able to provide the so-far optimal policy for execution [79].

2.7 Summary

Different methods to solve left-turning planning and decision-making of autonomous vehicles are evaluated in this chapter. Among all the approaches, the POMDP based approach has shown strong potential of providing high commute-efficient left-turning decisions of the ego vehicle while avoiding collision with surrounding vehicles. The exact solutions, online and offline approximate solution methods are introduced. The online approximate solution methods had outperformed other approaches in dealing with complex POMDP problems in a timely manner, which would suit our complex left-turning problem.

Chapter 3

Left-turning Human Driving Behavior Analysis

3.1 Introduction

With the objection of enabling the intersection planner to perform actions like experienced drivers which are not only more likely to provide higher commute efficiency at intersections and also the human-like behaviors of the autonomous vehicle would be easier for the surrounding human drivers to predict, thus, increases the safety of the planner since it is easier to be coop with. Therefore, inspired by the real-life driving experience and naturalistic driving datasets, a novel crtical turning point (CTP) has been borough out, and the CTP model which could be applied in various shaped intersections has been proposed. Finally, the model is verified through more thorough measurements of the left-turning behaviors in the datasets.

3.2 Left-Turning Behavior Analysis

To obtain a perceptual understanding of how people drive at intersections, a quadcopter was used to record the driving behavior of people left-turning at intersections. Videos were taken from a bird-eye view at a few major intersections in urban areas of China. The videos include both heavy traffic and light traffic scenarios at regular-shaped intersections and irregular intersections.

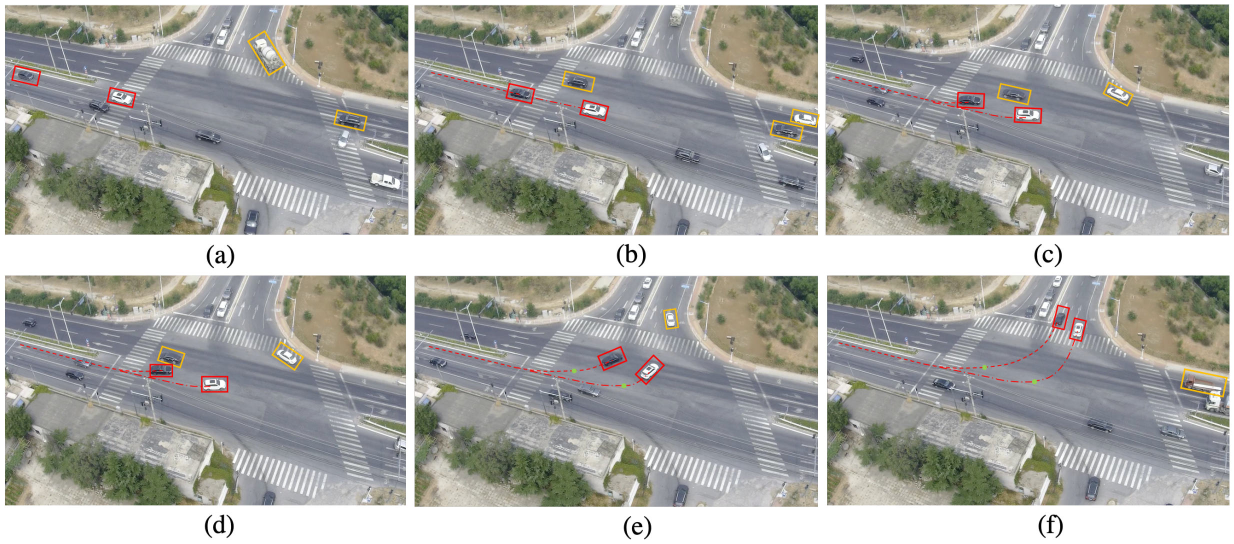


Figure 3.1: Real-life driving behavior of left turns

From the large amount of real-life driving video, left-turning behaviors in different traffic are observed. It is found that when left-turning at intersections, if there exist oncoming traffic (the left-turning path is blocked), the ego vehicle would creep forward and wait in the middle of the intersection. If the road is clear, the vehicle would then make a sharp turn, which the left turn process is demonstrated in Figure 3.1. The red bounded vehicles are the left-turning vehicle, and the yellow bounded vehicles are the oncoming vehicles. The dashed line and the dash-dotted line indicate the history path of the two left-turning vehicles accordingly. When the oncoming road is clear, the vehicle would steer the vehicle to make the turn at the beginning of the intersection, and the average path is more round compare to left-turning paths with heavy oncoming traffic.

It is also observed from the videos that at the same intersection, if there exist oncoming vehicles, the ego vehicles would make sharp turns at different locations. This is because the future path is cleared before the ego vehicle reached the waiting point, thus, the shift from creeping forward to sharp steering happens at different timing. This phenomenon can also be found in Figure 3.1. In Figure 3.1 (c) it could be observed that the two vehicles bounded by red rectangles are creeping forward due to the occlusion caused by the yellow bounded vehicle. In Figure 3.1 (d) shows that the white vehicle is taking the sharp turn while the black vehicle with the red bound is still creeping forward because the future path is still blocked by the yellow bounded vehicle. Then in Figure 3.1 (e) the path is clear for

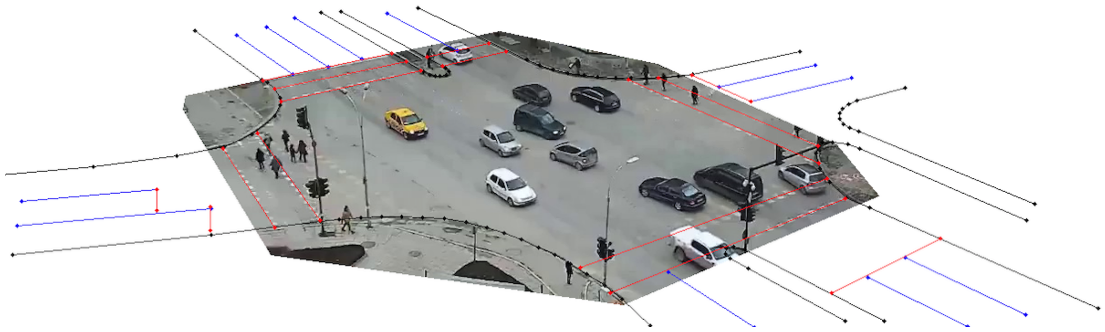


Figure 3.2: Selected scenarios in the INTERACTION dataset [4]

both of the vehicles, and different points have been chosen to make the sharp turn.

Observing through recorded videos only gives a rough and intuitive understanding of left-turning behaviors. To understand this behavior from an in-depth perspective, human drivers' driving behaviors are being studied through two intersection datasets. Which is the InD Drone dataset [80] and the INTERACTION dataset [4]. The InD dataset is a recently published dataset that records naturalistic vehicle and pedestrian trajectories at four different locations in Germany. The data was collected by a drone, thus, there is sufficient information in the occlude environment. The INTERACTION dataset is also a dataset that records the naturalistic traffic participants' trajectories in various kinds of highly interactive driving scenarios which include irregular intersections. The major difference from the INTERACTION dataset and the InD dataset from our perspective is that the INTERACTION dataset includes irregular intersections that the road is not strictly perpendicular to each other, and also that intersection includes median strips. The stereoscopic representation of the selected intersection is presented in Figure 3.2. The difference and irregular shape of the intersection enable more generalized driving behavior observation.

A regular intersection in the InD dataset is evaluated for observing driving behaviors at regular intersections. First, the left-turning trajectories are extracted by restricting the starting point and the ending point of all the trajectories of traffic participants. The result of all the extracted trajectories is presented in Figure 3.3. The figure shows that the left-turning curves cover a a recognizable portion of the left turning space, and different people would take the sharp turns at different locations along the road.

The yaw rate and longitudinal speed along the left-turning paths are also extracted, and are plotted with respect to the x-axis in the global coordinates, as shown in Figure 3.4 part (a) and (c)), and the yaw rate and speed are also plotted with respect to the local

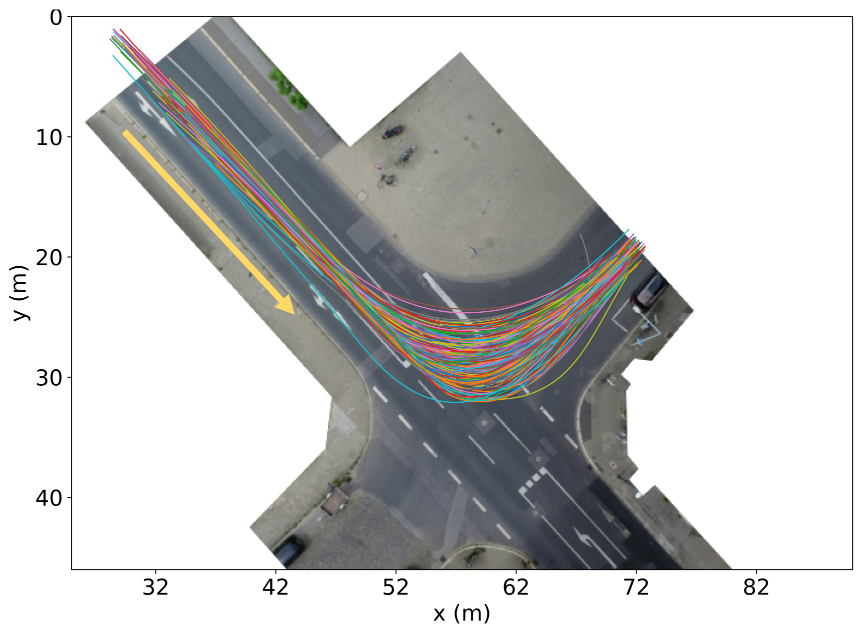


Figure 3.3: Extracted left-turning trajectories from naturalistic dataset

coordinates (the direction of the initial heading angle of the ego vehicle which is also presented as the yellow arrow in Figure 3.3) as shown in Figure 3.4 part (b) and (d). Figure 3.4 part (a) and (b) show that, most of the vehicle comes to a full stop or decrease to a low speed in both coordinates (the lowest speed of the trajectories at the intersection is plotted as red dots), then increase their speed afterward. Figure 3.4 part (b) and (d) show that at certain points the yaw rates of the vehicles increase sharply and eventually reach a very high value (the biggest yaw rates are indicated with red dots), which indicates that the ego vehicle is making a sharp turn. Through comparing part (a) and (c) and part (b) and (d) of Figure 3.4, it is observed that the positions of the points where the ego vehicle is making the left turn and the positions of the points where the vehicles decrease their speed to the lowest value are in the same range. This could indicate that when making left-turns, most drivers would first decelerate its speed or even comes to a full stop at a certain point, before accelerate and make a sharp turn to finish the left turn.

Similar data extraction and processing methods are used to observe the INTERACTION dataset as well. In the INTERACTION dataset, an irregular-shape intersection has been studied, this intersection not only have median strips, but also have non-perpendicular crossroads. The extracted yaw rates also follow a rapid increase and decrease pattern, and at the same time, most of the vehicles decelerate to its lowest speed at the location where

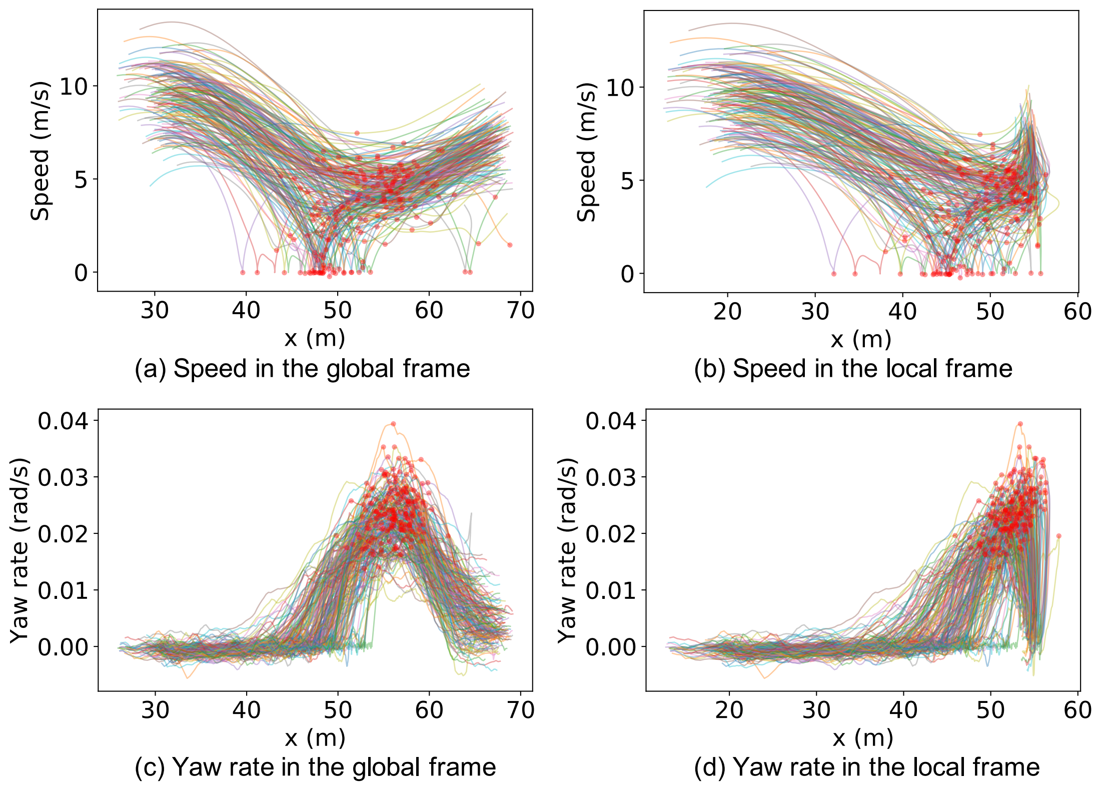


Figure 3.4: Yaw rate and speed along the global coordinates and left-turning paths

the vehicle is making sharp steering. This shows that the left-turning phenomena have been observed from the INTERACTION dataset too. This shows that the observed phenomena exist in different kinds of intersections, and has shown its universality.

3.3 Critical Turning Point Model

From the observation of the left turning behaviors in the previously mentioned videos, it could be concluded that: for left turns at intersections, there are 2 phases: a creeping forward phase that the driver assesses the safety of the environment, and a intense steering phase when the driver feels confident to drive through the potential collision area (red shaded area in Figure 3.5). The different points where the driver makes the phase change are crucial to the safety and commuting efficiency of the left-turn vehicle, therefore, these

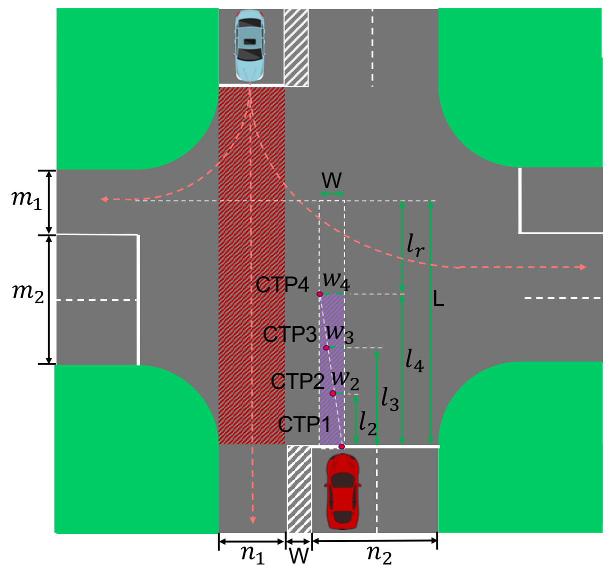


Figure 3.5: Critical turning point model

points are defined as critical turning points (CTPs). To be able to extract CTP in various shapes of intersections, a generalized and parametric CTP extraction model is proposed as shown in Figure 3.5. m_1, m_2 are the number of the horizontal road lanes and n_1, n_2 are the number of the vertical road lanes. W is the width of the median strip (white shaded area in Fig. 3.5) and also the width of the purple-color rectangle. and L is the horizontal distance from the starting point to the extension of the goal lane.

As shown in Figure 3.5, all the CTPs are allocated on the purple-color rectangle in Figure 3.5. The longitudinal and lateral distance to the starting point of each CTP is defined as:

$$\begin{cases} l_i = k_{l,i} \cdot (L - l_r) \quad i \in 1, 2, \ldots, q \\ w_i = k_{w,i} \cdot W \quad i \in 1, 2, \ldots, q, \end{cases} \tag{3.1}$$

where

$$l_r = c_r \cdot R_{min}, \tag{3.2}$$

i represents the number of a CTP, q indicates the total number of CTPs. $L - l_r$ is the length of the rectangle where l_r is calculated by multiplying a slackness ratio c_r with the vehicle's minimum turning radius R_{min} as shown in Equation 3.1.This improves the trajectory-tracking feasibility of our proposed planner at a kinematic level. Finally, $k_{l,i}$ and $k_{w,i}$ are the longitudinal and lateral distance coefficients that allocate the CTPs in the rectangle. L which is also presented in Figure 3.5 is closely related to the number of horizontal road lanes m_2 and the road width, with bigger intersections, the CTPs will be allocated in a long purple shaded area, thus, this ensures the generality of the CTPs extraction model. The width of the rectangle is only related to the width of the median strip. If the median strip does not exist, the width is defined as 0, in that case, all the CTPs lies evenly on a vertical line.

The CTP concept is proposed by observing the videos of naturalistic behaviors of vehicles at intersections, however, this concept is only proposed through rough visual measurements, thus, to obtain more accurate measurements of this phenomenon, it is appropriate to analyze some more detailed left-turning data for more comprehensive observation. Also, though the parameters (e.g. $c_r, k_{l,i}, k_{w,i}$) can be obtained with personalized modification. It would be ideal to extract the recommended parameters from studying naturalistic driving data, and thus with the recommended parameters, the model would be able to extract CTPs in a more human-like manner. Therefore, this extraction model is verified and the parameters are approximated by naturalistic driving datasets in the next section.

3.4 CTP Model Identification and Validation

The rationality and the recommended parameters of the CTP model need further evaluation and verification. More accurate measurements of this phenomenon and the exact location of the turning points need to be extracted from the datasets. By definition, the point where the vehicle is making changes in the left-turning phase is the CTP. Therefore the point where the yaw rate is starting to increase rapidly is one possible representation of the aforementioned CTP. However, the data is interference by noise caused by lane-keeping steering. To overcome the disturbance of the noise, a threshold is set and is presented as the green dashed line in Fig. 3.6 part (a). If the average yaw rate in five consecutive time steps is greater than the threshold value, then the position at the first time step of the consecutive five steps is determined as the CTP. The average value further diminished the influence of the pulse-like noise, and makes it possible to extract the CTP in a more sensitive manner. The distribution of the longitudinal-directional location of CTPs are plotted as a histogram in Fig. 3.6 part (c), different bin widths are set to present the distribution in various microscopic aspects. This histogram shows that most of the CTPs distributed around the end corner in the longitudinal-direction (around 45m to 53m).

The points in the left-turning trajectories where the vehicles decrease the longitudinal speed or come to a full stop is another possible representation of the CTPs. The location of the lowest speed is presented by red dots in Fig. 3.6 part (b), and the distribution is

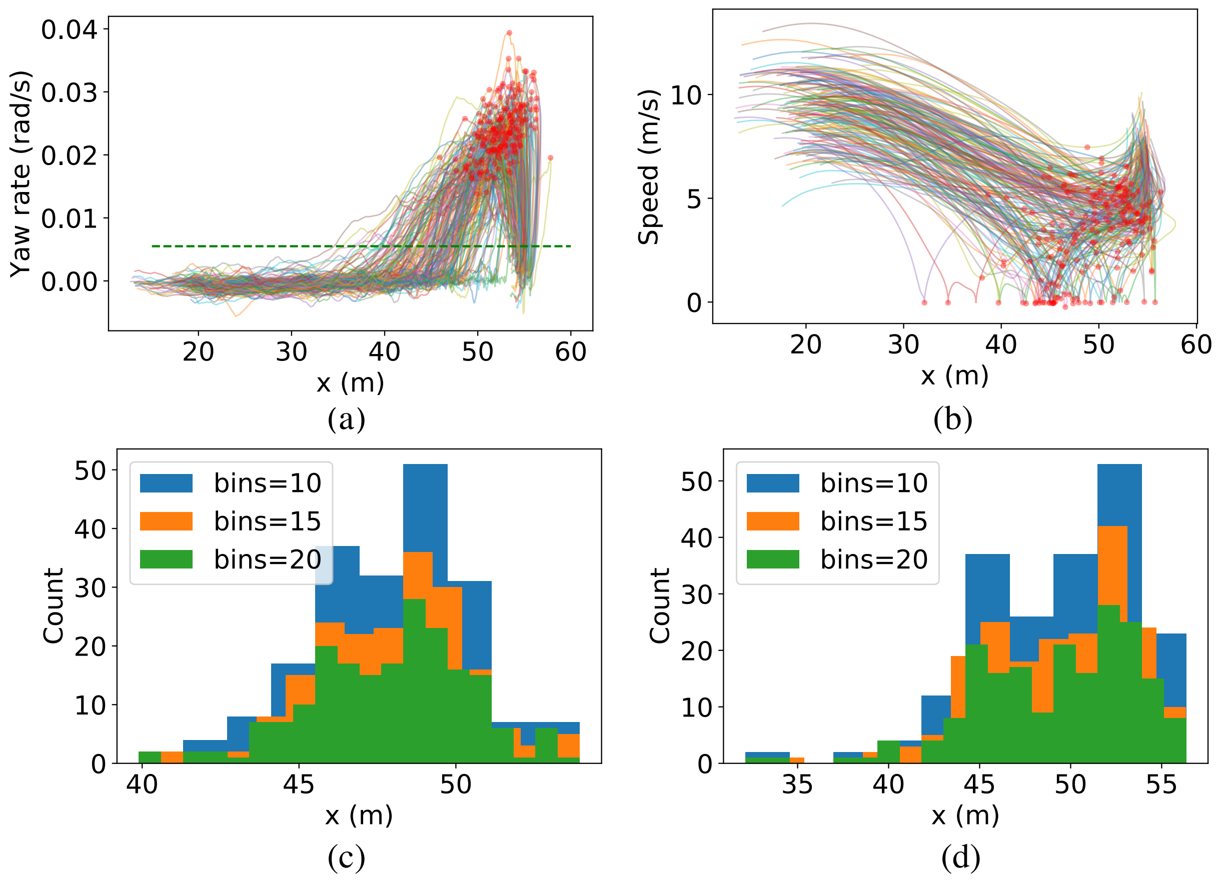


Figure 3.6: CTP location distribution in the local frame

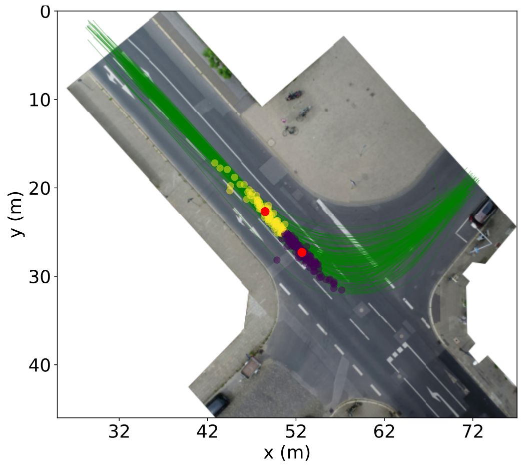
presented in Fig. 3.6 part (d). This distribution is similar to the CTPs extracted considering yaw rates. There is a little offset between the two distribution, this is because both CTP extraction criteria could lead to a very small portion of mistakenly extracted CTP position. Drivers some times decrease speed or comes to a full stop because they are stuck in heavy traffic, or they are responding to emergency situations. Therefore, when longitudinal speed is used as the criteria, the position of the point with the lowest speed may not always represent the CTP. If using the yaw rate as the criteria, the points where the yaw rate increases to its maximum are the exact CTPs, but the noise caused by steering makes it hard to extract the position precisely. Therefore, comparing the two distributions, the yaw rate is selected as a major criterion because all the CTPs can be extracted, though the positions are not precise, the approximation is still acceptable.

With the extracted CTP for every left-turning trajectories, which are plotted as semi-transparent dots in Figure 3.7, the average positions of the CTPs could be obtained through K-means clustering. K-mean clustering is a popular partitioning algorithm for clustering. K Cluster centers (centroids) are repeatedly chosen randomly and the distance between each point to their closest centroids is being calculated until the algorithm converges. This is used to find the minimum sum of squared distances (Euclidean distance) from the points to their closest centroid. The object function which is also known as the sum-of-squared errors (SSE) [81] is presented in Equation 3.3.

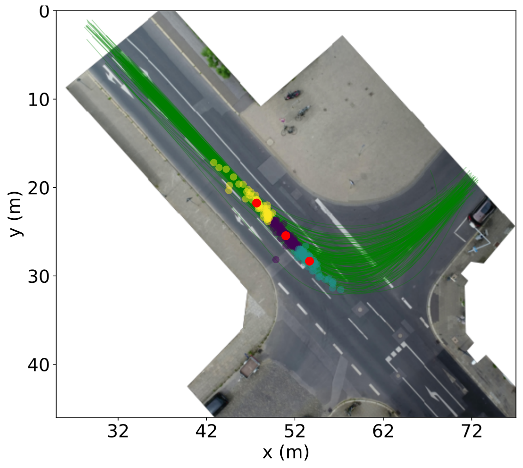
$$O(C) = \sum_{k=1}^{K} \sum_{p_i \in c_k} \left\| p_i - \mu_k \right\|^2 , \tag{3.3}$$

p_i is the two-dimensional vector representing the position (x, y) of every extracted point, and μ_k is the mean of the corresponding cluster c_k, K represents the total number of clusters while the goal is to minimize this objective function $O(C)$. Then the average point of each cluster can be obtained, and Figure. 3.7 is a demonstration of the case of two to four clusters being taken. As shown in the figure, when different numbers of clusters are being selected, the clustered average positions of CTPs follow an isometric pattern, which also shows that the CTPs are located evenly along the road. In reality, the number of clusters could be different case by case, but the extracted naturalistic driving behaviors showed that they could all be represented by Equation 3.1. For easier implementation, better generalization and without much loss of feasibility, we assume that the CTPs located evenly along the longitudinal axis.

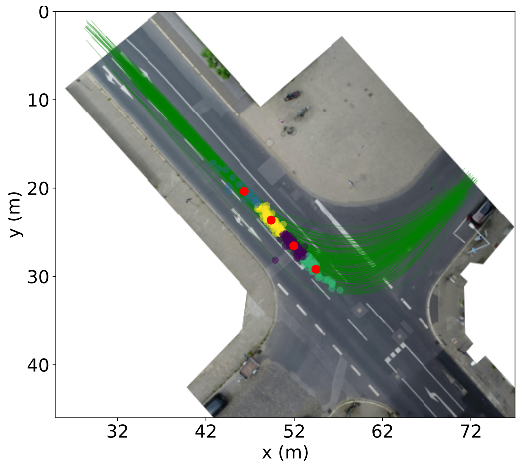
The different driving styles are also evaluated through evaluating the maximum yaw rate and maximum lateral acceleration of the left-turning vehicles, the distribution is shown in Figure 3.4. Different colors represented the number of equal-width bins in the range. The



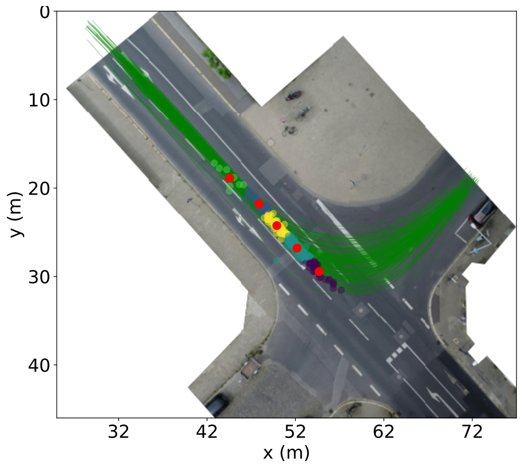
(a)



(b)

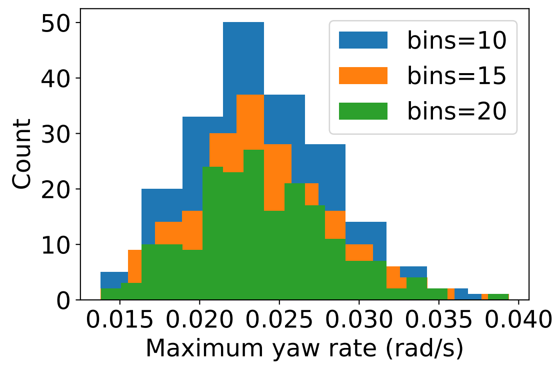


(c)

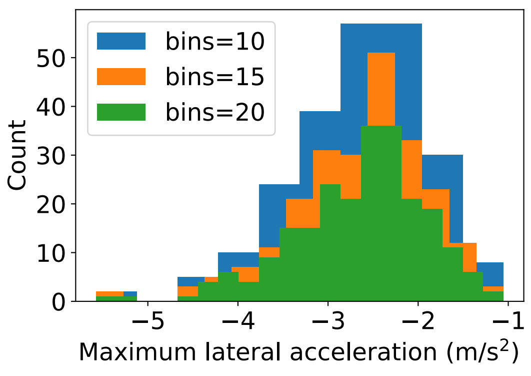


(d)

Figure 3.7: CTPs extracted from regular intersection dataset



(a)



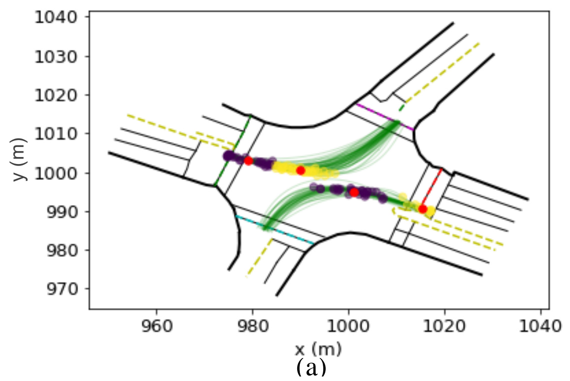
(b)

Figure 3.8: Driving style evaluation

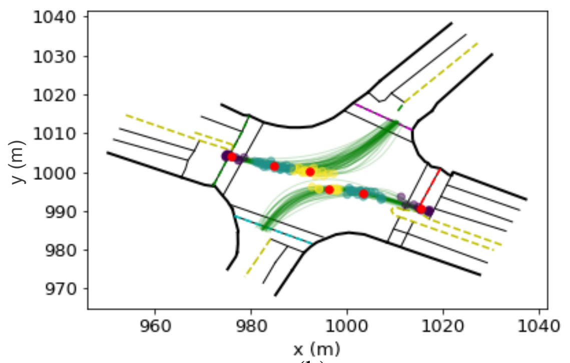
count for both maximum yaw rate and maximum lateral acceleration all follow a normal-like distribution, thus most of the driver would take 0.02 rad/s to 0.027 rad/s as their maximum yaw rate, and -1.8 m/s^2 to -3 m/s^2 as their maximum lateral acceleration. As one of our goals is to perform human-like driving behaviors, for the surrounding vehicles to better understand the decisions of the ego vehicle. It would be appropriate to limit our most intensive behavior in that domain. The figure also shown feasibility for more conservative driving style for the maximum lateral acceleration can be greater than -1.2 m/s^2 and the maximum yaw rate can be low as 0.015 rad/s^2, same with more aggressive driving style. With this evaluation, different dynamic constrains for different driving styles could be applied for more personalized autonomous driving.

Same as extracting the InD dataset, the left-turning trajectories by the staring and ending position of the left-turning vehicles, which is also presented in Figure 3.9. Then the yaw rate is plotted for every trajectory and the threshold is set to extract all the CTPs, and finally the position of each cluster of CTP is grouped as shown in Figure 3.9.

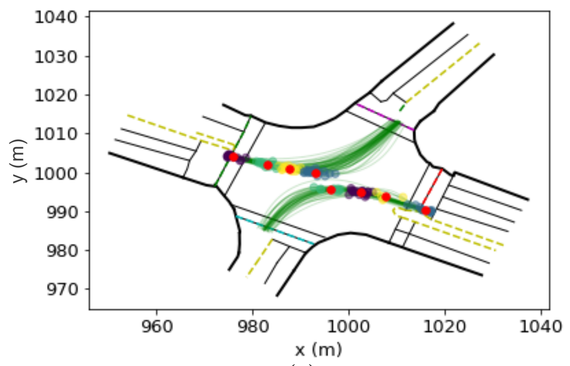
From Figure 3.9 it can be observed that the red dots does not locate in the direction of the road, but have offsets in the horizontal direction due to the existence of the median strip. The CTP model considered the existence of median strips and thus, the CTPs could still be presented using Equation 3.1. Also, with different numbers of CTPs choices, the average locations of the average CTPs still roughly follow a isometric pattern. Therefore, a similar simplification assumption for the general CTPs location on the lateral direction is made, that the general CTPs located evenly along the lateral axis.



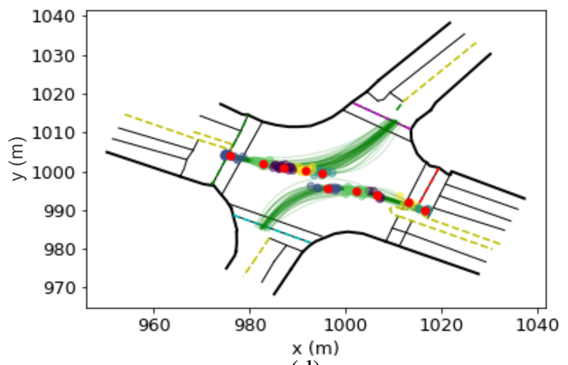
(a)



(b)



(c)



(d)

Figure 3.9: CTP extracted from irregular intersection dataset

3.5 Summary

In this chapter, a CTP concept is proposed by observing several real-life intersection driving videos and analyzing naturalistic intersection driving datasets. Then using these observations, a generalized and parametric CTP model is proposed. To verify the rationality of the CTP concept and provide recommended parameters for the extraction model, two recently published intersection datasets are used for evaluation. Left-turning trajectories and the corresponding CTPs are extracted and the CTPs are grouped to extract the average positions of CTPs. The dataset analysis showed that the average location of the CTPs could be simplified to an isometric pattern both on the lateral and longitudinal axis.

Chapter 4

CTP-Based Left-Turning Planning and Decision-Making Framework

4.1 Introduction

The CTP based left-turning planning and decision-making hierarchical framework is introduced in this chapter. First, an overview of the framework is proposed. Then, a more detailed explanation of the high-level candidate path planning method of the framework is introduced. The critical components for better generalization of the planner to different shapes of intersections are discussed. Finally, the low-level decision-making approach including a problem modeling method and the solving algorithm is explained too.

4.2 A Hierarchical Planning and Decision Making Framework

The complete structure of our proposed CTP based hierarchical left-turn planner and decision-maker is illustrated in Fig. 4.1 and Algorithm 5. The framework includes a high-level candidate path generator that operates offline and a low-level path selector and speed planner that operates online.

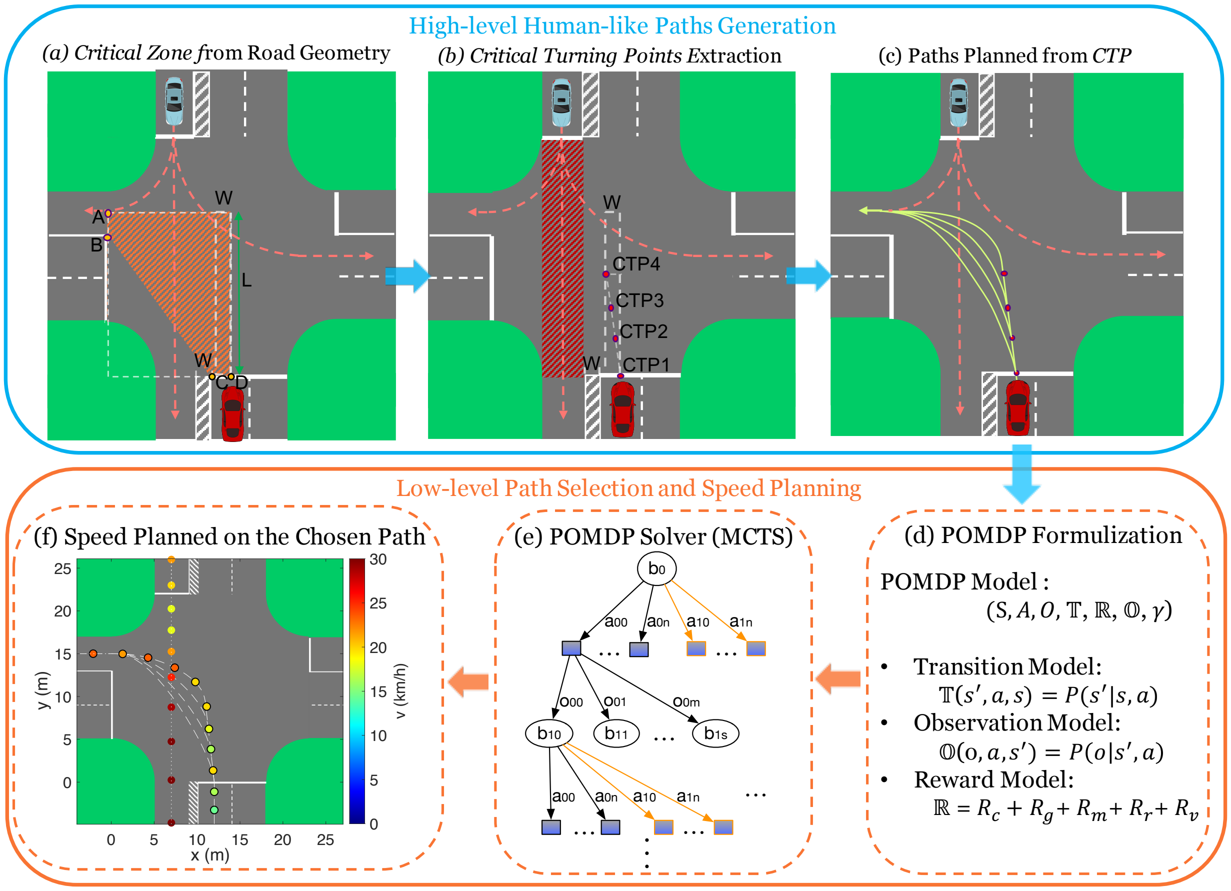

High-level Human-like Paths Generation
(a) Critical Zone from Road Geometry
(b) Critical Turning Points Extraction
(c) Paths Planned from CTP
A
B
W
L
C
D
CTP4
CTP3
CTP2
CTP1
Low-level Path Selection and Speed Planning
(f) Speed Planned on the Chosen Path
(e) POMDP Solver (MCTS)
(d) POMDP Formulization
POMDP Model :
$(S, A, O, \mathbb{T}, \mathbb{R}, \mathbb{O}, \gamma)$
Transition Model:
$\mathbb{T}(s', a, s) = P(s'|s, a)$
Observation Model:
$\mathbb{O}(o, a, s') = P(o|s', a)$
Reward Model:
$\mathbb{R} = R_c + R_g + R_m + R_r + R_v$
x (m)
y (m)
v (km/h)
b_0
a_{00}
a_{0n}
a_{10}
a_{1n}
o_{00}
o_{01}
o_{0m}
b_{10}
b_{11}
b_{1s}

Figure 4.1: CTP-Based left-turning planning and decision-making framework overview

4.2.1 High-level Behavior-Oriented Paths Generation

With the proposed concept of CTP and the phenomenon that the left-turning vehicle in congested traffics usually drives slowly into the intersection before making sharp turns at certain points to finish the left turn, a candidate path generation method for higher left-turn commuting efficiency is proposed. The structure and strategy of the high-level candidate path generator are explained more thoroughly as follows.

Critical Zone Extraction from Road Geometry

The Critical Zone is the yellow shaded area in Fig. 4.1 (a) where all the possible left-turn trajectories take place. When a vehicle drives into the intersection, it first generates a rectangle using the middle point on the edge of the start and goal lane (point A and D in Fig. 4.1) as diagonal vertices, then the corner points of the current and goal lane (B and C in Fig. 4.1) are used to help limit the Critical Zone as a pentagon. The length (L) of the rectangle bounding box of the Critical Zone along with the width of the median strip (W) are sent to the next part for CTP extraction.

CTP Extraction

After receiving the parameters of the intersection, CTPs can be allocated using the CTP extraction model presented in Section 3.3 which includes detailed demonstration of the CTP extraction model and verification with naturalistic driving datasets, which is also shown in Fig. 4.1 (b). These CTPs are crucial waypoints of all the candidate paths.

Candidate Path Generation Using CTP

Candidate paths are generated with the starting point, CTPs and goal point as important waypoints. As shown in Fig. 4.1 (c), straight lines are used to connect the starting point and CTPs, while quintic polynomials [5] are used to generate the left-turn paths from CTPs to the goal point. The detailed quintic polynomial model and car model used will be demonstrated in Section 4.3. After the entire paths have been generated, they are transferred from the Cartesian coordinate system to the Frenet–Serret frame [82] coordinate system. Since using the traveled distance s and speed v along the fixed paths, the ego vehicle's position (x, y) and velocity (v_x, v_y) can be represented in lower dimensions, which increases the search efficiency of the online planner proposed later.

Algorithm 5 CTP based POMDP planning algorithm

```
1: OFFLINE PREPARATION
2: $\textit{Critical Turning Area} \leftarrow \textit{Road Map}$
3: $\textit{CTPs} \leftarrow \textit{Critical Turning Area}$
4: $(p_1, ..., p_n) = \textit{Quintic\_Polynomial}\ (CTP, pt_s, pt_e)$
5: $M_0 \leftarrow \textit{Model\_Modification}\ (p_1, ..., p_n)$
6: $(H, \tau) = \textit{Generate\_Policy}\ (M_0, b_0)$
7: $M_i \leftarrow \textit{POMDP model at time } i$
8: $R \leftarrow \textit{Range tree}$
9: $b = b_0$
10: ____________________________________________
11: ONLINE PLANNING
12: while not Collide or Finish Left Turn do
13:     if $M_t \neq M_{t-1}$ then
14:         $H' = \textit{Get\_Affected\_Parts}\ (M_t, M_{t-1}, H, R, \tau)$
15:         $\textit{Revised\_and\_Update\_Tree}\ (M_t, \tau, b, H')$
16:         while There is still time do
17:             $\textit{Improve\_Policy}\ (M_t, M_{t-1}, H, R, \tau, b)$
18:     $a = \arg\max_{a \in A}(\hat{Q}(b, a) + c\sqrt{\frac{\log(|H_b|)}{|H_{(b,a)}|}})$
19:     $s = \mathbb{T}\,(s', a, s)$
20:     $o = \mathbb{O}\,(o, a, s')$
21:     $b \leftarrow \textit{Update belief}$
22:     $t = t + 1$
```

4.2.2 Candidate Path Selection and Speed planning

The candidate paths generated from the high-level planner are sent to the low-level planner for POMDP problem formulation as shown in Fig. 4.1 (d). Then the appropriate candidate path is chosen and speed is planned on that path in an online manner using MCTS [6] as shown in Fig. 4.1 (e). However, the model should be carefully formulated to ensure the online performance of the solver.

4.3 CTP-based Candidate Paths Generation

With the CTPs extracted from the CTP extraction model, the interpolating curves approach is used to generate paths from each CTP the goal point to achieve smooth and efficient planning. Some popular interpolating spline curves include Bézier curve, cubic polynomial curve and quintic polynomial curves. Bézier curves used several points to shape the desired curve, the curve can be generated considering the road geometric and kinematic of the model. The Bézier curves could be represented as [83]:

$$P(\xi) = \sum_{k=0}^{n} p_k \begin{pmatrix} n \\ k \end{pmatrix} (1-\xi)^{n-k}\xi^k, \tag{4.1}$$

where $p_0, \cdots, p_n$ represented the control points, n is the dimension of the desired curve, $\xi \in [0,1]$ and $\begin{pmatrix} n \\ k \end{pmatrix}(1-\xi)^{n-k}\xi^k$ is the Bernstein basis polynomial with $\begin{pmatrix} n \\ k \end{pmatrix}$ as the coefficient. The Bézier curves are able to provide computational efficient and smooth curves and take the road geometric and kinematic constrains into consideration. However, this method could not control the curves locally and if one control point changes, the shape of the whole curves changes, this shows the drawbacks of generality and flexibility of this method. Cubic polynomial interpolation curve uses 4 constrains to determine the coefficients of the curve, however, the four constrains limits the diversity of the possible paths. Quintic polynomial provide diverse local adjustable curves, and could compute the interpolation curves in a timely manner, therefore, quintic polynomials [5] are chosen for path generation from the CTPs to the goal point.

In consideration of the lateral motion dynamics of the ego vehicle, and to improve the design of the automatic steering of the vehicle, a simplified nonholonomic model is used to provide lateral motion dynamics which is presented as Figure 4.2 and Equation 4.2. Where x and y stands for the rear axle midpoint of the vehicle, $\dot{x}$ and $\dot{y}$ represents the speed on

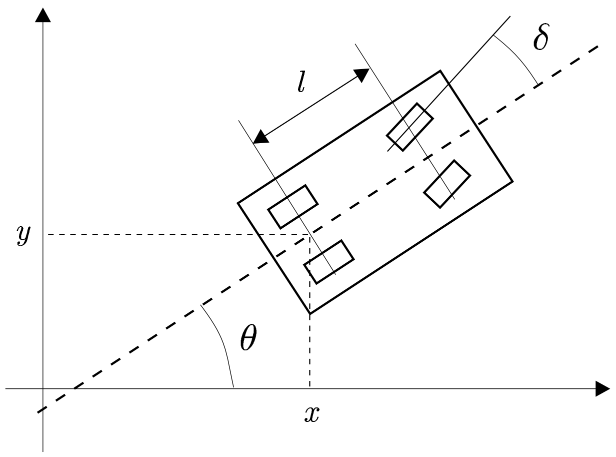


Figure 4.2: Simplified nonholonomic vehicle model for path generation [5]

the x and y axis. θ is the heading angle and v and l is the vehicle's velocity and inter-axle distance respectively, while δ represents the steering angle of the front wheel.

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{l}\tan\delta \end{cases} \tag{4.2}$$

[5] proved that the path generated in the 2-D plane x, y using the car model as shown in Figure 4.2 through a continuous control input $\delta(t)$ if and only if the 2-D path is a second order geometric continuous path, and also provided that the generated path could be assigned as a given parametric curve. Thus, the problem could be taken as an interpolation problem given the starting states and ending states of the ego vehicle. The starting and ending positions of the 2-D plane is defined as $\mathbf{p}_A = \begin{bmatrix} x_A & y_A \end{bmatrix}^T$ and $\mathbf{p}_B = \begin{bmatrix} x_B & y_B \end{bmatrix}^T$ accordingly with corresponding tangent vectors θ_A and θ_B as well as scalar curvature κ_A and κ_B (signs of κ_A and κ_B are decided according the Frenet formulae) as presented as Figure 4.3. The scalar curvature κ could be represented as Equation 4.3 where $\delta(t)$ is the steering angle at time t.

$$\kappa = (1/l)\tan\delta(t), \tag{4.3}$$

The interpolated path can be represented as $(x(u), y(u))$ $u \in [0,1]$, where $u = 0$ and $u = 1$ indicate that the trajectory is at the starting and ending position respectively, and

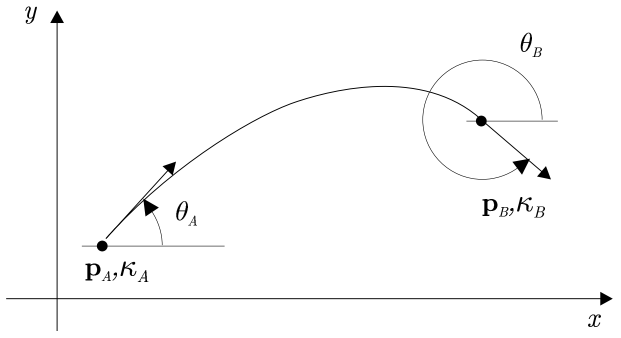


Figure 4.3: Interpolating of the 2-D curve [5]

$x(u)$, $y(u)$ can be expanded as shown in Equation 4.5.

$$\begin{cases} x(u) = x_0 + x_1 u + x_2 u^2 + x_3 u^3 + x_4 u^4 + x_5 u^5 \\ y(u) = y_0 + y_1 u + y_2 u^2 + y_3 u^3 + y_4 u^4 + y_5 u^5, \end{cases} \tag{4.4}$$

where interpolation conditions could be written as:

$$\begin{aligned} &\mathbf{p}(0) = \mathbf{p}_A, \quad \mathbf{p}(1) = \mathbf{p}_B, \\ &\frac{\dot{\mathbf{p}}(0)}{\|\dot{\mathbf{p}}(0)\|} = \begin{bmatrix} \cos\theta_A \\ \sin\theta_A \end{bmatrix}, \quad \frac{\dot{\mathbf{p}}(1)}{\|\dot{\mathbf{p}}(1)\|} = \begin{bmatrix} \cos\theta_B \\ \sin\theta_B \end{bmatrix}, \\ &\kappa(0) = \kappa_A, \quad \kappa(1) = \kappa_B. \end{aligned} \tag{4.5}$$

These conditions only determines four parameters for x and y, thus, this leaves four degrees of freedom which could be decided and modifies through giving extra constrains. In [5], a another set of fine-tuning parameters $\boldsymbol{\eta} := [\eta_1,\ \eta_2,\ \eta_3,\ \eta_4]^T$ to uniquely determine the parametric curve. This allows the curve to also be represented as $\mathbf{p}(u; \boldsymbol{\eta})$. η_1 and η_1 describe the "velocity", for example, if a large η_1 is selected, it will make it hard to form a sharp curve around the starting point. The η_3, η_4 denote the "twist" of the curve, which enhance the curvature of the generated path. The closed-form solution of all the parameters can be calculate with:

$$\begin{cases} x_0 = x_A \\ x_1 = \eta_1 \cos\theta_A \\ x_2 = \frac{1}{2}\left(\eta_3 \cos\theta_A - \eta_1^2 \kappa_A \sin\theta_A\right) \\ x_3 = 10\left(x_B - x_A\right) - \left(6\eta_1 + \frac{3}{2}\eta_3\right)\cos\theta_A - \left(4\eta_2 - \frac{1}{2}\eta_4\right)\cos\theta_B \\ \qquad + \frac{3}{2}\eta_1^2\kappa_A \sin\theta_A - \frac{1}{2}\eta_2^2\kappa_B \sin\theta_B \\ x_4 = -15\left(x_B - x_A\right) + \left(8\eta_1 + \frac{3}{2}\eta_3\right)\cos\theta_A + \left(7\eta_2 - \eta_4\right)\cos\theta_B \\ \qquad - \frac{3}{2}\eta_1^2\kappa_A \sin\theta_A + \eta_2^2\kappa_B \sin\theta_B \\ x_5 = 6\left(x_B - x_A\right) - \left(3\eta_1 + \frac{1}{2}\eta_3\right)\cos\theta_A - \left(3\eta_2 - \frac{1}{2}\eta_4\right)\cos\theta_B \\ \qquad + \frac{1}{2}\eta_1^2\kappa_A \sin\theta_A - \frac{1}{2}\eta_2^2\kappa_B \sin\theta_B, \end{cases} \tag{4.6}$$

$$\begin{cases} y_0 = y_A \\ y_1 = \eta_1 \sin\theta_A \\ y_2 = \frac{1}{2}\left(\eta_3 \sin\theta_A + \eta_1^2 \kappa_A \cos\theta_A\right) \\ y_3 = 10\left(y_B - y_A\right) - \left(6\eta_1 + \frac{3}{2}\eta_3\right)\sin\theta_A - \left(4\eta_2 - \frac{1}{2}\eta_4\right)\sin\theta_B \\ \qquad - \frac{3}{2}\eta_1^2\kappa_A \cos\theta_A + \frac{1}{2}\eta_2^2\kappa_B \cos\theta_B \\ y_4 = -15\left(y_B - y_A\right) + \left(8\eta_1 + \frac{3}{2}\eta_3\right)\sin\theta_A + \left(7\eta_2 - \eta_4\right)\sin\theta_B \\ \qquad + \frac{3}{2}\eta_1^2\kappa_A \cos\theta_A - \eta_2^2\kappa_B \cos\theta_B \\ y_5 = 6\left(y_B - y_A\right) - \left(3\eta_1 + \frac{1}{2}\eta_3\right)\sin\theta_A - \left(3\eta_2 - \frac{1}{2}\eta_4\right)\sin\theta_B \\ \qquad - \frac{1}{2}\eta_1^2\kappa_A \cos\theta_A + \frac{1}{2}\eta_2^2\kappa_B \cos\theta_B. \end{cases} \tag{4.7}$$

The interpolation conditions given in Equation 4.5 decide the starting and ending states of the ego vehicle, and $\boldsymbol{\eta}$ decides the shapes of the candidate paths. To generate candidate paths closer to naturalistic driving trajectories that enable naturalistic driving behaviors at different intersections, proper fine-tuning parameter selection is essential. For regular intersections, there are only limited kinds of shapes due to the limited number of road lanes that forms the intersection (m_1, m_2, n_1, n_2 in Fig. 3.5). Therefore, fine-tune parameters could be pre-defined for each type of intersection. In this paper, recommended fine-tuning parameters for 4 common intersection scenarios are given in Table 4.1. Parameters for other types of intersections (i.e. different number of traffic lanes and irregularly shaped intersections) or more customized paths could be obtained in a similar way.

Table 4.1: Recommended parameters for path generation of four typical intersections

(m_1, m_2, n_1, n_2)	$(\eta_1, \eta_2, \eta_3, \eta_4)$			
	Path1	Path2	Path3	Path4
(1,1,1,1)	(8,13,-2,0)	(12,13,-2,0)	(13,13,-2,0)	(14,13,-2,0)
(2,1,2,1)	(6,13,-2,0)	(9,13,-2,0)	(10,13,-2,0)	(11,13,-2,0)
(1,2,1,2)	(8,13,-2,0)	(9,13,-2,0)	(10,13,-2,0)	(10,13,-2,0)
(2,2,2,2)	(8,13,-2,0)	(10,13,-2,0)	(10,13,-2,0)	(10,13,-2,0)

4.4 POMDP Problem Formulation

POMDP provide a way to make decisions on the best sequences of actions to perform when the agent can not fully observe the states of the environment, in our case, it is the intention of the surrounding vehicles. To properly present the model, a POMDP problem can be summed to a tuple $(S, A, O, \mathbb{T}, \mathbb{R}, \mathbb{O}, \gamma)$ as shown in Fig. 4.1 (d). The component of each element in the model are explained as follows:

- $S = \{s_1, s_2, \ldots\}$ describes all the information of the scenario at a certain moment which is critical information for the decision-making process.
- $A = \{a_1, a_2, \ldots\}$ is the action set that represents all the possible actions.
- $O = \{o_1, o_2, \ldots\}$ is the observation space that includes all the possible observations.
- $\mathbb{T}(s', s, a) = \operatorname{Pr}(s' \mid s, a)$ is the transition model that describe the probability of obtaining of state $s^{'}$ by performing action a from state s.

- $\mathbb{R}(s, a)$ is the reward model that returns the reward from executing action a from state s at each step.
- $\mathbb{O}(a, s, o) = \Pr(o \mid a, s')$ is the observation model that represents the probability of obtaining an observation o from the new state s' which is gained by performing action a from the current state s .
- γ represents the discount factor which $\gamma \in (0, 1)$, this is useful when the horizon of the action sequence is very large.

With the partially known information of the environment, the states are presented as a distribution of states, also known as $beliefs$ (b), which the set contains all the possible beliefs is called the belief space B. The policy is represented by assigning an action a to a belief b. The goal of the agent is to generate a policy to enable the planner to select a sequence of actions that maximizes the accumulative expected reward. Also, a value function $V(b, \pi)$ for every belief within the policy is set to represent the expected cumulative reward of performing policy π from belief b. This value function can be computed as [6]:

$$V(b, \pi) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid b, \pi\right]. \tag{4.8}$$

4.4.1 State Space

The states space of the model is presented by Equation 4.9, which includes both the ego vehicle's states $x_{0,t}$, and the oncoming vehicle's states $x_{i,t} (i = 1, 2 \ldots, n)$ at time t. The explicit expansion of each vehicle's states can be found in Equation 4.10, where $S_{i,t}$ stands for the traveled distance on the pre-defined path, $V_{i,t}$ represents the speed along that path. $P_{0,t}$ indicates the path that is chosen by the ego vehicle, and $P_{i,t} (i = 1, 2 \ldots, n)$ denotes the intention of the oncoming vehicles, which includes going straight, making right or left turns.

$$X_t = (x_{0,t}, x_{1,t} \ldots, x_{n,t}), \tag{4.9}$$

$$x_{i,t} = (S_{i,t}, V_{i,t}, P_{i,t}) \quad i \in 0, 1, 2, \ldots, n. \tag{4.10}$$

It is worth mentioning that the position and velocity of the vehicles are defined on the Frenet–Serret frame [82]. By using pre-defined paths, the dimensions to represent the vehicle's states are reduced, which helps to limit the size of the policy tree, and as a result, increase the search efficiency. The states of oncoming vehicles can be divided into unobservable states such as the driver's intention $P_{i,t}$, and explicit states such as the vehicle's position and velocity, which closely related to the proposed observation model.

4.4.2 Observation Space and Observation Model

The observation model records the observations of all the vehicles at time t and is represented in the observation space (Equation 4.11). The speed and position of each oncoming vehicle in the Cartesian coordinate system are written as $O_{i,t}$ in Equation 4.12, and the ego vehicle's route, speed along the route and traveled distance are shown as $O_{0,t}$ in Equation 4.12.

$$O_t = (O_{0,t}, O_{1,t}, \ldots, O_{n,t}), \tag{4.11}$$

$$\begin{cases} O_{i,t} = (x'_{i,t}, y'_{i,t}, v'_{i,t}) & i \in 1, 2, \ldots, n \\ O_{0,t} = (s'_{0,t}, v'_{0,t}, P_{0,t}) & i = 0. \end{cases} \tag{4.12}$$

The model assumes that there is no sensor noise during perception and the ego vehicle has full knowledge of its own states. However, the states (such as the route) of the surrounding vehicles are not fully observable.

The observation space is a crucial component for deciding the size of the tree which is critical to make efficient planning. Therefore, an accuracy variable κ_a for the observation model is introduced in Equation 4.13. The states are multiplied by the accuracy variable first, then rounded to the closest integer and finally divided by κ_a. By tuning κ_a, the width of the search tree could be adjusted.

$$\mathcal{X}'_{i,t} = \lfloor \frac{\mathcal{X}_{i,t}}{\kappa_a} \rceil \cdot \kappa_a. \tag{4.13}$$

4.4.3 Action Space

The action space includes an acceleration variable (a_v in Equation 4.14) and a left-turning "instruction" Boolean variable a_t. The acceleration variable (a_v ranged from -4 m/s to 4 m/s with a interval of 1 m/s, it only controls the ego vehicle's speed along the predefined path. The left-turning"instruction" variable a_t on the other hand, conveys sharp turn instructions. When this parameter is set to "1", the ego vehicle will shift from the creeping forward phase to the sharp left-turning phase at the next CTP ahead. With this Boolean variable, the ego vehicle would be able to plan in a two-dimension but smaller search space.

$$a = (a_v, a_l). \tag{4.14}$$

4.4.4 Reward

The model's reward includes rewards for reaching the goal, moving as the referenced speed, selecting the proper path and also a penalty for crashing with the oncoming vehicle and going backward, which is written as

$$\mathbb{R} = R_v + R_c + R_g + R_r + R_m. \tag{4.15}$$

$R_v = -300\left(V - v_{ref}\right)^2$ is set for the vehicle to follow the desired speed, the speed difference is squared for fast and closed tracking of the ego vehicle's speed to the reference. The crash penalty is formulate as

$$R_c = \begin{cases} -5 \times 10^6, & \text{if dist } < \text{dist}_{\text{safe}} \\ 0, & \text{otherwise.} \end{cases} \tag{4.16}$$

similarly, the reward for reaching the goal is set as

$$R_g = \begin{cases} 5 \times 10^4, & \text{if } S > S_{full} \\ 0, & \text{otherwise.} \end{cases} \tag{4.17}$$

where S stands for the traveled distance on the chosen path, S_{full} is the full length of the chosen left-turn candidate path. The penalty for the ego vehicle to move backward is set as

$$R_r = \begin{cases} -1 \times 10^5, & \text{if } v < 0 \\ 0, & \text{otherwise.} \end{cases} \tag{4.18}$$

to avoid dangerous revers movements.

Finally, the key reward that enables the vehicle to select the appropriate path is set as $R_m = 10y - 50x^2$, which encourages the vehicle to move closer to the goal on both x and y-axis. Thus, the ego vehicle would be able to select the appropriate path for better commute efficiency in different scenarios.

The specific values for the parameters of the rewards are selected in a priority sequence. Safety is the highest demand for the framework. Therefore, R_c is selected first and is set to a large value, and R_r is selected afterward with a relevantly high value. It is also critical for the vehicle to behave as commanded and finally reach the goal. Thus, R_v and R_g is then selected with a smaller value. Finally, for the framework to help the ego vehicle to select the proper candidate path, R_m is selected as a small value, which has a small influence on the whole reward function, but enables the path selection function.

4.4.5 Transition Model

With the assumption that all the vehicles are moving on the pre-defined paths, the transition model could also be defined on the Frenet–Serret frame. Before the ego vehicle enters the intersection, the turning state $P_{0,t}$ is set as -1. When the ego vehicle is driving close to a CTP and no path has been selected, the next state of the ego vehicle is generated using the following equation.

$$\begin{bmatrix} S_{0,t+1} \\ V_{0,t+1} \\ P_{0,t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{0,t} \\ V_{0,t} \\ P_{0,t} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \lceil \frac{S_{0,t}+L}{L} \rceil \end{bmatrix} \begin{bmatrix} a_{\mathrm{v}} \\ a_{\mathrm{l}} \end{bmatrix}. \tag{4.19}$$

Where L represents the distance between neighboring CTPs (e.g. distance between CTP1 and CTP2 in Fig. 4.1). The vehicle's position and speed on the fixed path are controlled by the acceleration action variable a_v and the candidate path is selected by the sharp-turn instruction variable (a_l). When a sharp turn instruction ($a_l = 1$) is given, the number of the closest CTP ahead (which is also the number of the selected path) is recorded using the transition function (Equation 4.19). Once $P_{0,t}$ is updated to a non-negative value, the left-turning "instruction" variable a_l is invalidated and the vehicle will take the corresponding candidate path. Also, if the ego vehicle passed the last CTP, then $P_{0,t}$ is set to the number of the last CTP (4 in the case of Fig 4.1), and a_l is invalidated as well.

As for the oncoming vehicles, their state-transition strategy is similar to the ego vehicle. Only the route ($P_{i,t}$) of each vehicle is pre-defined and fixed, the acceleration action a_s of each vehicle follows a normal distribution of $\mathcal{N}(0,1)$ ranging from $[-2\ m/s^2, 2\ m/s^2]$, with a interval of $1\ m/s^2$. The transfer function is given as

$$\begin{bmatrix} S_{i,t+1} \\ V_{i,t+1} \\ P_{i,t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{i,t} \\ V_{i,t} \\ P_{i,t} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \\ 0 \end{bmatrix} a_{\mathrm{s}}. \tag{4.20}$$

With the 2 given transfer functions of the vehicles, if there is only 1 oncoming vehicle, the distribution of uncertainties of the transition model can be represented as a normal distribution as

$$\mathbb{T} = (s'|s,a) = \mathcal{N}(0,1), \tag{4.21}$$

where the uncertainty comes from the unknown actions of the surrounding vehicles.

4.5 Solving the Decision-Making Problem

Due to the complexity and rapidly changing nature of intersections, it would be appropriate to solve left-turning decision-making problems online. Therefore, according to the literature review made in Section 2.6.3, the Monte Carlo tree search method is implemented in our framework for left-turning navigation because of its aforementioned 3 major advantages. Among all the MCTS methods, Adaptive Belief Tree (ABT) [6] is implemented. This solver uses MCTS to generate a policy tree, then the action with the highest reward is executed and belief is updated using a particle filter. After that, instead of discarding the entire policy tree as the classical POMCP online solver [78], ABT only identifies and trims the parts of the policy tree that are influenced by the change of the model (e.g. change of action space, transition model, observation model, etc.), then revises the influence part of the policy tree and improves it if there is still time left in that time cycle. Since most part of the policy tree is saved after each time step, the search efficiency is boosted. Thus, the real-time performance and safety of the left-turn planner is significantly improved. A detailed explanation of the ABT-based POMDP solver is presented in the rest of this section.

4.5.1 Representation of the Policy

The policy is represented by sequences of pairs of action and beliefs, and also the relation between the states, beliefs and their reachability information. This representation could increase the speed of finding the part of the policy tree that had been influenced by the change of the POMDP model. The change of the POMDP model is defined as changing any element of the POMDP model, which in this case could be the change of the updated belief and the transition function.

ABT represents the policy through a belief tree $\mathcal{T}$. The belief tree is formed by a set of sampled episodes H, an episode $h \in H$ is represented by a sequences of quadruples (s, a, o, r) as shown in the table in Figure 4.4, where $s \in S$ represents the state, $a \in A$ represents the action $o \in O$ and $r \in R$ represent the reward. The beliefs in the tree are represented as belief nodes which is represented as nodes in Figure 4.4 (e.g. b_0 represents the root of the belief tree $\mathcal{T}$), each node is formed by a set of particles (which is represented as dots in the nodes of Figure 4.4), where every particle is associated with a quadruple. The edge that connects the nodes b and $b^{'}$ in the tree is formed by a action-observation pair (e.g. $a_0 - o_0$ pair in Figure 4.4), which means that a particle in the belief b performs an action a and reached to belief $b^{'}$ (i.e. $b' = \tau(b, a, o)$) which perceives an observation o. These

Algorithm 6 Episode generation

```
1: $b = b_{\text{start}}$
2: Let $l$ be the depth level of node $b$ in $\mathcal{T}$
3:   Let $s$ be a state sampled from $b$ .
     The sampled state $s$ is essentially the state at the $l^{th}$ quadruple of an episode $h' \in H$
4: Initialize $h$ with the first $l$ elements of $h'$
5: Initialize doneMode as UCB.
6: Let $A$ be the action space of POMDP model $P$
7: **while** $\gamma^l > \varepsilon$ AND doneMode == UCB **do**
8:     Let $A'$ be the set of actions that labelled the edges from $b$ in $\mathcal{T}$
9:     **if** $|A'| == |A|$ **then**
10:        $a = \text{UCB} - \text{ACTION} - \text{SELECTION}(\mathcal{T}, b)$
11:    **else**
12:        $a$ = an action sampled trom $A \backslash A'$ unitormly at random.
13:         doneMode = Rollout.
14:    $(o, r, s') =$ GenerativeModel $(P, s, a)$
15:    Insert $(s, a, o, r)$ to $h$
16:    Add $h_l.s$ to the set of particles that represent belief node $b$ and associate $b$ with $h_l$
17:    $\boldsymbol{s = s'}$
18:    $b =$ child node of $b$ via an edge labelled $a$ -o. If no such child exist, create the child.
19:    $l = l + 1$
20: **if** doneMode == Rollout **then**
21:    Let $p$ be a number sampled uniformly at random from $[0, 1]$
22:    **if** $p < p_{\text{policy}}$ **then**
23:        $r =$ ROLLOUT-POLICY $(\mathcal{T}, s, b)$
24:        rolloutUsed = policy.
25:    **else**
26:        $r =$ ROLLOUT-DET $(P, s)$
27:        rolloutUsed = deterministic.
28: **else**
29:    $r =$ GenerativeModel $(P, s)$
30: Insert $(s, -, -, r)$ to $h$
31: Add $h_l.s$ to the set of particles that represent belief node $b$ and associate $b$ with $h_l$
32: valueImprovement = UPDATE-VALUES( $\mathcal{T}, h$)
33: $p_{\text{policy}}$ = UPDATE-ROLLOUT-PROB(rolloutUsed, valueImprovement).
34:  Insert $h$ to $H$
```

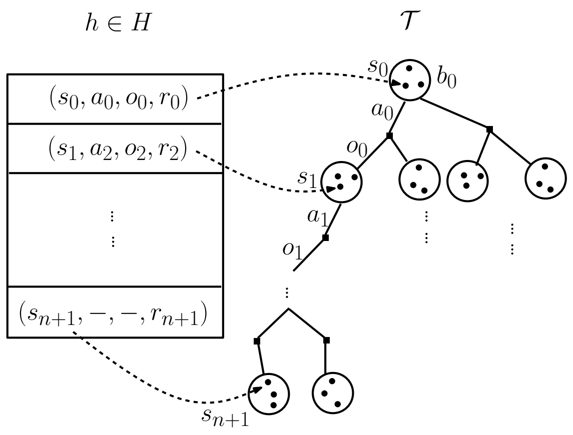


Figure 4.4: Illustration of the relation between the episode and the path in the belief tree [6]

edges that connect the initial belief to the final belief are defined as paths. Assume ϕ is a path of the belief tree, then ϕ can be represented by $\phi = (b_0, a_0, o_0, r_0, \ldots, a_n, o_n, b_{n+1}))$. This path is associated with the all the corresponding episodes $H_\phi \subseteq H$ that contains $((*, a_0, o_0, *), \ldots, (*, a_n, o_n, *), (*, -, -, *))$. As shown in Figure 4.4, an episode can be matched to a single path in the belief tree. However, that path can be matched to many episodes of the tree since there are three particles in the belief nodes that shares the same path.

The policy π is embedded in the policy tree, which is represented by:

$$\pi(b) = \underset{a \in A(E,b)}{\arg\max} \hat{Q}(b, a), \tag{4.22}$$

with

$$V(b, \pi) = \max_{a \in A(E,b)} \hat{Q}(b, a), \tag{4.23}$$

where $b \in B$ is the belief, E is the set of all the edges of the tree and $A(E, b)$ represents the possible actions that generate b' from b. $\hat{Q}(b, a)$ is an estimation of the Q-value, which is the reward received after executing action a and keep on performing optimally until the horizon has been reached, which are also presented as:

$$Q(b, a) = R(b, a) + \gamma \sum_{o \in O} \tau(b, a, o) V^*(\tau(b, a, o)). \tag{4.24}$$

ABT estimates the Q-value through:

$$\hat{Q}(b,a) = \frac{1}{\left|H_{(b,a)}\right|} \sum_{h \in H_{(b,a)}} \sum_{i=l}^{|h|} \gamma^{i-l} R\left(h_i \cdot s, h_i \cdot a\right), \tag{4.25}$$

where $H_{(b,a)} \subseteq H$ represents all the of episodes that include the corresponding belief-action, l is the tree depth and $V(h,l) = \sum_{i=l}^{|h|} \gamma^{i-l} R\left(h_i \cdot s, h_i \cdot a\right)$ is the value of an episode h starting from the element of l^{th} depth, γ stands for the discount factor and R denotes the reward function. It has been proved in [6] that the estimation of the Q-value which is the $\hat{Q}(b,a)$ as presented in Equation 4.25 can properly approximate the true Q-value.

4.5.2 Policy Generation and Update

The policy is represented by the belief tree which is formed by episodes. To properly sample the episodes, thus grown the policy tree to an ideal size, this algorithm uses two major search methods, which is the upper confidence bound, version 1 (UCB1) search [84] and the *rollout* search [78]. The thorough representation of sampling an episode is presented in Algorithm 6 [6].

UCB1 Algorithm

The upper confidence bound (UCB) strategy is widely used in reinforcement learning problems, it formed the action selection problem from the belief nodes into a multi-arm bandit problem. In this POMDP solver, UCB1 algorithm is used to balance between the exploitation (e.g. select the action that showed the best performance so far) and exploration (e.g. select the action that has not been selected yet, but have the potential to show good performances) in the tree search. UCB1 is selected also because it is one of the best multi-arm bandit algorithms that solved the problem in which the reward for executing an action follows a stationary distribution and is unknown before searching. The action selection strategy of UCB1 can be written as:

$$a = \arg\max_{a \in A}(\hat{Q}(b,a) + c\sqrt{\frac{\log\left(|H_b|\right)}{\left|H_{(b,a)}\right|}}), \tag{4.26}$$

where H_b is all the episodes that include the belief b, and similarly $H_{(a,b)}$ represents all the episodes that execute action a from belief b. $|*|$ indicates the size of $*$ (e.g. could represent the visiting counts of a belief node). c is a critical factor that balances between exploration and exploitation ($c = 0$ means that the algorithm is searched greedily inside the tree). This algorithm has been widely used in many POMDP solvers [6,78], and enables the POMDP solvers to converge to optimal policies.

Rollout Search

The rollout search is a simple Monte Carlo method that helps to evaluate the state s. The actions are first selected according to a certain probability distribution, until a terminal condition has been reached or the planning horizon has bee reached. Then a sequence of reward is returned, and the average return of all the reward returned is used to estimate the value of the state s as [79]:

$$\hat{R} = \sum_{d=0}^{d_n} \gamma^d r^d, \tag{4.27}$$

where $r^{d=0}, r^{d=1}, \ldots, r^{d_n}$ is the sequence of the reward. In the ABT framework, two types of rollout policy are used, which are the ROLLOUT-DET and ROLLOUT-POLICY, which is presented in Algorithm 6 line 21-27. ROLLOUT-DET (Algorithm 6 line 26) is the rollout method that assumes that the problem is a deterministic problem, and thus calculate the accumulative reward of the robot starts from state s execute action a and then operates optimally based on the assumption. ROLLOUT-POLICY (Algorithm 6 line 23) applied heuristic based on existing policy. The algorithm first uses the *generative model* to sample a simulation and obtains an observation, then uses a particle filter to compute the resulting belief $b' = \tau(b, a, o)$. With the belief, the algorithm finds the belief node $\widehat{b'}$ in the existing tree that is closest to b', this distance is measured using the expected state-space distance assuming that the beliefs do not correspond to each other. If the distance between the beliefs is small enough (a preset threshold), then the beliefs are equal, thus the algorithm simulates the total discounted reward using the policy that is already existed inside the policy tree from $\widehat{b'}$ until a leaf node is reached. This discounted reward is the heuristic estimation of the Q-value. Otherwise, if the distance between the beliefs is large, the algorithm still uses the ROLLOUT-DET to estimate the Q-value.

Generating an Initial Policy

The major steps of generating an initial policy are sampling episodes. To sample an episode, the algorithm first sample a particle (state $s \in S$) from the initial belief b_0. Then, given the initial particle, the algorithm has to select an action to execute, the action selection strategy is: if all the actions corresponded to that node has been used to expand the belief node at least once, the algorithm will use the UCB1 algorithm, which is represented in Algorithm 6 line 9-10. If the conditions of executing UCB1 are not satisfied, then the rollout strategy is applied. The algorithm selects an action randomly from all the actions that have not been used to expand the belief node (Algorithm 6 line 12). The rollout strategy will give a relatively good estimation of the Q-value (Algorithm 6 line 21-27.), which is very useful for the UCB1 method to converge faster to the optimal action when the UCB1 operation conditions is validated. The search continues until the tree is grown to an ideal size or time has run out. If there is still time, the algorithm improves policy (Algorithm 5 line 17).

Policy Execution and Belief Tree Update

After the belief tree is grown, the agent executes the action with the highest Q-value from the initial belief node b_0 ($a = \pi(b_0)$). Then the agent receives an observation from the observation function. Using this observation the belief is updated using a particle filter. After the operation of the agent, the POMDP model is changed, including the initial belief, the policy tree, etc. The algorithm finds the subsets of the policy that are influenced by the change of the model (the set of states influenced by the changes), once the influenced states have been identified, the relevant episodes that are influenced can be located. The algorithm then revises the affected elements of all the located episodes through the following procedure. First, the algorithm finds the most shallow tree depth of all the affected in the episodes. For each of these three episodes, the algorithm deletes the part of the episode that is deeper than the obtained tree depth, and then re-samples the deleted part of the episode. Finally, the algorithm replaces the deleted quadruples of the episodes with the re-sampled sequence of the next states, observations and rewards, etc. This revision process leads to a rebuilt of the policy tree, thus the Q-values of the revised belief in the policy tree are updated, which enables the update of the policy embedded in the policy tree.

4.6 Summary

The CTP-based left-turning planning and decision-making framework are proposed, the high-level candidate paths generation model uses quintic polynomials to generate paths from the CTPs to the goal point. Different modifications are made for the path planner to adapt to various shapes of intersections. Then the framework modifies the intersection decision-making problem into a POMDP model which includes state space, action space, observation space, etc. The POMDP model is solved through an POMDP planner, the planner generates the policy tree with policy embedded inside. With the generation of the original policy tree and the fast policy tree update, optimal actions can be chosen in a timely manner.

Chapter 5

Simulation and Results

5.1 Introduction

The performance of the proposed left-turn planner is evaluated in different aspects. First, the framework is tested in typical intersection scenarios that a collision may occur, and the ability to consider the unknown intentions of the oncoming vehicle is testes too. The ability of the framework to adapt to various shapes of intersection is also verified, which stands as an important part of the generalization of the intersection planner and decision-maker. Finally, the appropriate three sizes which enable a thorough search of the intersection and a further evaluation horizon, and meanwhile also provide a good real-time performance of the decision-maker.

In the test scenarios, left-turns take place at regularized intersections where the width of the median strip is set as 1 meter. It is also assumed that only 1 oncoming vehicle is entering the unsignalized intersection, and the vehicle has an intention (route to follow) that is unknown to the ego vehicle, the surrounding vehicle also has an uncertain acceleration deceleration which follows a normal distribution. The ego vehicle, on the other hand, is entering the intersection at a relatively high speed or low speed.

5.2 Simulation under Typical Scenarios

3 typical scenarios that collisions may occur are tested, which is the oncoming vehicle turns right at low speed, goes straight with low and high speed. The results are shown in Fig. 5.1.

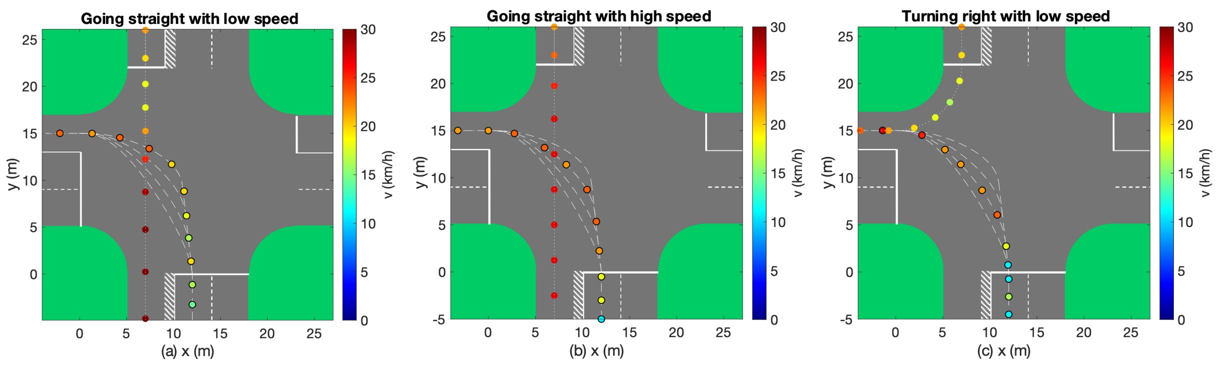


Figure 5.1: Simulation results of three typical scenarios with different oncoming vehicle's intention

Colored dots with black contours indicate the ego vehicle's trajectory, and the rest of the dots represent the oncoming vehicle's trajectory. The color of each dot represents the speed of the vehicle at that position. The dashed lines are the possible candidate paths of the ego vehicle, and the dotted lines are the pre-defined path of the oncoming vehicle.

If the oncoming vehicle is going straight, the left-turning vehicle might collide with the oncoming vehicle at the potential collision area (red shaded area in Fig. 4.1 (b)), thus when the intention of the oncoming vehicle is uncertain, there is a possibility that the Critical Zone (orange shaded area in Fig. 4.1 (a)) is blocked. As presented in Fig. 5.1, the ego vehicle is able to consider this risk and drives slowly into the intersection and waits for the intention to be clear. After the intention of the oncoming vehicle is certain, more "confident" actions were made, that the ego vehicle either decelerates if the Critical Zone is blocked or take a sharp turn when the Critical Zone is clear. What's more, when the oncoming vehicle with an intention of going straight blocks the Critical Zone, the ego vehicle takes sharp turns around different CTPs according to different timing for the oncoming vehicle to drive pass the collision area (Fig. 5.1 (a), (b)). When the oncoming vehicle is taking a right turn, the Critical Zone is cleared at an earlier stage, the ego vehicle then correspondingly makes the sharp turn earlier (Fig. 5.1 (c)). The results show that the planner would be able to select and plan different candidate paths and plan speed along the path in a sensible manner considering the uncertain intention and actions of the oncoming vehicle.

The safety distance ($dist_{safe}$ in Equation 4.16) is set to 2.4 meters to avoid collision when making left turns. In all test cases, this distance is strictly ensured. The results

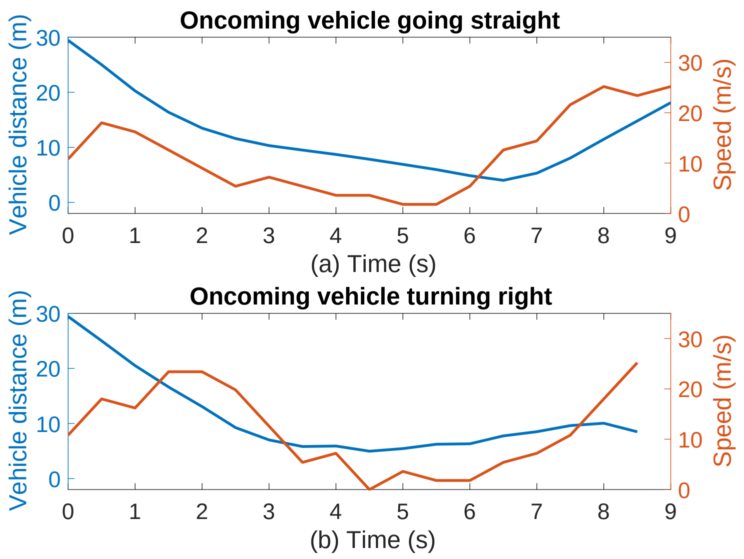


Figure 5.2: Vehicle distance and speed profile with respect to time in two critical scenarios

of 2 specific test cases in which a crash may occur are shown in Fig. 5.2. The blue lines indicate the distance between the ego vehicle and the oncoming vehicle, and the orange lines represent the changing speed of the ego vehicle. In Fig. 5.2 (a) and (b), the distance between the 2 vehicles is always greater than 2.4 meters and the ego vehicle's speed starts to decrease before the distance is at its minimum. This shows that the planner is able to keep the pre-set safety distance and avoid collisions.

The main feature of the POMDP problem is that it can deal with the problem with uncertainties, the feature has been tested in our simulation. Fig. 5.3 demonstrates that the planner is able to make decisions considering the oncoming vehicle's uncertain intentions, the blue lines are the beliefs of the intentions and the orange lines stand for the action of the ego vehicle. As shown in Fig. 5.3 (a), (b) and (c), the ego vehicle performs similar actions from 0s - 1.5s. After that, if the oncoming vehicle's left-turning intention becomes certain, which means that the Critical Zone is clear, the ego vehicle maintains its current speed (Fig. 5.3 (a)). If the intention of going straight or turning right is still uncertain, the ego vehicle decelerates consider the probability that the Critical Zone might be blocked. At last, if the oncoming vehicle's intention of going straight becomes certain, the Critical Zone is blocked, then the ego vehicle continues to decelerate (Fig. 5.3 (b)). Moreover, if the oncoming vehicle is confirmed to turn right at last, there is more space to make left turns, then acceleration is performed (Fig. 5.3 (c)). These results show that our proposed method enables the ego vehicle to make less conservative actions considering the uncertainties of

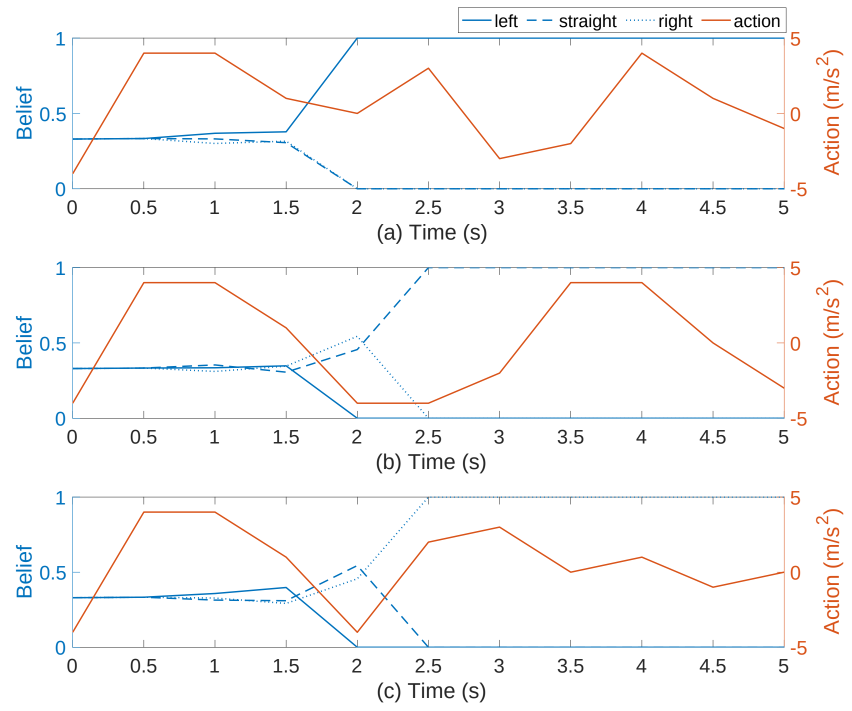


Figure 5.3: Change of actions due to different beliefs

the oncoming vehicle.

5.3 Adaption to Various Road Geometries

The performance of the CTP based hierarchical planner is evaluated in different road geometric environments. Intersections with different shapes are tested with planners with and without CTPs. The different shapes of intersections are mainly differs from the number of road lanes, as shown in the first column of Table 5.1, these variables (m_1, m_2, n_1, n_2) represent the road lanes as shown in Figure 3.5. Each scenario is tested 10 times with a time step of 0.5 seconds, the average time steps of each scenario are shown in Table 5.1, where the time to finish each left turn is calculated by multiplying the number of time steps with 0.5. From the table, it can be seen that, in all the test cases, the commuting efficiency of the planner with CTPs is no less than the planner without CTPs. Moreover,

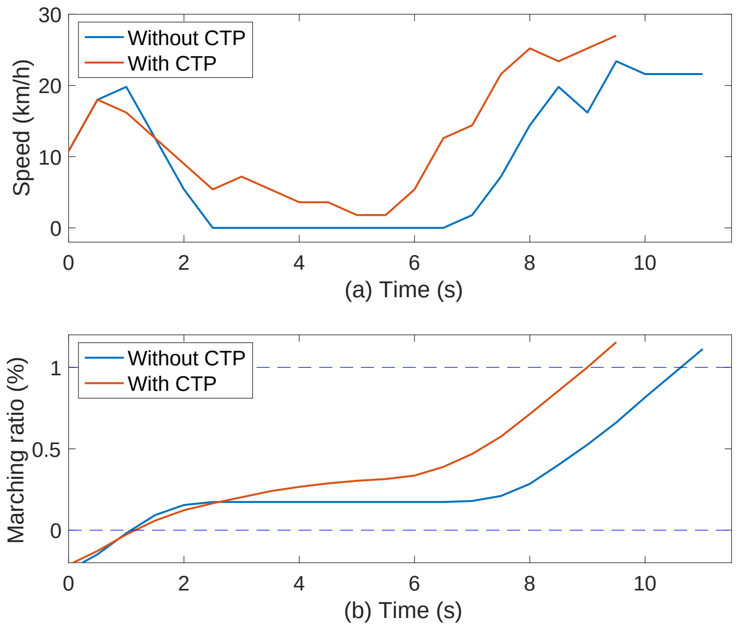


Figure 5.4: Speed profile and marching ratio with respect to time

commuting efficiency increases conspicuously when the oncoming vehicle is going straight. These results show that the proposed planner can be applied in more general road geometric environments and produce finer performances.

The reason that commuting efficiency is improved is illustrated in Fig. 5.4. The figure shows the change of the marching ratio and speed at a specific scenario ($m_1 = 1, m_2 = 2, n_1 = 1, n_2 = 2$ with the oncoming vehicle going straight) with and without CTPs. Marching ratio is defined as the distance traveled by the ego vehicle divided by the length of the chosen path, because the lengths of the paths are different when using different planners, simply compare the traveled distance would be confusing.

As shown in Fig. 5.4 (a), without the uses of CTPs, the path of the ego vehicle is fixed, thus, when the Critical Zone is blocked by the oncoming vehicle, the ego vehicle stops and waits for the critical zone to clear until any acceleration can be executed. However, with the uses of CTPs, if one of the paths is blocked, the vehicle has the option to keep moving forward and take the sharp turn at the next CTP. This is also demonstrated in Fig. 5.4 (b), when the blue line becomes flat, the red line still rises. In other words, the vehicle will manage to get closer to the goal point on the y-axis when moving along the x-axis is dangerous. As a result, when the critical zone is cleared, the remaining distance of travel is decreased with the uses of CTPs, also, the average speed is increased too.

Table 5.1: Performance evaluated in different intersections

(m_1, m_2, n_1, n_2)	Intention (Speed)	CTP	Average Steps	Average Time (s)
(1, 2, 1, 2)	straight (low)	no	24	12
		yes	21.1	10.55
(1, 2, 1, 2)	left (fast)	no	11	5.5
		yes	11	5.5
(1, 1, 1, 1)	right (fast)	no	12.9	6.45
		yes	12.2	6.1
(1, 1, 1, 1)	left (fast)	no	10.7	5.35
		yes	10.5	5.25
(2, 2, 2, 2)	straight (fast)	no	12.7	6.35
		yes	11	5.5

5.4 Optimization between Tree Size and Performance Stability

The performance of the POMDP based planner is closely related to the size of the search tree. If the tree size is large (i.e. having great tree depth or width), the planner will be more likely to find the solution closer to the optimal policy, however, generating a huge policy tree would cost a great amount of time, and that effect real-time performance. In intersection decision making and planning problems, real-time performance is as important as finding the optimal policy. Therefore, finding the proper width and depth of the tree will be important to generate proper actions and ensures real-time performance.

The test scenario is formulated as a regular intersection with numbers of traffic lanes set as $m_1 = 1$, $m_2 = 2$, $n_1 = 1$, $n_2 = 2$ (parameters in Fig. 3.5), and a oncoming vehicle that enters the intersection with the intention of going straight. 4 CTPs are used in this scenario. In each case, the planner is tested for ten times with a time step of half a second, then the average number of steps and time is recorded in Tabel 5.2, taken fewer steps means shorter time needed to finish the left turn, which brings higher commuting efficiency.

Only the first test case is not operated in real-time, it takes 10 times the time needed for real-time operation to generate the policy, therefore the policy is closer to the optimal policy among all test cases. Thus, the policy generated is defined as the benchmark policy, and the path taken is defined as the ideal path. As shown in Tabel 5.2, with a tree depth

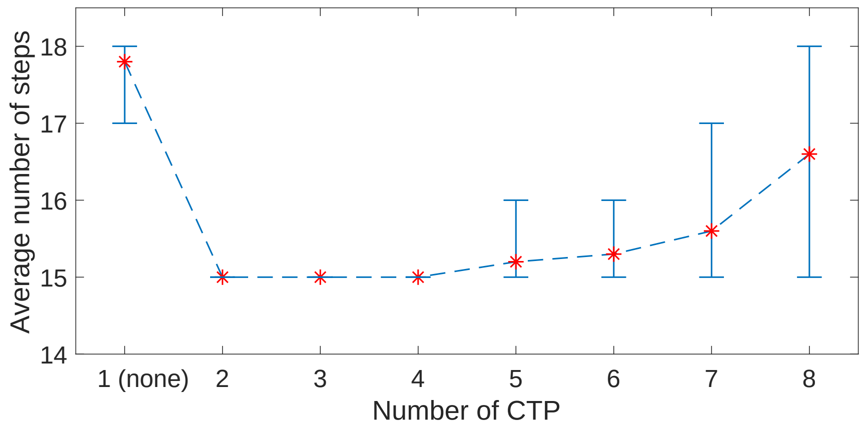


Figure 5.5: Commuting efficiency with different number of CTPs

of 5, an observation accuracy ratio of 1 and the deepest tree depth among all the other test cases, the planner takes the same path every time and the average number of steps are 12.9 (6.45 seconds of operation time and 64.5 seconds of calculation time).

The rest of the test cases all operate in real-time. The second to the fourth test cases (2-4 row) uses different tree depths, and as shown in the second test case, the optimal path has not been chosen, and the average number of steps is 15. This shows that a tree depth of 3 is insufficient for the planner to "foresee" the potential collisions in the future and make good decisions. The third and fourth test cases take deeper tree depth and have generated a policy that is much closer to the benchmark policy. The last 3 test case takes a higher resolution of the observation space, thus, the width of the tree becomes larger. As shown in the last 3 rows of Tabel 5.2, the average steps taken and the percentage of selecting the ideal path both decreases conspicuously. Due to these results, an observation accuracy as 1 and a tree depth of 4 or 5 would be an ideal tree size for good performance in this test scenario. For different shapes of intersections, tree size could be tuned to meet different personal requirements (e.g. larger intersections may require a deeper tree depth).

The complexity of the model is also a major influence on the performance of the planner. In this case, complexity is influenced by the numbers of CTPs, the more CTPs that are used, the more candidate paths would be generated to be taken into account to find an appropriate behavior. higher complexity means a thorough exploration of the intersection, however, that also expands the observation space and state-space correspondingly, which means more particles would be needed to generate a proper policy tree. The test environment is formulated as a multi-lane ($m_1 = 1$, $m_2 = 2$, $n_1 = 1$, $n_2 = 2$) regular intersection with an oncoming vehicle going straight, different number of CTPs are tested for ten times

Table 5.2: Different tree sizes with according performances

Observation Accuracy	Tree Depth	Calculation Time (s)/Step	Ideal Path	Average Steps	Average Time (s)
1	5	5	100%	12.9	6.45
1	3	0.5	0%	15	7.5
1	4	0.5	100%	12.9	6.45
1	5	0.5	90%	13.1	6.55
1.5	4	0.5	50%	13.3	6.65
1.5	5	0.5	50%	13.4	6.7
2	4	0.5	0%	15	7.5

each. Then the results are presented as an error bar as shown in Fig. 5.5. The red dot represents the average number of steps and the upper and lower bound of the error bar represents the range of the number of time steps for each case.

As shown in Fig. 5.5, without the use of CTPs, the steps needed to drive through the intersection is around 18 steps (9 seconds), when there are 2-4 CTPs used, the steps needed is reduced to 15 (7.5 seconds). However, when more than 4 CTPs are used, the results become unstable and the average number of steps grows accordingly. This means that when more CTPs are used, a more complex problem is to be solved in a limited amount of time, the tree is spanned to an inadequate size and may not always be able to find a good solution. So in this case, 4 CTPs would be appropriate.

5.5 Summary

The simulation results show that our proposed CTP-based intersection planning and decision making framework can be applied to various shapes of intersections with median strips. The use of the CTPs enables the ego vehicle to increase the commuting efficiency in left turns. With the application of the POMDP modeling and solving, the ego vehicle is able to consider the uncertainties of the surrounding vehicles, and make less conservative yet safe actions. The sufficiency of tuning the tree size and model complexity is also verified and our proposed framework had shown the ability to be adjusted and fit to different left-turning scenarios.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The objective of this thesis to propose a safe and efficient intersection planning and decision-making framework for autonomous vehicles. To achieve this goal, a generalized CTP-based hierarchical left-turning planner and decision-maker are proposed. The model uses the novel generalized CTP concept by observing real-life left-turning driving behaviors. Using this CTP concept, a CTP model is developed which is applicable at intersections with multiple traffic lanes and median strips. This model is proposed and verified through observing naturalistic driving-behavior datasets at intersections. CTPs extracted from the model is also a key component of the left-turning planning and decision-making framework. The high-level candidate path generator which could adapt to different intersections uses CTP as a critical component to generating candidate paths. With the generated candidate left-turning paths, the left-turn problem is formulated as a partially observable Markov decision process (POMDP) problem. Then the problem is solved using a MCTS-based POMDP solver which generates a policy tree and updates the tree in every time step. During this process, the POMDP solver gives an approximate solution to the problem in real-time.

The proposed framework is first tested in several critical scenarios that collisions may occur. The simulation results show that the framework is able to keep the preset safety distance in all the cases, while considering the intentions of the surrounding vehicles to avoid overly conservative actions. It is also shown that the proposed left-turning planning and decision-making framework that uses CTPs has shown better performance than the framework that is not using CTPs in some critical scenarios, and performs equally well in rest of the cases. The generality of the framework has also been tested. It is being tested in various different shapes of intersections that are common in daily lives. The framework has shown that it could adapt to all the cases and perform efficient and safe performances. The tree size of the generated policy tree in the POMDP solver is highly related with the performance of the framework, therefore, the appropriate tree size to provide the balance between obtaining relative good policy and stability is tested and discussed. This also shows the ability that the framework could provide personalized modification to be able to meet the demand for real-time performance or more optimal solutions.

The uses of CTP in the framework which is extracted from real-life driving behaviors enable more human-like driving behaviors. This allows the surrounding vehicles to be easier to understand the intentions of the ego vehicle which improves safety for both the ego vehicle and other agents in the scenario. In all, using this proposed approach, the autonomous vehicle is able to make less conservative decisions while considers the risks of collision in real-time, which improves the commute-efficiency, and urban driving of autonomous vehicles will be able to integrate into surrounding human-driven vehicles in a friendly manner.

6.2 Future Work

The framework only output high-level sequences of decisions, therefore, for better implantation, a low-level controller could be developed in the future since in real-life it would be uncomfortable for passengers to encounter jerks on the ride. Also, from the perspective of the vehicle model, though the minimum turning circle is considered and a simplified non-holonomic vehicle model is used. For easier path tracking and action execution in real-life, a more complex vehicle model could be used for candidate paths generation and action selection. Finally, though the framework had shown the potential of having the previously mentioned advantages both in theory and in simulation, real-world testing and evaluation would still be appropriate for works needed to be done in the future.

References

[1] Douglas Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, Technical report, National ICT Australia, 2003.

[2] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of artificial intelligence research*, 13:33–94, 2000.

[3] Mark HM Winands, Yngvi Bjornsson, and Jahn-Takeshi Saito. Monte carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.

[4] Wei Zhan, Liting Sun, Di Wang, Haojie Shi, Aubrey Clausse, Maximilian Naumann, Julius Kümmerle, Hendrik Königshof, Christoph Stiller, Arnaud de La Fortelle, and Masayoshi Tomizuka. INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps. *arXiv:1910.03088 [cs, eess]*, 2019.

[5] Alberto Broggi, Massimo Bertozzi, Alessandra Fascioli, C Guarino Lo Bianco, and Aurelio Piazzi. The argo autonomous vehicle's vision and control systems. *International Journal of Intelligent Control and Systems*, 3(4):409–441, 1999.

[6] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer, 2016.

[7] Yang Bai, Zhuang Jie Chong, Marcelo H Ang, and Xueshan Gao. An online approach for intersection navigation of autonomous vehicle. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 2127–2132. IEEE, 2014.

[8] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The 2005 DARPA grand challenge: the great robot race*, volume 36. Springer, 2007.

[9] SAE On-Road Automated Vehicle Standards Committee et al. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Standard J*, 3016:1–16, 2014.

[10] Mohammad Shokrolah Shirazi and Brendan Tran Morris. Looking at intersections: a survey of intersection monitoring, behavior and safety analysis of recent studies. *IEEE Transactions on Intelligent Transportation Systems*, 18(1):4–24, 2016.

[11] Yuxiao Chen, Huei Peng, and Jessy Grizzle. Obstacle avoidance for low-speed autonomous vehicles with barrier function. *IEEE Transactions on Control Systems Technology*, 26(1):194–206, 2017.

[12] Ming-Yuan Yu, Ram Vasudevan, and Matthew Johnson-Roberson. Risk assessment and planning with bidirectional reachability for autonomous driving. *arXiv preprint arXiv:1909.08059*, 2019.

[13] Wei Zhan, Changliu Liu, Ching-Yao Chan, and Masayoshi Tomizuka. A non-conservatively defensive strategy for urban autonomous driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 459–464. IEEE, 2016.

[14] Xiaohui Li, Zhenping Sun, Dongpu Cao, Zhen He, and Qi Zhu. Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, 21(2):740–753, 2015.

[15] Xin Huang, Stephen G McGill, Brian C Williams, Luke Fletcher, and Guy Rosman. Uncertainty-aware driver trajectory prediction at urban intersections. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9718–9724. IEEE, 2019.

[16] Mohammad Shokrolah Shirazi and Brendan Tran Morris. Trajectory prediction of vehicles turning at intersections using deep neural networks. *Machine Vision and Applications*, 30(6):1097–1109, 2019.

[17] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1(1):1, 2014.

[18] Alex Zyner, Stewart Worrall, and Eduardo Nebot. Naturalistic driver intention and path prediction using recurrent neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[19] Derek J Phillips, Tim A Wheeler, and Mykel J Kochenderfer. Generalizable intention prediction of human drivers at intersections. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1665–1670. IEEE, 2017.

[20] Constantin Hubmann, Nils Quetschlich, Jens Schulz, Julian Bernhard, Daniel Althoff, and Christoph Stiller. A pomdp maneuver planner for occlusions in urban scenarios. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2172–2179. IEEE, 2019.

[21] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[22] Mathieu Barbier, Javier Ibañez-Guzmán, Christian Laugier, and Olivier Simonin. Validation framework applied to the decision-making process for an autonomous vehicle crossing road intersections. In *Validation and Verification of Automated Systems*, pages 179–205. Springer, 2020.

[23] Lin Hu, Jian Ou, Jing Huang, Yimin Chen, and Dongpu Cao. A review of research on traffic conflicts based on intelligent vehicles. *IEEE Access*, 8:24471–24483, 2020.

[24] Stefan Klingelschmitt, Florian Damerow, and Julian Eggert. Managing the complexity of inner-city scenes: An efficient situation hypotheses selection scheme. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 1232–1239. IEEE, 2015.

[25] Ralf Kohlhaas, Thomas Bittner, Thomas Schamm, and J Marius Zöllner. Semantic state space for high-level maneuver planning in structured traffic scenes. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1060–1065. IEEE, 2014.

[26] Lihua Zhao, Ryutaro Ichise, Tatsuya Yoshikawa, Takeshi Naito, Toshiaki Kakinami, and Yutaka Sasaki. Ontology-based decision making on uncontrolled intersections and narrow roads. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 83–88. IEEE, 2015.

[27] Constantin Hubmann, Michael Aeberhard, and Christoph Stiller. A generic driving strategy for urban environments. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1010–1016. IEEE, 2016.

[28] Stefan Klingelschmitt, Volker Willert, and Julian Eggert. Probabilistic, discriminative maneuver estimation in generic traffic scenes using pairwise probability coupling.

In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1269–1276. IEEE, 2016.

[29] Oren Salzman and Dan Halperin. Asymptotically near-optimal rrt for fast, high-quality motion planning. *IEEE Transactions on Robotics*, 32(3):473–483, 2016.

[30] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[31] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[32] Yoshiaki Kuwata, Justin Teo, Sertac Karaman, Gaston Fiore, Emilio Frazzoli, and Jonathan How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7166, 2008.

[33] Liang Ma, Jianru Xue, Kuniaki Kawabata, Jihua Zhu, Chao Ma, and Nanning Zheng. Efficient sampling-based motion planning for on-road autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1961–1976, 2015.

[34] Chao Chen, Markus Rickert, and Alois Knoll. Combining task and motion planning for intersection assistance systems. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1242–1247. IEEE, 2016.

[35] Sangyol Yoon, Dasol Lee, Jiwon Jung, and David Hyunchul Shim. Spline-based rrt using piecewise continuous collision-checking algorithm for car-like vehicles. *Journal of Intelligent & Robotic Systems*, 90(3-4):537–549, 2018.

[36] Siddhartha S Mehta, Chau Ton, Michael J McCourt, Zhen Kan, Emily A Doucette, and W Curtis. Human-assisted rrt for path planning in urban environments. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 941–946. IEEE, 2015.

[37] Alex Zyner, Stewart Worrall, and Eduardo Nebot. A recurrent neural network solution for predicting driver intention at unsignalized intersections. *IEEE Robotics and Automation Letters*, 3(3):1759–1764, 2018.

[38] Debaditya Roy, Tetsuhiro Ishizaka, C Krishna Mohan, and Atsushi Fukuda. Vehicle trajectory prediction at intersections using interaction based generative adversarial networks. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2318–2323. IEEE, 2019.

[39] Yonghwan Jeong, Kyongsu Yi, and Sungmin Park. Svm based intention inference and motion planning at uncontrolled intersection. *IFAC-PapersOnLine*, 52(8):356–361, 2019.

[40] Yonghwan Jeong, Seonwook Kim, and Kyongsu Yi. Surround vehicle motion prediction using lstm-rnn for motion planning of autonomous vehicles at multi-lane turn intersections. *IEEE Open Journal of Intelligent Transportation Systems*, 1:2–14, 2020.

[41] Yijing Wang, Zhengxuan Liu, Zhiqiang Zuo, Zheng Li, Li Wang, and Xiaoyuan Luo. Trajectory planning and safety assessment of autonomous vehicles based on motion prediction and model predictive control. *IEEE Transactions on Vehicular Technology*, 68(9):8546–8556, 2019.

[42] Yadollah Rasekhipour. Prioritized obstacle avoidance in motion planning of autonomous vehicles. 2017.

[43] Georg Schildbach, Matthias Soppert, and Francesco Borrelli. A collision avoidance system at intersections using robust model predictive control. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 233–238. IEEE, 2016.

[44] Robert Hult, Mario Zanon, Sebastien Gros, and Paolo Falcone. Optimal coordination of automated vehicles at intersections: Theory and experiments. *IEEE Transactions on Control Systems Technology*, 27(6):2510–2525, 2018.

[45] Julia Nilsson, Mattias Brännström, Jonas Fredriksson, and Erik Coelingh. Longitudinal and lateral control for automated yielding maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 17(5):1404–1414, 2016.

[46] Chang Liu, Seungho Lee, Scott Varnhagen, and H Eric Tseng. Path planning for autonomous vehicles using model predictive control. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 174–179. IEEE, 2017.

[47] Changliu Liu, Wei Zhan, and Masayoshi Tomizuka. Speed profile planning in dynamic environments via temporal optimization. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 154–159. IEEE, 2017.

[48] Maxime Bouton, Alireza Nakhaei, Kikuo Fujimura, and Mykel J Kochenderfer. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1469–1476. IEEE, 2019.

[49] Danial Kamran, Carlos Fernandez Lopez, Martin Lauer, and Christoph Stiller. Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning. *arXiv preprint arXiv:2004.04450*, 2020.

[50] Tommy Tram, Anton Jansson, Robin Grönberg, Mohammad Ali, and Jonas Sjöberg. Learning negotiating behavior between cars in intersections using deep q-learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3169–3174. IEEE, 2018.

[51] Ali Demir and Volkan Sezer. Intersection navigation under dynamic constraints using deep reinforcement learning. In *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pages 1–5. IEEE, 2018.

[52] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039. IEEE, 2018.

[53] Zhiqian Qiao, Katharina Muelling, John M Dolan, Praveen Palanisamy, and Priyantha Mudalige. Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1233–1238. IEEE, 2018.

[54] Maxime Bouton, Akansel Cosgun, and Mykel J Kochenderfer. Belief state planning for autonomously navigating urban intersections. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 825–830. IEEE, 2017.

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

[56] Constantin Hubmann, Jens Schulz, Marvin Becker, Daniel Althoff, and Christoph Stiller. Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *IEEE Transactions on Intelligent Vehicles*, 3(1):5–17, 2018.

[57] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 392–399. IEEE, 2014.

[58] Xiao Lin, Jiucai Zhang, Jin Shang, Yi Wang, Hongkai Yu, and Xiaoli Zhang. Decision making through occluded intersections for autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2449–2455. IEEE, 2019.

[59] Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088, 1973.

[60] Edward J Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations research*, 26(2):282–304, 1978.

[61] Chelsea C White. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.

[62] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application.* MIT press, 2015.

[63] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

[64] Edward Jay Sondik. The optimal control of partially observable markov processes. Technical report, Stanford Univ Calif Stanford Electronics Labs, 1971.

[65] Hsien-Te Cheng. *Algorithms for partially observable Markov decision processes.* PhD thesis, University of British Columbia, 1988.

[66] Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, pages 1023–1028, 1994.

[67] Nevin L Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical report, Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology, 1996.

[68] Brian Sallans. Learning factored representations for partially observable markov decision processes. In *Advances in neural information processing systems*, pages 1050–1056, 2000.

[69] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[70] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R Cassandra. Solving pomdps by searching the space of finite policies. *arXiv preprint arXiv:1301.6720*, 2013.

[71] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI/IAAI*, pages 541–548, 1999.

[72] Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*, 2012.

[73] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

[74] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003.

[75] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[76] Dimitri P Bertsekas and David A Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

[77] Michael J Kearns, Yishay Mansour, and Andrew Y Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems*, pages 1001–1007, 2000.

[78] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[79] Andreas Ten Pas. *Simulation based planning for partially observable markov decision processes with continuous observation spaces.* PhD thesis, Citeseer, 2012.

[80] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. 2019.

[81] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48(12):4743–4759, 2018.

[82] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.

[83] Huina Chen, Xiaonian Wang, and Jun Wang. A trajectory planning method considering intention-aware uncertainty for autonomous vehicles. In *2018 Chinese Automation Congress (CAC)*, pages 1460–1465. IEEE, 2018.

[84] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.