Behavior Prediction in Autonomous Driving

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Department of Electrical and Computer Engineering

Chiyu Dong

B.S., Precision Instruments and Mechanology, Tsinghua University

M.S., Mechanical Engineering, Carnegie Mellon University

Carnegie Mellon University

Pittsburgh, PA

May 2019

Acknowledgments

I would like to express my deepest gratitude to my supervisor Professor John M. Dolan. This thesis would hardly have been completed without his patience, motivation, and immense knowledge. I also want to express my warmest gratitude to my thesis committee: Professor Stephen F. Smith, Professor Anthony Rowe, and Dr. Bakhtiar B. Litkouhi.

My sincere thanks also goes to all colleagues in GM-CMU ADCRL: Professor Ragunathan Rajkumar, Jarrod Snider, Junqing Wei, Tianyu Gu, Wenda Xu, Gaurav Bhatia, Hyoseung Kim, Jongho Lee, and Junsung Kim. They enlightened me the first glance of the autonomous driving research. I would also like to thank General Motors Research and Development for sponsoring the research.

I am forever thankful to my labmates: Zhiqian Qiao, Jordan Ford, Chen Fu, Yilun Chen, Aman Khurana, Adam Villaflor, Yanjun Pan and Jing Zhao. The hard time before deadlines and fun days together are unforgettable. I thank my colleagues in the department of Electrical and Computer Engineering, the Robotics Institute and Mechanical Engineering. In particular, I am grateful to Wenhao Luo, Sasanka Nagavalli, Yuqing Tang, Hsu-Chieh Hu, Anqi Li, Weikun Zhen, for fruitful discussions in research topics.

Last but not the least, I would like to thank my family, especially my parents, for their unconditional and endless support, inspiration and encouragement with my academic pursuit and my life in general.

Abstract

As autonomous driving vehicles are being tested on public roads, they will share the road with human-driven vehicles. It becomes important for autonomous driving vehicles to estimate human drivers' intentions in order to interact properly with the human drivers to achieve safe and efficient experiences. The current work proposes a new cooperative driving framework which is capable of predicting other vehicles' behaviors. The estimated prediction provides an input for a trajectory planner to perform cooperative behavior and to generate a path to react to other vehicles. The system has three stages:

1 Abstract intention prediction;

2 Intermediate-level important points prediction;

3 Ultimate trajectory prediction.

The system bridges the gap between higher-level mission planning and behavioral execution or trajectory planning, especially in interactive scenarios. The validation contains two aspects: Firstly, the estimated trajectory is compared with the ground-truth in datasets. Secondly, the estimated trajectory is applied to current trajectory planners to generate cooperative plans. The second step evaluates the closed-loop performance of the behavioral estimation in the whole system. The proposed method outperforms previous solutions in terms of collision rates, safety distance, and error when tested against a human-driven trajectory database. The method is implemented in simulation and on a real autonomous driving platform to test its feasibility in real scenarios.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous vehicles have been testing on public roads for some time. Safety and economical concerns are the top reasons for the development of autonomous driving technology. The analysis from the National Highway Traffic Safety Administration (NHTSA) shows that human drivers are the leading causes of unsafe events. Recognition and decision error are the leading causes for these events, and contribute 41% and 34% of the total errors, respectively. The development of autonomous driving techniques aims to reduce accidents, costs, and human labor, and to increase transportation efficiency.

NHTSA has also published a white paper [1] suggesting 4 levels of autonomous driving. The highest level is full autonomy in all environments, whereas the lowest level is manual driving. The intermediate levels differ in whether the vehicle just warns the driver or operates the vehicle, and how many functions are controlled by the vehicle itself. There is an increasing number of companies seeking to commercialize autonomous driving vehicles at level 3 or 4. For example, Google has been testing their fully automated vehicle since 2009, and has accumulated more than one million miles of testing data on public roads. Its autonomous driving department has now "graduated" from Google [X] and established Waymo to further promote its self-driving vehicle. Uber started testing their autonomous driving vehicle on public roads in the Pittsburgh area in 2016. It is also the first company to offer a commercial vehicle sharing service. Lyft and Aptiv released their autonomous ride-sharing service in parts of Las Vegas after Uber announced their self-driving public road tests.

Unlike IT companies, who built their vehicles mainly for research and exploration, traditional car makers have moved rather slowly, since they are not only interested in the specific advanced technique, but also commercialization problems such as industrial integration, supply chain, marketing and safety. Car makers such as General Motors, Ford, Tesla, Volkswagen, Mercedes-Benz, BMW, and Audi have been gradually and continuously integrating state-of-the art autonomous driving techniques or Advanced Driver-Assistance Systems (ADAS) into their commercial products. For example, they offer Lane Departure Warning Systems (LDWS), Lane Keeping Systems(LKS), Collision Warning (CW), and Full Speed Range Adaptive Cruise Control (FSRACC). General Motors' "SuperCruise" performs single-lane hands-free autonomous driving at freeway speeds; Tesla's "AutoPilot" combines lane keeping with ACC on the highway, and features the lane-changing assistant. Audi's "Stop and Go" frees human drivers from having to control vehicles in congestion at low speeds. Nowadays, almost all car makers, ranging from basic car brands like Toyota, Honda and Subaru to luxury brands, can provide consumers with optional self-driving packages which enable the cars with level 2 or 3 features.

As the technology proceeds, in the near future, autonomous driving vehicles will be commercialized and share roads with human-driven vehicles. However, current autonomous driving systems do not deal well enough with complex environments filled with human-driven vehicles. The main reason is that the current systems are not able to accurately predict the intention or future movement of surrounding vehicles and then negotiate with them. Ideally, there should be a prediction module in the autonomous driving system architecture. There are two modules in most systems that are related to the understanding and negotiation, i.e., Behavior and Trajectory Planning. The behavior planning module normally outputs high-level commands, such us "Go to the goal", "Make a lane change" or "Turn left". The trajectory planning then generate a feasible plan to conduct the command given by the behavior planner. Unfortunately, most behavior planners are not able to predict human behaviors and negotiate with human drivers, then generate interactive behavioral commands. Similarly, in order to generate cooperative trajectories, most of the planning algorithms require an accurate trajectory estimate for other vehicles.

It is therefore important for the autonomous driving vehicle to have an extra module to enable the car to understand other human drivers' intentions, predict human-driven vehicles' behaviors in the near future, and then use the well-developed trajectory planning algorithms to socially cooperate with surrounding vehicles. Figure 1.1 shows the whole autonomous driving system architecture, without including detailed perception modules and controller modules. The main idea of this figure is to illustrate the differences between the classic framework and our prediction-augmented one. The dashed-line box highlights the proposed extra module. Note that in the previous framework, there is not an extra module between the behavioral and trajectory planner. As discussed above, the intermediate layer that we propose is a prediction engine. To better manage and fulfill different scenarios or tasks, the prediction layer consists of various sub-modules with differing complexity and computational demands.

A command from the Behavioral Planner, for example, Lane Change, will be sent to the Behavioral Estimation layer. In this layer, it will activate a corresponding module to process the prediction, for example, the Trajectory Prediction for all surrounding vehicles. After the prediction, the Behavioral Estimation layer will select a proper module in the actuation level. In this case, the predicted trajectories will be sent to the trajectory planner to generate a cooperative lane-change path plan. In the case of ramp merging, since the task is not as complex as lane change, after Intention Estimation is completed, a lower-level distance-keeping model is activated in the actuation layer, instead of calculating a full trajectory. Details will be discussed in Chapter 3. This framework allows the system to use computational power efficiently, and adopt the most robust actuators/controllers for individual tasks.

However, human drivers' behaviors contain uncertainty, and there are numerous factors affecting these behaviors. Modeling the influence of these factors on a human driver's behavior is a challenging task. These factors include but are not limited to:

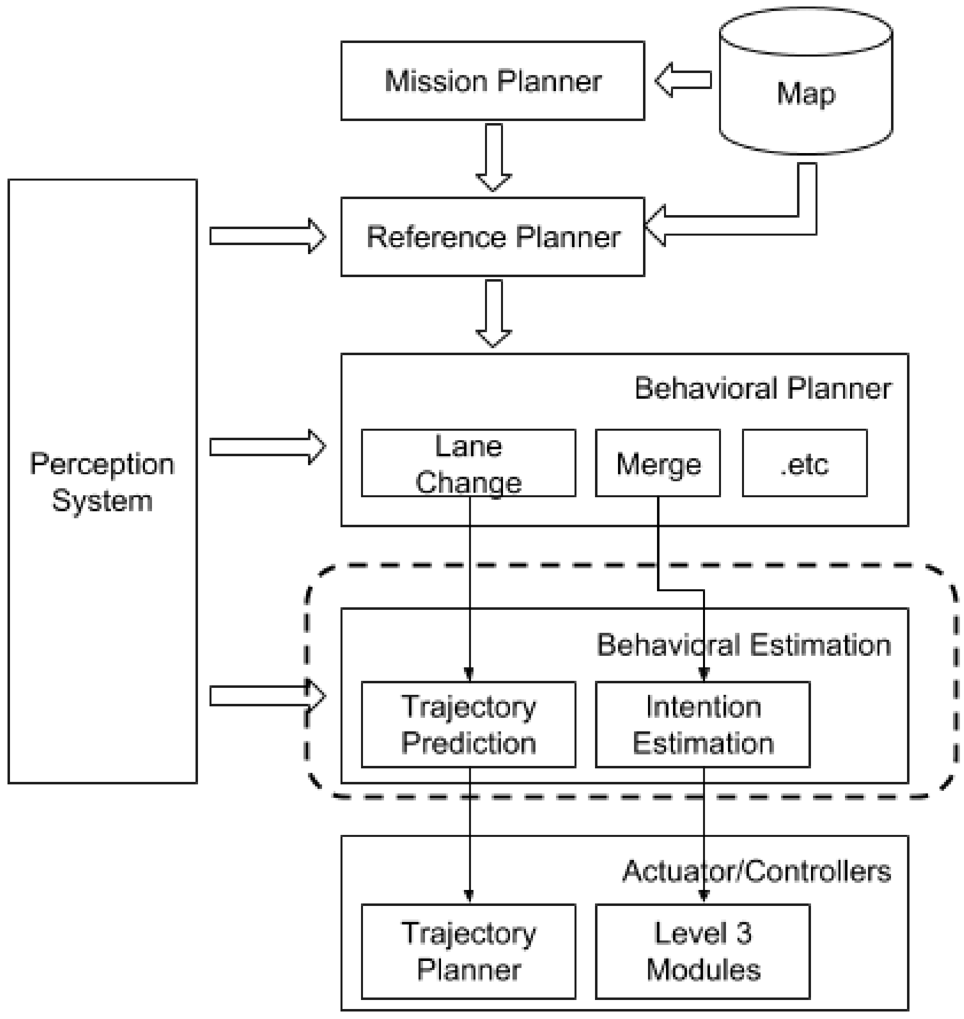


Figure 1.1: An overview of the autonomous driving system architecture and the proposed module. Arrows represent data flow. The dashed bounding box highlights the key contribution of this dissertation.

1 The current goal, such as lane-keeping, lane-changing, ramp-merging, and intersection-traversing. The ego-vehicle should perform a proper behavior to achieve the goal.

2 Surrounding traffic vehicles' status, e.g., one or some of the surrounding vehicles make a lane-change or a car in the neighboring lane speeds up to prevent a lane change.

3 Traffic rules, signs, and signals. For example, if a "Yield" sign appears at the end of an entrance ramp, a merging vehicle should wait for the main-road vehicle until the traffic condition meets the merging requirement.

Since these factors can be coupled, the driver's behavior becomes even more complicated. For instance, in dense urban traffic, say that a vehicle plans to make a left turn at the next intersection. The current goal is "Left-Turn". Unfortunately, there is no green arrow for a left-turn, so the ego vehicle must keep monitoring the approaching vehicles from the opposite direction while waiting for a condition that satisfies the left turn requirements. This requires the vehicles to understand the approaching vehicles' intentions, have a reasonable prediction of their behaviors, and finally perform a sequence of proper reactions. Due to the complexity of the interaction, it is extremely hard for a single model to handle all cooperative scenarios, such as ramp-merging, lane-changing or intersection traversing. However, a universal framework may be able to serve as a solution for various scenarios, with minor modifications or specific data to fit in.

The core task for such a framework is to formulate the interaction between the host vehicle and other human-driven vehicles. Then, a prediction of surrounding vehicles' behaviors and trajectories can be generated. Based on the prediction, a cooperative trajectory for the vehicles can be planned. The framework bridges the gap between the trajectory planner and higher-level task planner, especially in dynamic environments where the trajectory planner needs other vehicles' future movements as a reference. Because there are differences in the abstraction of input information, the trajectory planners and higher-level planners (such as reference planners and mission planners) can encounter conflicts in the decision making and its execution. For example, the behavioral planner may send out a command to make a lane-change, but due to the dense traffic or aggressive human drivers' behaviors, the trajectory planner cannot generate a feasible and collision-free path in a very limited area and time.

Generally, a behavioral planner does not consider the failure of lower-level planners (the trajectory planner and controller), and there is also no feedback mechanism if the lower-level planners/controllers do not work properly. Addressing this problem has two aspects: 1. The behavioral planning should consider more detailed information and have the capability to predict environmental changes, e.g., the future trajectories of surrounding vehicles, with bounded uncertainty. 2. The behavioral planner should take feedback from the trajectory planner and always prepare to make follow-on decisions. The second step serves as a backup mechanism if the first step fails. The feedback mechanism will be the future work followed by this project. The first step is the main topic of this dissertation. To accomplish the first step, the autonomous driving vehicle needs a prediction engine. The prediction engine takes surrounding vehicles' trajectories as inputs and it outputs a future trajectory for a target vehicle.

To fulfill the requirements for the different levels of decision making and planning, there are three stages of prediction:

1 Discrete intentional prediction: The behavioral predictor only gives an abstract command from a discrete options set. For example, in a merging scenario, the ego-vehicle on the main road only needs to know whether one merging vehicle intends to yield or not, then the ego-vehicle will select a needed target to keep a safe distance, either the leading car in the current lane or the merging vehicle.

2 Semi-continuous prediction: In this stage, the behavioral predictor implicitly considers the surrounding vehicles' movements and outputs key points for the ego-vehicle to aid cooperative driving. For example, in a dense-traffic lane-change scenario, the path planner in the ego-vehicle needs to know the location where it is possible to make/finish a lane-change, and the instant when the car can start/end the behavior. Those key points are start and end points for the cooperative behavior. However, there is no explicit modeling of

surrounding vehicles' movements; instead, direct advice (the start/end points) is given to the vehicles. Trajectory planners highly rely on those key points to make a detailed path with a speed profile.

3 Continuous prediction: the object of this stage is one of the surrounding vehicles. It focuses on predicting its future trajectory by explicitly analyzing the interaction between other surrounding vehicles and the factors which are discussed below.

The predicted trajectories of all surrounding vehicles provide more detailed information for decision making and path planning. Most existing decision making and path planning algorithms do not consider other agents' future trajectories, or simply assume a constant model for all agents. Those solutions work well in low-speed robot agent interactions. However, neither is sufficiently realistic for autonomous driving vehicles, which face more challenging situations such as higher speeds, more dynamic environments, dense traffic interactions, and most importantly, extremely high safety and real-time requirements. These requirements naturally increase demands on the behavioral predictor: 1. Accuracy is the most obvious criterion. The predicted trajectory should be similar to the real human-driven trajectory in the same situation, and must have a bounded uncertainty. This criterion suggests that the algorithm should learn human-like behaviors from real data. 2. The algorithm should be consistent and robust. Since the decision making should be done at each timestep, but the prediction is over a long horizon, the results cannot change frequently or dramatically.

Besides these cooperative scenarios, the behavioral prediction engine can be applied to other kinds of situations. For example, as indicated in [2], in a two-direction narrow road with soft lane dividers, the ego-vehicle can enter the opposite lane to swerve around a large parked car, overtake a slow-moving vehicle or keep a safe distance to a bicyclist. To enable such behavior, especially in dense traffic areas, it is necessary to analyze the opposing vehicle's behavior. Normally, this approaching vehicle may not slow down for the ego-vehicle, thus, if it needs to perform such a behavior, it should accurately predict the incoming car's trajectory, and then plan a corresponding path to cross the lane divider and successfully perform the task without seriously interfering with vehicles moving in the opposite direction.

1.1 Problem Definition

Using probabilistic methods will improve the autonomous driving cars' behavior in interactive scenarios. The estimation of surrounding vehicles' future behaviors includes discrete intentions, important way points, and possible trajectories. It should consider the interaction between all surrounding traffic participants. The prediction is based on the historical observation of these vehicles. The main difficulty is to represent a continuous trajectory and compare the similarity between trajectories. Another challenge is to smoothly update a predicted trajectory over time according to the changes in surrounding traffic. In a symbolic representation, the system is a function which maps observations to predicted behavior:

$$O^{\tau} \xrightarrow{f^{\tau}} B^{\tau}$$

where $O \in R^{n \times t}$ denotes observation, which is a tuple containing n surrounding vehicles' dynamic information over the last t seconds; $B \in R^{h}$, which is an array of predicted behaviors for the target vehicle; and f_i is the predictor. Note that all of $O^{\tau}, B^{\tau}, f^{\tau}$ have specific forms to manage one scenario and task $\tau \in \mathbf{T}$. The main topic of this dissertation is to figure out f^{τ} for specific tasks τ. Probabilistic methods are proposed in the dissertation to understand other human-driven cars' behaviors and take the uncertainty of human drivers' intentions in to consideration. In addition, the forms of O^{τ}, B^{τ} will also be discussed, according to different scenarios.

1.2 Contributions

The main contribution of this thesis is a behavioral estimation framework for socially cooperative driving. The work is a probability framework which tolerates the uncertainty of traffic conditions and human drivers. The framework has the following novel features compared with previous work:

1 It estimates behaviors by considering historical trajectories of surrounding vehicles;

2 It smoothly updates the estimates according to the surrounding traffic using a non-parametric method, without assuming a human driver model;

3 It separates the behavioral estimation into three levels with increasing complexity to satisfy the requirements of different scenarios.

To validate the performance of the method, a realistic simulation environment is built. Real human-driven data are used as other agents and ground-truth. The simulation helps the test and integration of the system. Furthermore, sensitivity analysis is conducted in the simulator. Different types of sensing uncertainty and their effects are studied. This analysis gives guidelines and criteria for the design of an engineering realization of a perception system. Finally, the system is deployed on a real autonomous driving system to show its performance in urban traffic.

Chapter 2

Related Work

2.1 Social Behavior in human

Before reviewing behavioral planning methods for autonomous vehicles, we look to the field of psychology for a definition of “Social Behavior” among people. This definition can be applied to the social behaviors among autonomous vehicles and other human-driven vehicles.

2.1.1 Social Behavior in Social Psychology

In psychology research [3], social behaviors and interactions are defined as follows: “ Behavior that is peculiarly social is oriented towards other selves. Such behavior apprehends another as a perceiving, thinking, moral, intentional, and behaving person; considers the intentional or rational meaning of the others’ field of expression; involves expectations about the others’ acts and actions and manifests an intention to invoke in another self-certain experiences and intentions. ”

In autonomous driving scenarios, the ego-vehicle should understand surrounding vehicles’ actions and intentions, in order to achieve social behaviors in traffic.

2.1.2 Decision making in human brain

Rustichinir [4] reviewed two major theories which explain the decision making mechanism in the human brain. Though the real mechanism remains a topic of research in the field of psychology, it is agreed that there are different stages of decision making which are governed by different modules in the human brain. Decision making is described as a ”dual system”: immediate and long-term. Immediate decision making often occurs subconsciously, whereas long-term decision making “calculates” the accumulated rewards over time, which is similar to the framework of Markov Decision Making models. Different “modules” are in charge of different decisions. In other words, for a specific event, different modules can output opposite decisions. The problem is how the human selects a proper decision in the face of contradiction. Although psychologists do not agree on the architecture of these modules, in robotics, where the decision system is not as complex as that of a human, different types of decision modules in the system apply to different scenarios. An upper-level model activates the proper module to handle a specific situation, and the module is followed by a proper lower-level planner or controller to execute the behavioral planning.

Following this dual system, reflective-impulsive models [5] are applied to social behavior. The impulsive model explains the immediate decision-making, and the reflective part of the model considers decisions over long horizon with prediction. This part shares the same idea with the design of the behavioral planning of autonomous vehicles: collect enough past data to analyze and consider long-horizon rewards and mutual benefit at the same time.

2.2 Social behavior in autonomous vehicles

The social behavioral planner in autonomous vehicles is not as complicated as the decision-making component of the human brain. However, different approaches are able to manage specific scenarios, from simple cases to complex ones. In the autonomous vehicle application, there are three kinds of approaches to address cooperative driving.

- Rule-based
- Optimization- and Search-based
- Probabilistic

2.2.1 Rule-based methods

Rule-based methods are the most straightforward approaches. They have been applied on test vehicles since the 2007 DARPA Urban Challenge. Baker and Dolan [6] developed CMU Boss's merge planner using a slot-based approach. Kinematic information is used to check merge-in feasibility of each slot, such as the distance to the Goal, remaining distance in the current lane, etc. Then the target slot is selected from the set of feasible slots according to the context of the maneuver, and predictions of others. The slot-based approach is straightforward to implement and robust in simple scenarios. However, the lack of prior knowledge of surrounding vehicles' intentions makes it hard to estimate or predict their movements and corresponding behaviors. Most teams in the Urban Challenge used rule-based methods, such as a state machine with pre-defined behaviors [7], for decision making and behavioral planning [8, 9, 10]. These approaches are efficient, deterministic and feasible in the competition field, since the scenarios had fewer uncertainties than real driving environments.

Naranjo et al. [11] perform lane-change decision making by using fuzzy logic. The method is also straightforward and simple to implement. However, it also does not consider prior knowledge and prediction. Lu and Hedrick [12] formulated the merge planning as a platoon control problem. But the algorithm still requires a "coordination layer" to select a pair of main-road cars to interact with, i.e., it needs a decision to yield or not yield by the host vehicle. The rule-based algorithms mainly consider kinematic variables such as speed, distance or time-to-arrival, and make an instantaneous decision based on the current observations. These algorithms are suitable for simple scenarios which have a limited number of variables. The rule-based algorithms do not consider the reaction of other vehicles to the behaviors that are performed by the ego-vehicle. Therefore, rule-based algorithms can easily fail in dense traffic or interactive scenarios, or need an arbitrarily large expert system, which is infeasible for storage and efficient computation.

2.2.2 Optimization and Search-based methods

Nilsson et al. [13, 14] formulated cooperative planning as an optimization problem under a Model Predictive Control (MPC) framework. The weighted effects of acceleration and braking are optimized subject to the trajectory's shape and feasibility. The author provided a straightforward way to transform the problem into a well-defined optimization problem that can be solved by applying a specific solver. However, the manual tuning of weights is difficult. Also, the equation to be optimized and objective functions are designed by hand, without the use of data.

Liu et al. [15] convert the MPC to a convex optimization over its manifold, and achieved better performance and computational improvements. Their method is a sound theory to calculate a proper cooperative trajectory planning for robots. But it still needs a predictive engine to provide an initial estimate of the other agents' possible future trajectories and the associated uncertainty. Algorithms based on the Intelligent Driver Model (IDM) [16] optimize the ego-vehicle's reactions in scenarios requiring interaction such as freeway entrances and turns in intersections. The algorithms assume that all other vehicles apply the identical behavioral model (IDM). Given a proposed intention for the ego-vehicle, the optimization algorithms converge to a proper cooperative trajectory regarding the reactions from the IDM-driven agents. However, this approach has some problems. Firstly, the assumption of IDM does not necessarily fit all other vehicles, especially in interactive scenarios. Secondly, the IDM relies on several parameters which require hand-tuning. Thirdly, despite the fact that IDM is one of the most robust distance-keeping models, interactive scenarios do not merely require car-following, which only considers the leading car. Instead, a vehicle should also consider all surrounding vehicles' behaviors and then react.

Xu et al. [17] combined the path and velocity in a single search and optimization process to make the car navigate traffic. This work only considered the current interaction among cars, instead of explicitly predicting a target car's motion.

In order to optimize a cooperative trajectory, these methods need to know the future trajectory of other traffic participants, which is not included in these methods. In other words, these methods do not sufficiently consider future behaviors of surrounding vehicles.

2.2.3 Probabilistic methods

Probabilistic methods form the largest percentage of solutions to lane changing or cooperative driving. Montemerlo et al. [10] integrated lane-changing behavior into Stanford Junior's global path planner, which is an instance of dynamic programming (DP). In fact, the problem is formulated as optimizing a variant Bellman equation, which implicitly follows the MDP framework and value iteration. Each action is assigned a penalty cost. The lane changing behavior is a penalty term in the cumulative cost function which is optimized by the DP. However, the algorithm does not consider other traffic participants. Yao et al. [18] search for k-nearest-neighbors in a lane-change scenario database to generate a trajectory. Measuring differences between trajectories and scenarios remains a problem. If the data-set contains a large number of samples, searching for the k-nearest-neighbors is time-consuming. Galceran and Cunningham et al. [19, 20] make the decision depending on the probability of past trajectories of all traffic participants. Both of them report discrete actions such as left-lane-change, right-lane-change etc., which can be used as an upper-level module in our method. Dong et al. [21, 22] detect whether the other car will merge in by using PGM. However, this method only provides binary output of either Yield or Not Yield. Ulbrich et al. [23] and Wei et al. [24] proposed an online PoMDP [25] for lane-change using real-time belief space search [26]. However, to achieve real-time performance and use a simple PoMDP framework, they discretized state and action spaces. To avoid discrete states, Bai et al. [27] proposed a continuous-state PoMDP using a belief tree and the model was applied to navigating intersections. However, its actions are discrete and represented by a generalized policy graph (GPG). Seiler et al. [28] proposed an online and approximate solver for a continuous-action PoMDP, but only tested in toy problems, such as a 4-by-4 maze. The PoMDP solutions above still need manually designed probabilistic transition models and reward functions. Hadfield et al. [29] establish those transition models by (inverse) reinforcement learning, but their solutions are tied to specific scenario features, such as the number of traffic participants. E. Ward et al. and Klingelschmitt et al. [30, 31] predict the intention probabilities and then use the Intelligent Driver Model (IDM) as a planning module to execute actions such as over-take and right/left turns in traffic conditions or intersections. Althoff and Magdici [32] introduced Set-Based Prediction for traffic participants on arbitrary road geometry. In this work, vehicle dynamics and route shape are considered. However, the method does not consider mutual interactions in the constraints. It also tends to over-estimate the reachable sets of surrounding vehicles, which results in larger occupied regions and narrow corridors, sometimes making it impossible to traverse.

As Deep Learning techniques advance, there are methods that adopt Deep Network for the interactive behavior prediction. Kuefler et al. [33] used Generative Adversarial Networks (GANs) to mainly imitate and estimate single-lane behaviors (e.g., car following). Chen et al. [34] and Su et al. [35] solved the sequential prediction of the lane-change trajectory by using recurrent deep neural networks such as Long Short-Term Memory (LSTM). However, the interaction is not explicitly modeled in these LSTM or GANs methods. To better modeling the interaction among autonomous driving cars and other human-driven cars, Qiao et al. [36] and Sadigh et al. [37] used reinforcement learning to describe the interaction behaviors among vehicles and decision making for autonomous driving cars in intersections and lane change, respectively. In addition, Yan et al. [38] formulate highway decision making as a game theory problem, and a deep neural network is used to approximates a payoff in the game. However, those algorithms are mainly developed and tested in simulators. There are so called “realistic gap” between the real world and the simulators. To close the gap, Zhan et al. [39] and Li et al. [40] mainly focused on the safety guarantees for the probabilistic methods, and its deployment from simulation to real environments.

2.3 Summary

All of these methods have their own advantages and disadvantages. The rule-based algorithms are straightforward, understandable, and easy to tune and have low execution time. Empirically, rule-based algorithms can make decisions for easy and obvious scenarios. They have trouble with subtle scenarios since it is infeasible to enumerate all possible situations and their combinations for an expert system. In the case of complex scenarios, these systems have a high chance to suffer from logic loopholes. Additionally, the rule-based algorithms do not explicitly analyze surrounding vehicles' intentions and their future movements, which are important information for social behavioral planning. Most rule-based algorithms make decisions based on the current circumstance, regardless of possible environmental changes. This is based on the current observations, and there is no guarantee that they can result in consistent estimates. The result will oscillate if the environment changes or sensor noise is too severe.

The optimization-based algorithms are theoretically sound and guarantee an optimal result with respect to constraints. Since the only parts which require human design are objective functions and constraints which abstract scenarios and reward desired actions (or penalize undesired actions), they are less labor-intensive than rule-based algorithms, which require enumerating cases. One more advantage over the rule-based algorithms is that rather than an abstract command, most optimization-based algorithms can directly output detailed and continuous lower-level control, for example, acceleration rate and steering. There are still limitations for optimization algorithms in generating cooperative behaviors. The optimization algorithms for cooperative driving relay on *prior* knowledge of other vehicles' movements. This information is extremely hard to obtain and is one of the main topics which the current work will discuss in depth. PoMDP-based algorithms mainly look for a complete solution for a policy. However, solving the PoMDP online remains a challenge, and the transition models are hard to extract, especially in continuous state and action spaces whose dimensions can be infinite.

Our proposed method does not require an overall policy based on the belief states; instead, it outputs the prediction with the maximum posterior probability. In addition, unlike Value Iteration or Gradient Descent solvers, the proposed method has a closed-form solution. Compared with previous methods, the proposed method explicitly predicts the trajectory of one target, considering its surrounding vehicles. The method updates the estimation in a Bayesian Filter fashion: once a new segment of the trajectory is observed, the whole estimate of the future trajectory is adjusted. The new approach does not assume a parametric transition model of the state updating. Instead, the transition probability is directly extracted and built from data. The proposed model is non-parametric, namely, there is no assumed model to represent the transition. However, the price of the non-parametric model is the size of training data. It takes all training trajectories and saves to a single transition matrix, which means that the size of the transition matrix grows as the training size increases. Though both the updating and estimation steps are closed-form linear transforms, the size of these matrices can be huge. Thus, the method requires large memory space and efficient algorithms to handle the matrix manipulation on such a large scale.

Table 2.1 shows the summary of all discussed prior work. There are five important aspects in evaluating different methods:

- *Interactive:* Whether the algorithm considers the reactions between vehicles.

- *Predictive:* Whether the method can predict the future movement of surrounding traffic. This is explicit prediction of the movement that can directly affect changes in the ego-vehicle's behavior. For example, [19] unrolls the candidate policies for forward prediction, and then evaluates all candidates to find the best one.

- *Robustness:* Whether the method is stable against observation noise and is able to avoid oscillating decisions.

- *Real-Time:* Whether the feasible behaviors can be evaluated in real time.
- *Full-Trajectory:* Whether the method can estimate a full trajectory for other cars or the ego-vehicle.

In addition, the “Info” column describes the input information of the listed algorithms, for example, V means speed, S means location or trajectory, and T means time-to-arrival or headway.

The main advantages of the proposed method compared with previous methods are:

- Surrounding vehicles’ historical trajectories are taken into consideration;
- The full-trajectory estimation is explicit, and will be smoothly updated in response to new observations over time;
- The estimation is in three levels to satisfy different levels of traffic complexity.

Unlike previous methods, our solution focuses on all five of these important aspects. A detailed performance comparison with other methods is given in Chapter 4. Realistic aspects for the implementation and engineering are considered, for example, the solution takes full advantage of current Level-3 features on commercial passenger cars; Sensitivity analysis is provided in Chapter 5 to provide a guideline for the engineering realization of a perception system that satisfies the requests of the behavior estimation system.

Table 2.1: Summary and comparisons for prior methods

Category	Method	Interactive	Predictive	Robustness	Real-Time	Full-Traj	Info
Rule Based	Baker [6]	×	×	✓	✓	×	V, S
	Naranjo [11]	×	✓	✓	✓	×	V, S
	Lu [12]	×	×	✓	✓	×	S
Optimization	Xu [17]	✓	×	✓	✓	✓	V, S
	Gu [41]	×	×	✓	✓	✓	V, S
	Nilsson [13]	✓	×	✓	×	✓[1]	S
	Liu [15]	✓	×	✓	✓	✓[1]	S
Probabilistic	Wei [42]	✓	✓	×	✓	×[1]	V
	Galceran [19]	✓	✓	✓	×	✓[2]	V
	Dong [21]	✓	✓	✓	✓	×	V, T
	Althoff [32]	✓	✓	✓	✓	×	V
	Sadigh [37]	✓	×	✓	×	✓[2]	S
	Proposed	✓	✓	✓	✓	✓[3]	S

[1] Only ego-vehicle's trajectory estimation, assumes a trajectory for the other car.

[2] A policy is given for the interaction, a trajectory can be derived from the policy.

[3] A prediction of a full trajectory is given.

Chapter 3

Methods

In a real heterogeneous scenario where autonomous vehicles share the road with human-driven vehicles, especially in dense traffic conditions or scenarios that require interactions, it is important for the autonomous driving vehicle to firstly understand the human-driven vehicles' behaviors and intentions. However, the human intentions or behaviors are extremely hard to model. Most prior work does not consider social behavior and human-driver behaviors, especially for rule-based algorithms and optimization algorithms. Despite the fact that some probabilistic algorithms (PoMDP framework) do consider interactions, they may require massive computational power and do not guarantee real-time performance. However, an immediate understanding/estimate of surrounding vehicles is essential for social behavior in autonomous vehicles. To address the social behavioral problems, the proposed framework contains three stages for different levels of complexity in behavioral planning. Results from each stage can be directly used for the next-level planners or executive module.

- **Discrete Intention Prediction** provides an abstract behavioral estimation, such as yield or not, lane change or not. This task is accomplished by a *probabilistic graphical model.* This model is illustrated in Section 3.1.

- **Important Points Prediction** provides references for interactive planning, such as lane change start/end points. The task is based on a *non-parametric regression*. The model is introduced in Section 3.2.

- **Trajectory Prediction** provides an ultimate estimation of surrounding cars' future trajectories over a given horizon, which is based on a *neural process*. The model is discussed in Section 3.3.

All of the stages analyze future movements of surrounding traffic, ranging from intention to full trajectory predictions. The proposed method ultimately focuses on generating a future trajectory estimate. Since there is no assumption on the probabilistic distribution model for human behaviors, a non-parametric description is more powerful to describe human behaviors in response to other traffic participants. In addition, the estimation keeps updating itself by considering the last result and environmental changes.

3.1 Discrete Intention Prediction

In this section, a model that provides discrete behavioral estimation is discussed. At this stage, the estimator selects the most likely behavior from a discrete set of options. The discrete behavioral prediction uses a Probabilistic Graphical Model (PGM), shown in Figure 3.1, to organize the relationship between the observations of the environment and the intentions.

3.1.1 The Naïve Behavioral PGM

The PGM organizes the observation with discrete intentions, as illustrated in Figure 3.1. By collecting sufficiently long historical trajectories of other vehicles, the PGM can evaluate the intention of the target vehicle based on a set of training data. Intuitively, human drivers estimate intentions of other cars by their current and immediately previous state and by considering the driving environment, e.g., the velocity of surrounding cars and positions. Our method simulates this process to achieve human-like social behavior. The most important part of our method is

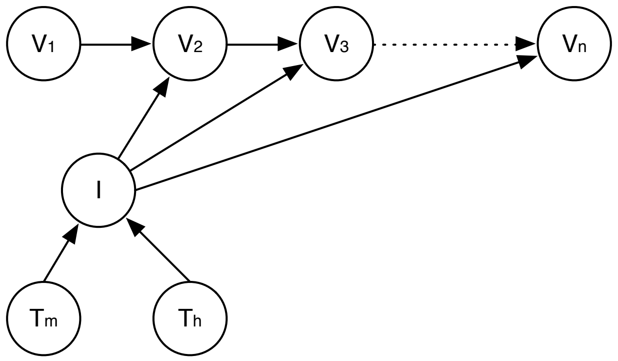


Figure 3.1: The PGM model for the binary merging intention estimation. V_is are speed nodes, T_m and T_h are the time-to-arrival to the merging point for the merging and host vehicle respectively, I is the intention to be estimated.

understanding the cause-effect relationship among previous states and intention. To simplify and abstract this dependency, we apply a probabilistic graphical model. There are three kinds of nodes in the model: (1) time nodes (T_h, T_m), which are the time-to-arrival to the merging point for the host and merging car respectively; (2) an intention node (I), which is either "Yield" or "Not Yield"; (3) speed nodes ($V_1, ..., V_n$), which contain the speed history of the target vehicle. The model's topology describes the dependency. Intuitively, current state affects intention, thus speed changes. So we design the graphical model as shown in Figure 3.1. The task here is to estimate the intention node once the car has observed enough information (speeds and times-to-arrival). As the program runs, the speed nodes are updated by the speeds of the last n cycles and the time-to-arrival nodes are updated by the current speeds and the distance-to-merging-point for each car.

Our model assumes that human intention does not oscillate as fast as the program's update rate. Therefore, one intention node will affect the next n speed nodes. These n speed nodes keep track of the target vehicle's speed during n cycles. The time-to-arrival for each car will initially decide the intention. However, this decision is solely based on current states. To further adjust the intention estimate, more evidence is needed. The speeds in the last n cycles can provide movement information of the car, and thus refine the intention estimate. Physically, given intention, those speeds form a Markov Chain, which means that every speed node is affected only by its parent node (the vehicle's previous speed).

Evaluation of PGM

We are interested in the following probability of the merging car's yielding or not yielding to the autonomous car: $P(I|\mathbf{V}, T_m, T_h)$, where $\mathbf{V}$ denotes a vector of speed during n cycles: $\mathbf{V} = [V_1, V_2, ..., V_n]$ and T_m, T_h are the time-to-arrival for the merging and host cars, respectively. I denotes the estimated intention of the target vehicle.

$$\begin{aligned} P(I|\mathbf{V}, T_m, T_h) =& \frac{P(I, \mathbf{V}, T_m, T_h)}{P(\mathbf{V}, T_m, T_h)} \\ &\propto P(\mathbf{V}, T_m, T_h|I)P(I) \end{aligned} \tag{3.1}$$

Equation 3.1 is based on Bayes' Rule, and we focus on the likelihood term $P(V, T_m, T_h|I)$. From the graphical model, it is assumed that $\mathbf{V}$ and T_m, T_h are conditionally independent, given intention. Therefore this term can be further separated into two parts:

$$P(\mathbf{V}, T_m, T_h|I) = P(\mathbf{V}|I)P(T_m, T_h|I) \tag{3.2}$$

The first term is the speed term, and the second one is the time term. Besides these two terms, we also rely on the prior information $P(I)$ in Equation 3.1.

Speed term

From the graphical model, the merging speed has the Markov Property given intention. Thus the speed term can be further simplified:

$$\begin{aligned} P(\mathbf{V}|I) &= P(V_1, V_2, ..., V_n|I) \\ &= P(V_1|I)P(V_2|V_1, I)...P(V_n|V_{n-1}, I) \end{aligned} \tag{3.3}$$

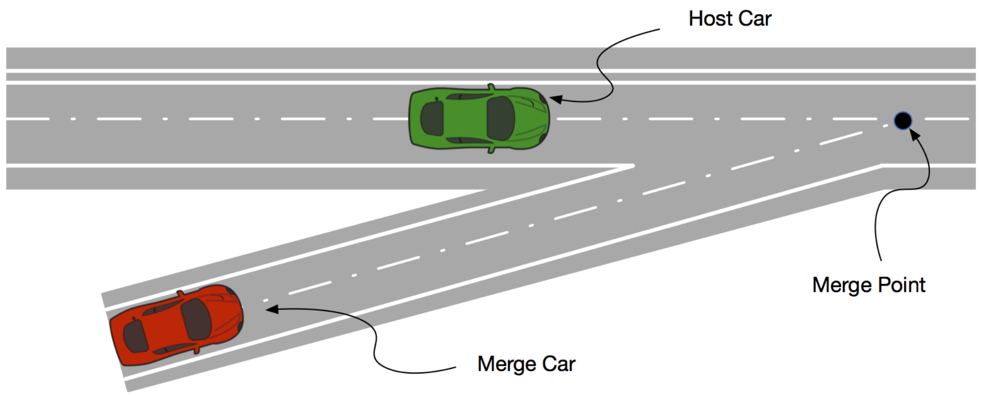


Figure 3.2: Merging vehicle following behind but accelerating. The host (green) vehicle is on the main road; the merging vehicle (red) is on the on-ramp. In fact, the host vehicle can slightly accelerate to avoid ambiguities and collisions. However, the iPCB model will slow down the host vehicle regardless of the distance from the merging vehicle to the host and the merging car's current speed. The PGM model sends proper commands by integrating speed and distance information.

Here we assume V_1, I are independent, thus $P(V_1|I) = P(V_1)$. Since there is no preference for V_1, it can be assumed to have a uniform distribution, namely $P(V_1) = \alpha$. To prevent underflow, log-likelihood is used:

$$\log P(\mathbf{V}|I) = \alpha \sum_{i=2}^{n} \log P(V_i|V_{i-1}, I) \tag{3.4}$$

There are only two intentions to be considered: yield and not yield, i.e., $P(\mathbf{V}|I = Y)$ and $P(\mathbf{V}|I = N)$. The probability distribution **P** will be learned directly from training data.

Time term

The time term implicitly contains two kinds of information: 1) current speed; 2) distance to the merge point. The time term describes how soon the cars will reach the merging point given current speed and distance. Values in the time term determine the intention of the merging vehicle. The time term will also reduce ambiguity in the speed term. In the earlier iPCB algorithm [42], only instantaneous acceleration is considered. Figure 3.2 shows a failure scenario for iPCB that results from solely considering the current acceleration. This is a non-trivial problem which contributes the majority of the failure cases in the iPCB algorithm, as shown in Table 4.2. However, in our proposed model, we additionally consider the time term, avoiding the ambiguity described above. The time term $P(T_m, T_h|I)$ is a joint conditional probability, where $T_m,\ T_h$ denote the time-to-arrival of the merging and host car, respectively. Time-to-arrival is defined as the current distance to the merging point divided by the current speed. To prevent underflow, we instead use $\log P(T_m, T_h|I)$.

Prior Term

In Equation 3.1, the last term $P(I)$ is the prior distribution for the merging vehicle intention, i.e., the percentage of merging vehicles that yield ($P(I = Y) = \gamma$) or do not yield ($P(I = N) = 1 - \gamma$). This prior term gives an initial statistical estimate of intentions. To emphasis the effects from speeds and time-to-arrivals, we set $\gamma = 0.5$, which does not affect the prediction. However, the γ does affect the estimation if there is enough and convincing prior probability for the intention. Normally, merging car is more likely to yield, i.e., $\gamma > 0.5$. One possible way to adopt the PGM for host-on-ramp scenario is to adjust the prior γ accordingly.

3.1.2 Intention Estimation Procedure

The final step is to combine the speed term and time term. Equations 3.1 and 3.2 yield:

$$\begin{aligned}\log P(I|\mathbf{V}, T_m, T_h) &\propto \log P(\mathbf{V}, T_m, T_h|I)P(I) \\ &= \log P(\mathbf{V}|I)P(T_m, T_h|I)P(I) \\ &= \underbrace{\alpha \sum_{i=2}^{n} \log P(V_i|V_{i-1}, I) +}_{\textit{Speed Term}} \\ &\quad \underbrace{\log P(T_m, T_h|I)}_{\textit{Time Term}} + \\ &\quad \underbrace{\log P(I)}_{\textit{Prior Term}}\end{aligned} \tag{3.5}$$

The estimated intention is:

$$I^* = \arg\max_I \log P(I|\mathbf{V}, T_m, T_h) \tag{3.6}$$

I^* is either "Yield" or "Not Yield". If "Not Yield", the merging car will be set as the target for the distance keeping model.

Algorithm 1 shows the full procedure for the intention estimation using the proposed PGM. We use a fixed-length queue Q to keep track of historical speed data. A new speed will be pushed to the end of the queue Q. If the size of Q is greater than length n, then the first element will be popped. This data structure ensures the program observes at most the last n cycles. The required state information S includes: current speeds and distances to the merging points of both cars. The current speeds and distances are used to calculate the time term in Equation 3.5. Lines 6 and 7 calculate the probability of the Yield $P(Y)$ or Not Yield $P(N)$ intention of the other car, by using the method introduced in Section 3.1.1. If $P(Y) > P(N)$, the host vehicle accelerates and tries to reach the merging point earlier than the merging car.

Algorithm 1 PGM-based intention estimation.

```
1:  procedure INITIALIZATION(n)
2:      $Q \leftarrow$ Queue with fixed-sized (n).
3:  procedure EVALUATE PGM
4:      while New State $S$ Comes in do
5:          Push new speed to $Q$.
6:          $P(Y) \leftarrow calculatePGM_Y(Q, S)$.
7:          $P(N) \leftarrow calculatePGM_N(Q, S)$.
8:          if $P(Y) > P(N)$ then
9:              Command $\leftarrow$ Controller(Accelerate,S)
10:         else
11:             Command $\leftarrow$ Controller(Decelerate,S)
```

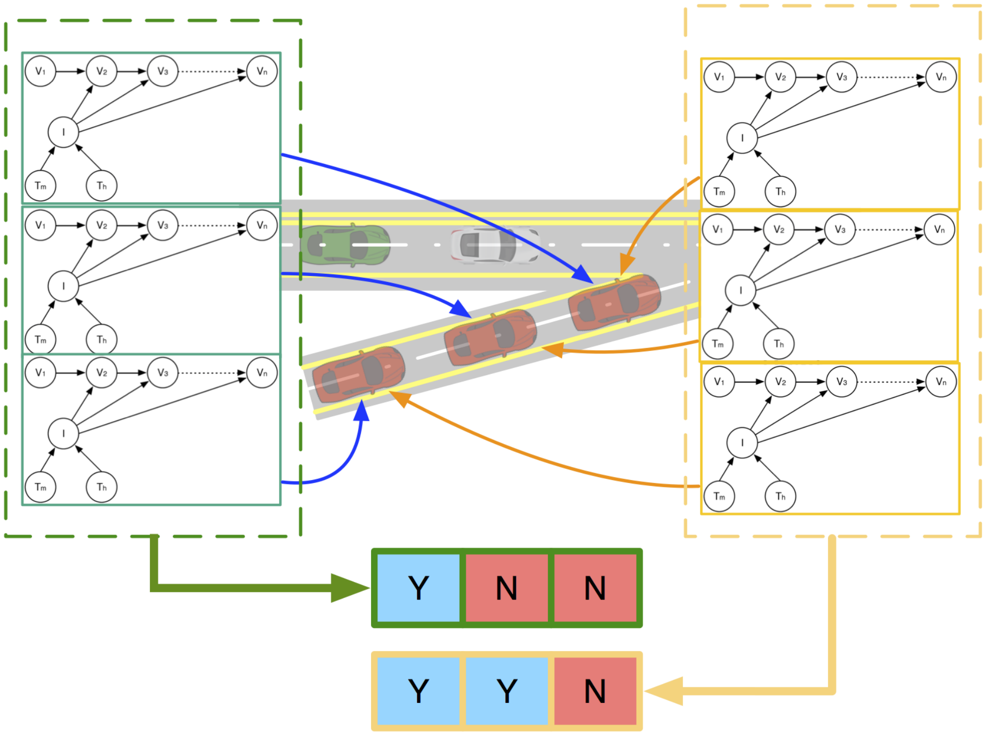


Figure 3.3: Multi-Merging PGM with a leading car. The green car is the host vehicle, and the gray one is the leading car. Both the host and leading car run on the main road. Red cars are merging vehicles on the ramp.

3.1.3 The Naïve Behavioral PGM in a Complex Setup

The 1-on-1 model handles one merging vehicle w.r.t. a host vehicle. In reality (also in our dataset), there is often more than one merging vehicle on the ramp, so the 1-on-1 model is insufficient. Moreover, in the main lane, there is often a leading vehicle running closely in front of the host vehicle. Obviously, while reacting to the merging vehicle, we need to be cognizant of the leading vehicle by keeping a safe distance. The following sections build on the 1-on-1 model to create a Multi-Merging Leading PGM to handle multiple merging vehicle and leading vehicle scenarios. In addition, a rule is introduced to react based on estimated intentions w.r.t. the host and the leading vehicle.

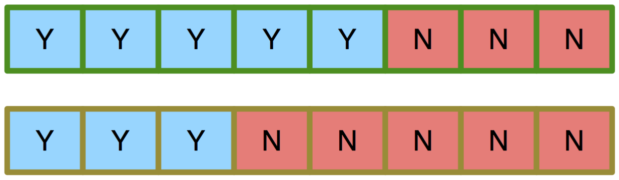
The Multi-Merging PGM method has two significant modifications compared with the single PGM model:

1 The single PGM model is duplicated and applied to each merging vehicle to generate an instant intention array.

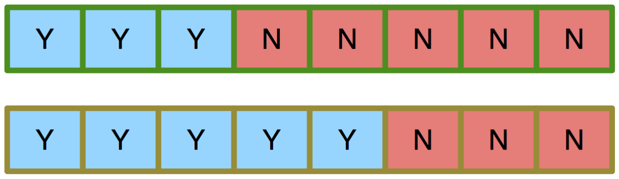
2 The process is also applied to the leading vehicle, to obtain the leading vehicle's estimate of the merging vehicles' intentions.

Figure 3.3 shows a merging example with three merging vehicles and one leading vehicle. The three PGMs on the left estimate the intentions of merging vehicles w.r.t. the host vehicle; the three PGMs on the right estimate the intentions w.r.t. the leading vehicle. At the bottom of the figure, there are two merging vehicle intention arrays: one for the host vehicle (with green-outlined boxes) and one for the leading vehicle (with yellow-outlined boxes). Each element of an array contains the estimated intention of one of the merging vehicles.

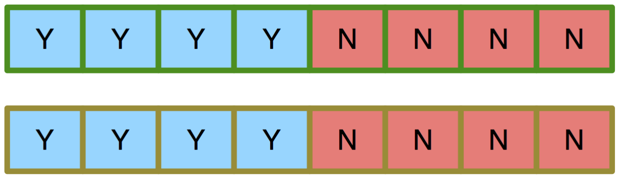
Ideally, each of the arrays contains at most one pivot. The pivot corresponds to the vehicle that divides the array of merging vehicles into two groups: a not yielding and a yielding group. Merging vehicles running ahead of the pivot (including the pivot itself) will not yield to the vehicle on the main road; those running behind the pivot will yield. According to the definition,



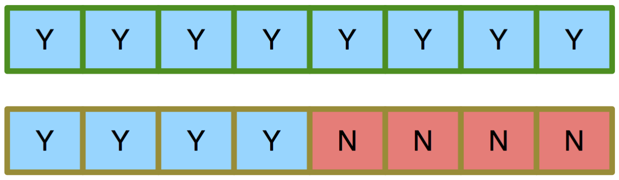
(a) $P_h > P_l$: **Host follows leading**



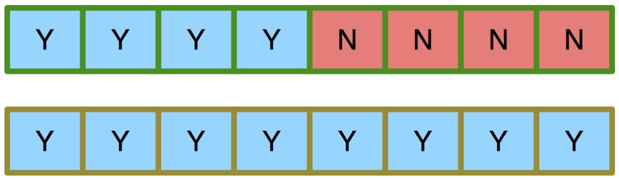
(b) $P_h < P_l$: **Host follows its own pivot**



(c) $P_h = P_l$(including null) : **Host follows leading**



(d) P_h is null and P_l is not null : **Host follows leading**



(e) P_l is null and P_h is not null : **Host Follows its own pivot**

Figure 3.4: Illustration of the pivot rules, with examples of pivot cases and corresponding rules. Numbers of cars and positions of pivot dynamically change according to real situations. The upper array is for the merging intentions w.r.t. the host car; and the lower one is for the merging intentions w.r.t. the leading car.

if no vehicle yields, the pivot vehicle will be the last one in the merging group; if all vehicles yield, the pivot is "NULL". Since the pivot is the last vehicle that does not yield, the host / leading vehicle should follow the pivot by using an aggressive distance keeping model, without considering other merging vehicles. (Because the merging vehicles which run behind the pivot are identified to yield to the vehicle on the main road, the merging vehicle behind the pivot will naturally keep a reasonable distance between itself and the main-road vehicle.) After identifying pivots for both the leading vehicle and the host vehicle, a deterministic rule-based planner is proposed to activate different behaviors for the host vehicle. The rule-based planner is described in the next section.

3.1.4 Pivot rules

Note that the host vehicle is always behind the leading vehicle, and the merging vehicles behind the pivot must yield to their corresponding main-road vehicle. We use P_h, P_l to denote the pivot merging car for the host car and the leading car, respectively. Then $P_h = P_l$ if the host car and the leading car have the same pivot; $P_h < P_l$ if the pivot for the host car runs behind the leading car's; $P_h > P_l$ if the pivot for the host car runs ahead of the leading car's. The high level description of the rule is that: if $P_h \geq P_l$, the host follows its leading; otherwise, host follows it own pivot. Detailed deterministic cases and corresponding rules are shown in Figure 3.4. In each subfigure, the upper array (green boxes) is the merging intentions *w.r.t.* the host vehicle; and the lower array (yellow boxes) is the merging intentions *w.r.t* the leading vehicle. In each array, each cell corresponds to a merging car. The number of merging cars and cells may change. A blue cell means *Yield*, and a red cell means *Not Yield*. The leftmost red cell corresponds to the pivot merging car.

3.1.5 The Smoothed Behavioral PGM with Hidden States

The smoothed behavioral PGM, shown in Figure 3.5, considers the restriction and uncertainty in sensors. Since RADAR can only give range and range rate (relative position and speed with respect to the host vehicle), speed and acceleration measurements can contain large error. The smooth PGM utilizes position observations and adds them as an extra layer in the Naïve Behavioral PGM. Using a RTS-smoother, the array of speeds can be derived from a sequence of position measurements. Compared with merely using a Naïve Behavioral PGM, the smoothed version requires fewer observations and achieves better performance. In the graphical model (Figure 3.5), only location nodes O_i are observable. The observation is the relative location of the target with respect to the end of the merging ramp; I is the intention node; latent node X_i is the state of the merging car, which is expected to have an optimal estimation. It consists of nodes S, V: S_i are the locations to be estimated; V_i nodes are speeds after smoothing. Compared with the models in the sections above, this improved model does not include Time-to-Arrival nodes, since their information can be derived from location and speed nodes.

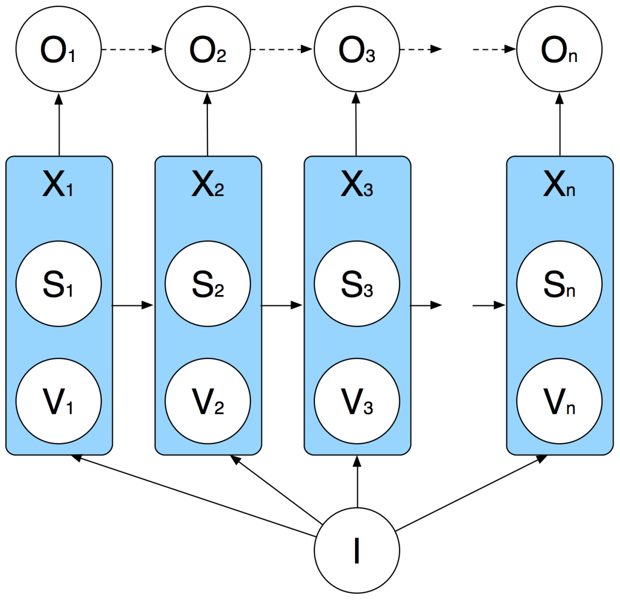


Figure 3.5: The structure of the smoothed PGM. Shown is the graphical model of ramp merging considering position observations and intentions. O nodes are positions for the merging car relative to the end of the auxiliary lane, and are the direct output of sensors; X is the latent state estimated by smoothing and it consists of S, V nodes for positions and speeds; The I node is the intention. The number of required observations and states is determined based on experiment.

Based on the observations, the model is expected to output probability of intentions, i.e.,:

$$P(I|\mathbf{O}, \mathbf{X}) \tag{3.7}$$

where I is the intention and $\mathbf{O}, \mathbf{X}$ are arrays of observations $\{O_i\}$, and state $\{X_i\}$, which contains the estimated distances to the merging point $\{S_i\}$ and estimated speeds $\{V_i\}$. According to the structure of the graphical model, I and $\mathbf{O}$ are conditionally independent given $\mathbf{X}$. Therefore, (3.7) yields the same probability which is stated in [21]:

$$P(I|\mathbf{X}) \tag{3.8}$$

From the graphical model, the state X_i has the Markov Property given intention. Thus (3.8) can be further simplified by assuming $\mathbf{X}$ to be multi-variable and uniformly distributed:

$$\begin{aligned} P(I|\mathbf{X}) &\propto P(\mathbf{X}|I)P(I) \\ P(\mathbf{X}|I) &= P(X_1, X_2, ..., X_n|I) \\ &= P(X_1|I)P(X_2|X_1, I)...P(X_n|X_{n-1}, I) \end{aligned} \tag{3.9}$$

Since the initial state X_1 can vary regardless of the intention, here we assume X_1, I are independent, thus $P(X_1|I) = P(X_1)$. Since there is no preference for X_1, it can be assumed to have a uniform distribution, namely $P(X_1) = \alpha$. To prevent underflow, log-likelihood is used:

$$\log P(\mathbf{X}|I) = \alpha \sum_{i=2}^{n} \log P(X_i|X_{i-1}, I) \tag{3.10}$$

The estimated intention is:

$$I^* = \arg\max_{I} \ \log P(I|\mathbf{X}) \tag{3.11}$$

There are only two intentions to be considered: yield and not yield, i.e., $P(\mathbf{X}|I = Y)$ and $P(\mathbf{X}|I = N)$. The problem remains finding the best estimate of $\mathbf{X}$.

Instead of measuring the speeds, the model relies on location measurements, and then derives speeds using filtering and smoothing. As typically the locations cannot be perfectly measured from sensors, it is necessary to estimate their true values. Since the model collects a certain number of location observations, all of these data can be utilized to smooth the estimation of true values, i.e., estimating $X_i = (S_i, V_i), 1 \leq i \leq n$ using a series of observations O_i.

$$P(X_i|O_{1:n}) \tag{3.12}$$

According to [43, 44] the predicted distribution at time step *i+1* by adapting Bayesian optimal smoothing is:

$$P(X_{i+1}|O_{1:i}) = \sum_{X_i} P(X_{i+1}|X_i)P(X_i|O_{1:i}) \tag{3.13}$$

And the smoothing estimation of the state at any time step $i < n$ is:

$$P(X_i|O_{1:n}) = P(X_i|O_{1:i}) \sum_{X_i} \frac{P(X_{i+1}|X_i)P(X_{i+1}|O_{1:n})}{P(X_{i+1}|O_{1:i})} \tag{3.14}$$

With a proper edge condition, i.e., $P(X_n|O_{1:n})$, which will be discussed in the following paragraphs, every past state can be solved by iteratively evaluating (3.14).

Here, we assume the distribution of the state at any time step given that the full observation is a Normal Distribution, i.e.,

$$X_i|O_{1:n} \sim N(X_i|\mathbf{M}, \mathbf{P}) \tag{3.15}$$

Then the Bayesian smoothing equations in (3.13) and (3.14) become a *Rauch-Tung-Striebel smoother[44]* (RTS smoother, which is also called the Kalman smoother). The speed part of the maximum *a posteriori* of the state will be taken as the condition of the intention I's estimate. The RTS smoother consists of two steps, the forward prediction and the backward recursion.

The Forward Prediction

The Forward Prediction step uses a Kalman Filter to update the mean M_i and the covariance P_i of the distribution, as well as the Kalman Gain G_i:

$$\begin{aligned} M_{i+1}^{-} &= AM_i, \\ P_{i+1}^{-} &= AP_{i+1}A^T + Q, \\ G_i &= P_i A^T [P_{i+1}^{-}]^{-1} \end{aligned} \tag{3.16}$$

where A is the motion model of the Kalman Filter, which will be specified in the following section. Q is the covariance for the process noise,

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} \frac{q\Delta t^3}{3} & \frac{q\Delta t^2}{2} \\ \frac{q\Delta t^2}{2} & q\Delta t \end{bmatrix}$$

The Backward Recursion

$$\begin{aligned} \hat{M}_i &= M_k + G_k[\hat{M}_{i+1} - M_{i+1}^{-}], \\ \hat{P}_i &= P_i + G_k[\hat{P}_{i+1} - P_{i+1}^{-}]G_k^T \end{aligned} \tag{3.17}$$

where the $\hat{\cdot}$ indicates the smoothed variables. The backward equations start from the last step, where $\hat{M}_n = M_n, \hat{P}_n = P_n$. Note that the result $\hat{\mathbf{M}}$ contains all optimal states, which is an array of location and speed tuples, $X = (S, V)$.

The estimation procedures

In the application, the model is used as a unit which can be executed in every cycle. As shown in Figure 3.6, the upper-level nodes are the state nodes, which contains the observation O and the smoothed optimal estimation X shown in Figure 3.5. The lower-level nodes are the intentions over the whole merging period, which will be estimated. Each intention node in the lower level links with n consecutive upper-level state nodes. n is the number of observation in each

Algorithm 2 smoothPGM intention estimation.

```
1: procedure INITIALIZATION(n)
2:     $Q \leftarrow$ Queue with fixed-sized (n).
3: procedure SMOOTHING($Q, O_{new}$)
4:     Push new Observation $O_{new}$ to $Q$.
5:     $(M^{-}, P^{-}) \leftarrow$ predictForward($Q$)
6:     $(\hat{M}, \hat{P}) \leftarrow$ recurseBackward($M^{-}, P^{-}$)
7:     $\mathbf{V} \leftarrow H \cdot \hat{\mathbf{M}}$
8: procedure EVALUATE PGM($\mathbf{V}$)
9:     $P(Y) \leftarrow$ calculatePGM$_Y$ ($\mathbf{V}$).
10:    $P(N) \leftarrow$ calculatePGM$_N$ ($\mathbf{V}$).
11:    if $P(Y) > P(N)$ then
12:        Command $\leftarrow$ Controller(Accelerate)
13:    else
14:        Command $\leftarrow$ Controller(Decelerate)
```

estimation cycle. Here, $n = 4$ is for the sake of illustration. Each pair of intention node and its corresponding n state nodes is a smoothedPGM unit, as in Figure 3.5.

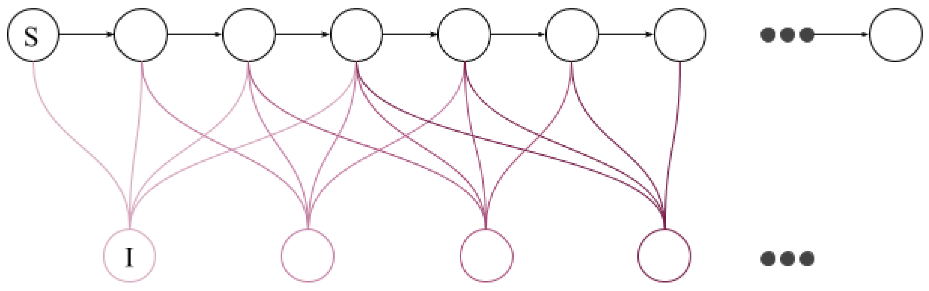


Figure 3.6: Use the PGM model as a unit over time in each intention estimation cycle.

The whole estimation procedure of the smoothPGM method is shown in Algorithm 2. Note that compared with Algorithm 1, the main improvement is the "SMOOTHING(Q, O_{new})" function from line 3 to line 7. This function implements the forward prediction step and backward recursion step. Then it outputs an optimal estimation of the hidden states. The H in line 7 is a variable selection matrix. Since lines 10 and 9 only require an array of velocities with dimension of $1 \times n$, and $\hat{\mathbf{M}}$ contains both location and velocity with dimension of $2 \times n$, H is a 1×2 matrix that helps in selecting the corresponding row of $\hat{\mathbf{M}}$. In the implementation, the second row of $\hat{\mathbf{M}}$ contains smoothed velocities. Therefore, with $H = [0, 1]$, an array of velocity $\mathbf{V}$ is selected. The array is then sent to the evaluation procedure.

3.1.6 Example Scenarios for Discrete Intention Prediction

There are scenarios which do not require detailed trajectory prediction. Trajectories in these scenarios are simple or deterministic. For example, in an ambiguous four-way stop scenario, the ego-vehicle does not need to predict the trajectory of other vehicles; instead, an abstract decision of “go” or “wait” is sufficient. In ramp merging control, cars on the main road only run along their lane, and merging cars only follow the ramp. With a known path, the interaction between the ego-vehicle and merging vehicles becomes the problem of whether the ego-vehicle should overtake the merging car or yield to it. A discrete behavioral prediction works with an off-the-shelf module and helps the ego-vehicle perform cooperatively. Based on results of the discrete behavioral prediction, it directly executes separate modules in different scenarios to react to the behavioral planning. When the ego-vehicle decides to yield to the merging car, it will initiate the Adaptive Cruise Control and set the following target as the merging car. Otherwise, the ego-vehicle keeps the original dynamics and overtakes the merging car.

3.2 Important Waypoints Prediction

This part of the behavioral estimation bridges the gap between higher-level behavior commands and the trajectory planner. There are two challenges in the task: 1) Analyzing the surrounding vehicles’ mutual effects from their trajectories; 2) Estimating the proper behavioral start point and end point according to the analysis of surrounding vehicles. We propose a learning-based approach to understanding surrounding traffic and making decisions to achieve cooperative behavior.

Intermediate important points prediction is suitable for scenarios such as lane change. As shown in Figure 3.7, the red car (Veh-s) is the autonomous car (host car), which received a left-

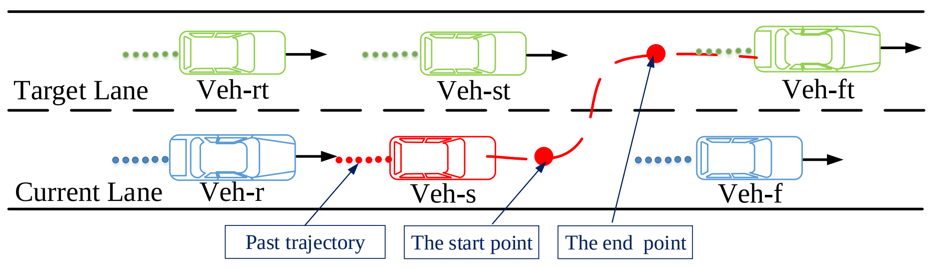


Figure 3.7: A left lane change scenario.

lane-change command. The blue cars are the leading car (Veh-f) and the following car (Veh-r) in the current lane. The vehicles in the target lane are also considered: the immediate left car (Veh-st), its leading car (Veh-ft) and its following car (Veh-rt). The method will generate a pair of start/end points for the left-lane-change behavior. The red dashed line just indicates the lane-change behavior, not necessarily as a planned path.

3.2.1 Reproducing Kernel Hilbert Space

The reproducing Kernel Hilbert Space (RKHS) representation for planning was introduced by Marinho et al. [45]. In this work, a trajectory is explicitly described by Gaussian radial basis functions, using a functional gradient to optimize a cost functional to avoid static obstacles or navigate a high-dimensional manipulator. However, they did not explore the opportunity to apply similar methods to dynamic environments and cooperative scenarios. We follow their formulation, but instead of using a functional gradient to find the optimal solution, our proposed method relies on RKHS non-parametric regression and a prior dataset to estimate continuous values, i.e., the start/end points.

Reproducing Kernel Hilbert Space **H** contains families of smooth functions which are defined by a Mercer Kernel. The Mercer Kernel is a continuous mapping $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, i.e., $K(x, y) =< f, g >_H$, where $f := K_x, g := K_y,\ f, g \in$ **H**. A function f in **H** can be represented by a linear combination of the kernel: $f(\cdot) = \sum \alpha_i K_{x_i}(\cdot)$, and this kernel has the reproducing property: $f(x) =< f, K_x >$, which is essential to RKHS. With the help of the kernel, it is possible to evaluate the function without explicit definition of the function (or basis functions) in the high-dimensional functional space [46]. A behavior generator is a function $\mathbf{F} : \mathcal{X} \to \mathcal{B}$, which maps a vector of trajectories ($\mathbf{X} \in \mathcal{X}$) to a behavior ($b \in \mathcal{B}$). $\mathcal{X}$ is a coordinate space which contains vectors of N surrounding vehicles' past trajectories $\mathbf{X} \overset{\text{def}}{=} \{x_i\}_1^N,\ x_i \in \mathrm{I\!R}^{\mathrm{T}}$. T is the length of the relevant historical poses. The input contributes to all elements in the output vector. Using $\mathbf{\Gamma} = \{X\}_i^N$ as the training set and X_i as a training sample, then $\mathbf{F}(\mathbf{\Gamma}) = \{[f_1(X_1), ..., f_D(X_1)], ..., [f_1(X_N), ..., f_D(X_N)]\}$. The output range $\mathcal{B} \subseteq \mathrm{I\!R}^{\mathrm{D}}$ represents the behavior. In the lane-changing problem, we are interested in two points: the start and the end points of the lane-changing behavior. Thus $D = 2$ in the current setup. Since the dimension of the range (the output domain) is $D > 1$, this function is a vector-valued function. The kernel $K(\cdot, \cdot)$ which is mentioned in the paragraph above is no longer a scalar-valued function but a matrix-valued one, i.e., $\mathcal{X} \times \mathcal{X} \to \mathrm{I\!R}^{\mathrm{D}\times\mathrm{D}}$,

$$K(X, U) = \begin{bmatrix} k(X,U)_{1,1} & \cdots & k(X,U)_{1,D} \\ k(X,U)_{2,1} & \cdots & k(X,U)_{2,D} \\ \vdots & \cdots & \vdots \\ k(X,U)_{D,1} & \cdots & k(X,U)_{D,D} \end{bmatrix} \tag{3.18}$$

Parallel to the scalar-valued kernel, the matrix-valued kernel has the reproducing property which is also given by the Representer Theorem [47, 48] :

$$F(X) = \sum_{i=1}^{N} K(X_i, X) \cdot \boldsymbol{\alpha}_j,\ \ \boldsymbol{\alpha}_j \in \mathrm{I\!R}^{\mathrm{D}} \tag{3.19}$$

The $\cdot$ operator is the normal inner product in Euclidean Space, and $\boldsymbol{\alpha}$ is a ND-dimensional coefficient. By constraining the behavior function $\mathbf{F}$ into the RKHS, we assume that $\mathbf{F}$ is continuous and can be represented by a linear combination of a set of basis functions. The functions are unknown to us, and we are not interested in the exact form of the behavior generator function; instead, we are interested in its evaluation given trajectories. In order to approximate the evaluation, and since we do not explicitly know the form of the function, we use non-parametric regression from the data in the RKHS, which is defined by the kernel above.

3.2.2 Important points estimation as a regression in RKHS

The method for the intermediate level of prediction has the following three features:

- Uses the dataset to generalize surrounding vehicles' effects on the autonomous vehicle's cooperative behaviors from the dataset.
- Surrounding vehicles' and the autonomous vehicle's past trajectories are applied as input to the method, thus historical information is also used.
- Formulates the behavior generator as a function in Reproducing Kernel Hilbert Space, and evaluates the start and end points from an input by a non-parametric regression (RKHS estimator). Therefore, no fitting model is assumed.

As the kernel representation of the behavior generator function was defined above, the function should be estimated from data and properly evaluated at a given input. Note that we do not explicitly define the form of the function; instead, we use a linear combination of kernels, as described in Equation 3.19. Once the kernel is decided (often given by users or separately learned from data), the only parameter left to be optimized is the coefficient $\boldsymbol{\alpha}$. Thus the approximation results in minimizing the regularized empirical error:

$$\hat{f} = \arg\min_{f \in H} \sum_{i=1}^{N} (b_i - f(X_i))^2 + \lambda J(f) \tag{3.20}$$

where (X_i, b_i) is training input and behavior output, and $J(f)$ is the penalty term. Here $||f||_H$ is used as the penalty term (or the regulation term). The coefficient has a closed-form solution [47, 48]:

$$\boldsymbol{\alpha} = (K(\mathbf{X}, \mathbf{X}) + \lambda N \mathbf{I})^{-1} \mathbf{b} \tag{3.21}$$

Substituting the evaluation from Equation 3.21 into Equation 3.19 yields the estimated behavior generator function $\hat{f}$. Given a new input X', the estimated behavior $\hat{b}$ becomes:

$$\hat{b} = K^*(K + \lambda I)^{-1}\mathbf{b} \tag{3.22}$$

Where $\mathbf{b} \overset{\text{def}}{=} \{b_i\}_1^N$ is the collection of the training behaviors. K^* is the new kernel result given incoming input, a $D \times ND$ matrix. The regularization factor λ leverages the smoothness and accuracy of the regression function. Note that the $(K + \lambda I)^{-1}\mathbf{b}$ part can be pre-calculated offline given the training samples. Once an input comes in, only the K^* will be re-evaluated, and matrix multiplication is performed with the pre-calculated $(K + \lambda I)^{-1}\mathbf{b}$.

3.3 Meta Induction Program and Conditional Neural Process

The meta induction architecture takes a set of demonstration examples and an additional observation as the inputs, i.e., $\mathcal{D} = \big((X_1, Y_1), ..., (X_n, Y_n)\big)$ and $\hat{X}$, and then it outputs the corresponding estimate $\hat{Y}$. The size of D can be as small as $1 \sim 5$, i.e., $n \in [1, 5]$. In each training step, n demonstration examples as well as a new observation $\hat{X}$ are used to approximate the corresponding output $\hat{Y}$. The main difference between the Meta Induction Program and the plain deep learning approaches (e.g., the plain LSTM trajectory prediction [34]) is the demonstration part. Instead of directly looking for the mapping between the new observed input $\hat{X}$ and its corresponding output $\hat{Y}$, the demonstration part encodes a higher-level description of the prior condition of the task. More specifically, most previous methods look for $P(\hat{Y}|\hat{X})$, but the Meta Induction Program uses one more condition: $\mathcal{D}$ (i.e., the demonstration examples), which results in $P(\hat{Y}|\hat{X}, \mathcal{D})$.

The lane change problem can be considered as a set of trajectory prediction tasks that have various numbers of surrounding cars, different types of dynamic changes, etc. The demonstration $\mathcal{D}$ and the network are used to generalize tasks in the training process. During evaluations, the demonstration network keeps a short period of past trajectory observations and ”categorize” the current task from these observations (demonstrations) when evaluating new inputs. Therefore, the condition $\mathcal{D}$ can better guide the training and evaluation of the network to achieve sound performance.

Conditional Neural Process [49] is one of the latest meta-learning induction [50] methods. There are three sub-modules in the original Conditional Neural Process (CNP) , shown in Equation 3.23:

$$\begin{aligned} r_i &= h(X_i, Y_i) \\ \mathbf{r} &= \bigoplus_{i}^{n} r_i \\ \phi_i &= g(X_t, \mathbf{r}) \end{aligned} \tag{3.23}$$

where h is the demonstration network, which can be considered as an encoder for the input of demonstration examples. g is the generator network, which can be considered as a decoder. The middle line is the condition which is extracted from the historical input. Using the result from the demonstration network, it generates a set of intermediate results r_i. Along with the latest demonstration, the result $\mathbf{r}$ which is aggregated from r_i is used to generate factorized parameters for a stochastic process. Due to the demonstration and the condition framework, it can be considered as an adaptive kernel. The kernel that defines the stochastic process can change according to the recent observations. For example, if the CNP’s results are tied to a Gaussian Process (GP), its kernel, i.e., mean μ_t and variance σ_t, will be a function of the observation data, and determined by the CNP.

3.3.1 Recurrent Meta Induction Network

In the vanilla CNP, the result of each demonstration is summarized by r_i. For N demonstrations, $r_0, ...r_{n-1}$ will be obtained from the observation sub-net. The total condition which is generated from the N demonstrations is represented by the aggregation of r_i, i.e.,

$$\mathbf{r} = r_0 \bigoplus r_1.... \bigoplus r_{n-1} \tag{3.24}$$

The main purpose of the aggregation is to ensure permutation invariance, which further satisfies the requirement that CNP is a stochastic process. In most cases, the aggregation is implemented by mean, i.e., Eq. 3.24 becomes

$$\mathbf{r} = \frac{1}{n}\sum_{0}^{n-1} r_i$$

.

The permutation invariance property is theoretically sound for most static tasks, for example, function regression or image completion, whereas in applications where sequence is more important, the aggregation eliminates the sequential information. However, in most dynamic scenarios and tasks, such as trajectory prediction, the observation and result are highly serialized, and permutation invariance cannot be satisfied.

To retain the sequential information, it is intuitive to introduce recurrent neural networks into the demonstration sub-net. In Fig. 3.8, the green box indicates the demonstration sub-net, which consists of an asynchronous LSTM network. The figure shows a roll-out description of the LSTM. The dashed-arrow lines between LSTMs indicate its inner variables, which are passed inside the LSTM cell over time. The LSTM's results remain internal until it obtains enough demonstrations.

The result of the LSTM is passed through a fully connected network before being used as a condition tensor $\mathbf{r}$. Therefore, the whole process can be expressed as:

$$\begin{aligned} \mathbf{r} &= h(\mathcal{X}_n, \mathcal{Y}_n) \\ \hat{Y} &= g(\hat{X}, \mathbf{r}) \end{aligned} \tag{3.25}$$

where h is the LSTM layer and is followed by a fully connected layer. Comparing with the orig-

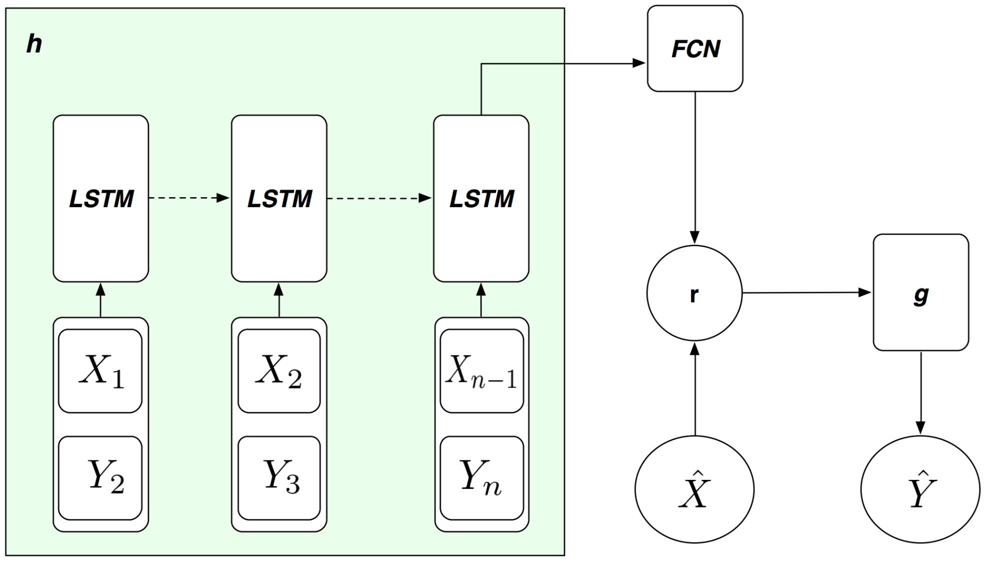


Figure 3.8: The proposed recurrent structure. The green box indicates the observer sub-net. The LSTM is an asynchronous setup, which gives outputs until obtaining enough demonstrations. This output is then passed through a fully connected network (FCN) to obtain r, the intermediate condition tensor. $\hat{X}$ is the current observation right before the prediction, $\hat{Y}$ is the expected trajectory, and g is the generator.

inal CNP's expression, shown in Equation (3.23), here the aggregation operation $\bigoplus$ is replaced by the recurrent network (which mainly retains the sequential information) and a fully connected network. $\mathcal{X}_n$ contains n segments of observed trajectories. $\mathcal{Y}_n$ contains n corresponding trajectories for the target vehicle. In detail, $X_i \in \mathcal{X}_n$ is the observation of historical trajectories from time t of all vehicles including the target and its surrounding cars; And $Y_{i+1} \in \mathcal{Y}_n$ is one segment of the trajectory of the target vehicle from time $i+1$. A pair of (X_i, Y_{i+1}) is taken as one demonstration. Note that for the input of one step for the LSTM cell, X_i and Y_{i+1} are asynchronous (X_i contains all cars' trajectories in the i-th time interval, and Y_{i+1} contains the target car's trajectory in the (i+1)-th time interval). X_{n+1} is the new input that is ready to be evaluated.

To avoid confusion, the inputs to the demonstration and generator sub-nets will be described separately. To better illustrate the idea, only lateral translation vs. timestamp plot is shown and discussed (see Figure 3.9 caption). Solid lines indicate the observed changes of lateral position over time of all related cars, i.e., one target and its surrounding cars. At time T, the goal is to predict the future trajectory of the target car, which is shown as the dark dashed line. L is the

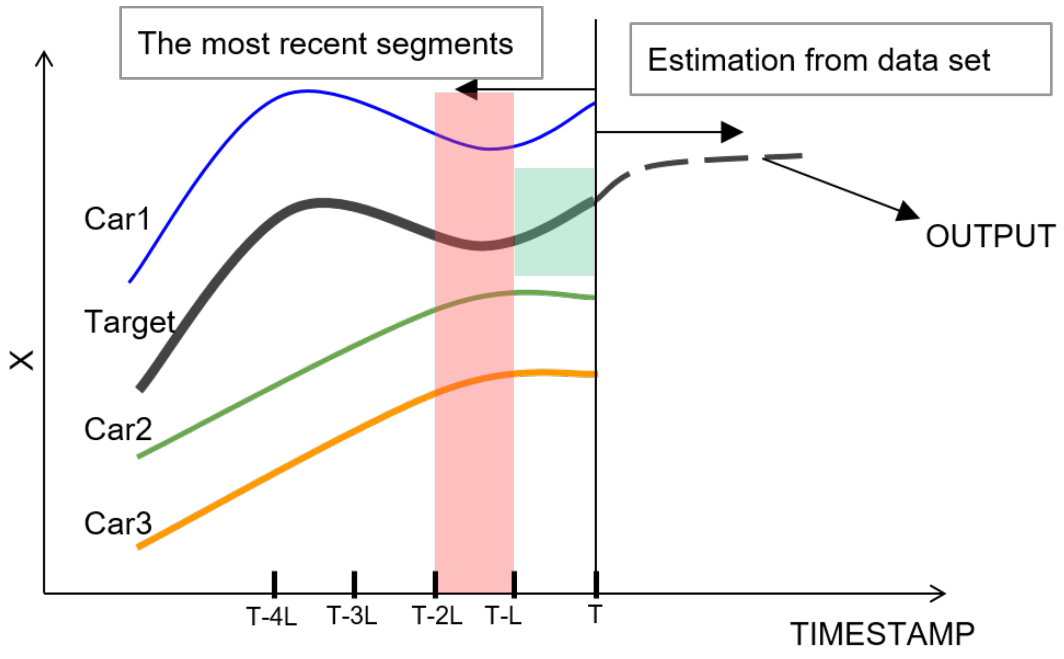


Figure 3.9: Illustration of the input and output of the network in a lane change scenario. The figure shows the idea, but does not correspond to real trajectories. The horizontal axis is the timestamp; the vertical axis is the lateral position, labeled as 'X'.

observation segment's time interval.

3.3.2 Input of the observer sub-net

The input of the demonstration network is the ordered set of examples

$$\mathcal{D} = \big((X_0, Y_1), (X_1, Y_2), ...(X_{n-1}, Y_n)\big),$$

where the input set here $\big(\cdot\big)$ retains the order of the demonstration (X_i, Y_{i+1}) for each time interval. At each timestamp, the single observation X_i is a segment of past trajectories of length L for all cars. In Fig. 3.9, X_i corresponds to the pink region. Y_{i+1} is the trajectory of the target vehicle in the next time interval of Y_i. Y_{i+1} corresponds to the green region in Fig. 3.9.

3.3.3 Input of the generator sub-net

The input of the generator sub-net has two parts: a) The condition representation, which is generated from the observer sub-net. b) The segments of all vehicles' trajectories in the time interval immediately before the time of prediction T. More specifically, this input contains all vehicles' trajectories in the time interval [T-L, T).

The main difference between the Conditional Meta induction and the plain Conditional Neural Process induction is that, instead of using an aggregator to remove the sequential effects, the proposed method uses a recurrent sub-network (LSTM) to extract conditions for the generator sub-network.

The loss function is a combination of the loss in two directions:

$$C = C_{lon} + \lambda C_{lat} \tag{3.26}$$

where C is the total loss, and C_{lon}, C_{lat} are the losses in the longitudinal and lateral directions, respectively. The main reason for this separation is that the longitudinal translation and lateral drifting are not in the same range. Empirically, the ratio is set to $\lambda = 10$ in the lane changing scenario. In detail, since we consider the sequence of the trajectory, a mean-squared-error is applied for both longitudinal and lateral losses.

3.4 Summary

This chapter introduced a three-stage behavioral estimation system for the interaction between autonomous driving cars and human-driven cars. The study focuses on two scenarios: 1) Ramp merging and 2) Lane changing. The ramp merging is a forced lane change, which has a fixed routine. Due to the nature of the ramp merging scenario, the merging cars' intentions can be categorized by two behaviors: Yield or Do not Yield. These behaviors are estimated by a probabilistic graphical model that takes both host and merging cars' moving dynamic and ramp length into consideration. The model is refined to tolerate observation uncertainties, and extended to a more complex and realistic scenario: a ramp that has multiple merging cars with a leading car. In the lane changing scenario, the intermediate points prediction implicitly models the interaction between a target car and its surrounding cars, and then outputs the cooperative lane change start and end points. A non-parametric regression model is applied here to avoid the assumption of the probability distribution of the human behavioral model.

Finally, a full-trajectory estimation method is proposed. The trajectory estimation takes all surrounding cars' past trajectories, and then outputs a future trajectory of a target in traffic (not necessarily an autonomous driving car). An induction network model is adopted here in order to use limited data and receive sound prediction results. The three-stage model leverages the computation and capability for the behavioral or trajectory estimation in various scenarios with different complexities.

In waht follows, Chapter 4 shows the individual and statistical experimental results for each of the stages in corresponding scenarios. Chapter 5 performs sensitivity analysis to studies the robustness of the estimation towards different type of sensing uncertainties.

Chapter 4

Examples and Experiments

In this chapter, experiments are designed to test the performance of the three-stage framework. The three stages are :

1 **Discrete intention estimation** [21, 22, 51], which is used in scenarios which only need to determine following targets, such as ramp merging. In these scenarios, since the merging route or the main road has a fixed geometry, every car should follow the determined route, and it is sufficient for an ego-vehicle to select a correct target to follow. Therefore, there is no need to spend time and computational power to make a full trajectory for a target vehicle in the merging scenario. In ramp merging, the main target for the algorithm is to estimate discrete high-level intentions of human-driven vehicles on the ramp.

2 **Semi-continuous points estimation** [52], which is used in more complex scenarios, such as lane change. The semi-continuous points estimation provides critical points prediction to aid the path planning. For example, in the lane change scenario, it is important for the ego-vehicle to determine the locations where the car starts and ends the lane change behavior.

3 To further provide more details, a **full trajectory predictor** [53] is established. Combined with the prediction, a path planner can generate a cooperative trajectory. A cooperative

trajectory is more critical for interactive scenarios such as lane changing. A sound full trajectory prediction is the foundation for the generation of a cooperative plan in such a scenario.

In the discrete intention estimation, low collision rates, large safety distances, and small K-L divergence are expected to be observed as the results from a series of the proposed probabilistic algorithms. In addition, in semi-continuous and continuous trajectory estimation, the proposed framework is expected to have limited error and standard deviation when compared with ground-truth. The methods are trained and tested mainly in NGSIM datasets which include scenarios such as ramp merging and lane changes. Individual vehicle trajectories are extracted from NGSIM, and details and statistics of the trajectories are shown in Table 4.1. These trajectory data are so far unique in the history of traffic research and provide a great and valuable basis for the validation and calibration of microscopic traffic models. The methods are tested on the datasets from the I80 and the US101 highways. The I80 dataset consists of three 15-minute periods: 4:00 p.m. to 4:15 p.m., 5:00 p.m. to 5:15 p.m., and 5:15 p.m. to 5:30 p.m. These periods represent the buildup of congestion, or the transition between uncongested and congested conditions, and full congestion during the peak period. A total of 45 minutes of data are available in the US101 dataset, which is segmented into three 15-minute periods: 7:50 a.m. to 8:05 a.m., 8:05 a.m. to 8:20 a.m., and 8:20 a.m. to 8:35 a.m. In both the I80 and the US101 datasets, vehicle trajectory data provide precise location of each vehicle within the study area every one-tenth of a second.

The road geometry is shown in Figure 4.1. 806 pairs of merging and 543 cases of lane-change scenarios are extracted from the dataset. The dataset contains a trajectory for every vehicle, with information such as position, speed and acceleration.

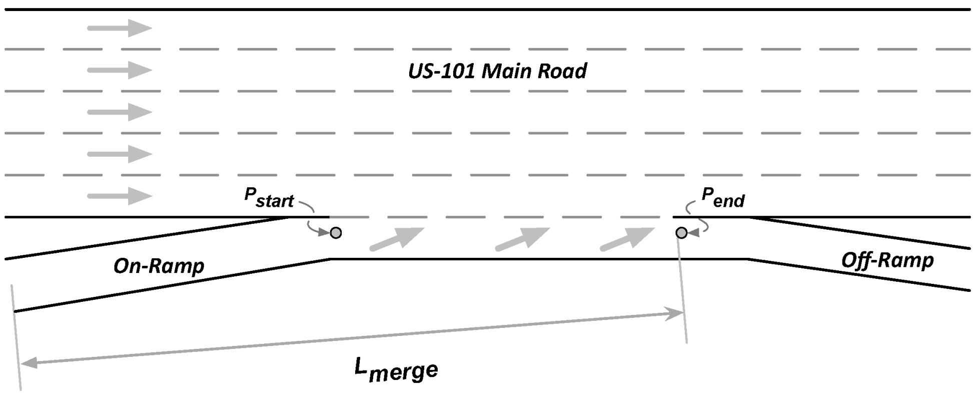


Figure 4.1: The studied ramp section on US101 in NGSIM dataset.

Table 4.1: Features of the US-101 and I-80 datasets

Dataset	Ramp Length m	SMS [54] m/s (mph)	Num. of Pairs.	Lane-Change
US-101	640	12.4 (27.7)	354	315
I-80	503	14.2 (31.3)	452	228

4.1 Experiments Results for Discrete Behavioral Prediction

4.1.1 Tests in Single-Host Single-Merging Scenario

The tests for discrete behavioral prediction contain two sets of experiments in simulation: 1) reacting to merging vehicles with real-data trajectories which are extracted from datasets; and 2) reacting to merging vehicles which use a manually designed motion strategy. The second set of tests is to evaluate the generality of our method with respect to differing merging car strategies and a broader range of initial conditions (speeds and relative location). It should be emphasized that even though the strategy of the merging vehicle is programmed in the second experiment, the host car does not know the strategy or true intention of the merging car. All the host car can do is observe the state of the merging car and estimate its intention by using the proposed models: Naive PGM and Smoothed PGM. We compare our new algorithm with the following methods:

1. *ACC merging*, a non-cooperative method that distance-keeps to the merging car if it is closer to the merging point;

2. *Slot checking*, which is adopted from the Urban Challenge [8].

3. *iPCB*, which is proposed in [42].

4. *PGM-G*, which uses the proposed PGM structure, but assumes a Gaussian Distribution for the speed transition probability, like iPCB [42].

5. *PGM*, which is proposed in [21].

To evaluate the performance, three criteria are examined:

1 **Collision Rate:** the ratio of the failure numbers over the total number of testing cases;

2 **Safety Distance:** the distance between the ego- and merging vehicle when the first of them passes the merging point (or the merging vehicle enters the main road in auxiliary-lane scenarios);

3 **Update Rate:** The frequency or period of time to analyze a vehicle's intention.

Table 4.2: Statistical results for different methods

Method	US-101 Data		I-80 Data		Designed Test I	Designed Test II	Cycle rate
	%	D(m)	%	D(m)	%	%	ms
ACC	17.6	22.2	16.5	7.0	13.3	6.8	0.05
Slot	14.8	22.8	10.4	10.3	2.4	4.2	N/A
iPCB	19.3	23.7	15.8	13.7	0.9	1.2	0.20
PGM-G	20.1	24.3	11.3	14.6	0.9	1.8	0.51
PGM	8.7	25.8	7.6	16.4	0.4	0.2	0.08
smoothPGM	3.6	25.5	5.3	17.4	0.3	0.2	0.08

The host car uses each of the methods in the "Method" column for intention estimation and LQR for lower-level control. In a given test, the merge car replays a real trajectory from the dataset, and the host vehicle's start position and speed are also taken from the dataset. We then run our proposed PGM method to estimate the intention of the merging car and apply the LQR control model. The failure rates are shown in Table 4.2 in the "US-101 Data" and the "I-80 Data" columns. smoothPGM has the lowest failure rate, and does well on I-80 even though it was trained on US-101. In tests of different datasets, the D(m) columns show average distances between the host vehicle and the closest merging car when the first of these two cars reaches the merge point. smoothPGM has the highest average distance (it has similar D(m) to PGM), which is an indication of its greater safety. There are two reasons that make smoothPGM outperform our previous approach (PGM), which are highly related to the design of the proposed model: 1. smoothPGM does not rely on a fixed merging point. Since the merging point of PGM is fixed and it ignores the effect of the auxiliary lane, there are merging cars which run beyond the assumed merging point, resulting in a negative or undesired measurement (distance-to-merge); 2. The proposed method smooths out the speeds from location measurements instead of directly trusting noisy speed measurements.

4.1.2 Determine the optimal number of the observation nodes

To determine the proper number of state nodes, and inspect the smoother's effect on the estimation result, the proposed method was applied to a subset of the dataset with a varying number of speed nodes and compared with a version which does not have the smoothing step. In Figure 4.2, the blue line is the result for non-smoothing PGM, and the red one is for the proposed method. Since more past information can help the estimation, both lines go down as we increase the number of state nodes to around 20, and then go up since redundant nodes affect the sensitivity to the present dynamic changes. For the same reason, the two lines converge to one collision rate. Furthermore, the red line is below the blue line and its minimum is lower than that of the blue one. This plot indicates that by using the new approach, one can achieve the same level of collision rate by using fewer state nodes than the previous method, and also obtain a lower collision rate with no more than 20 state nodes.

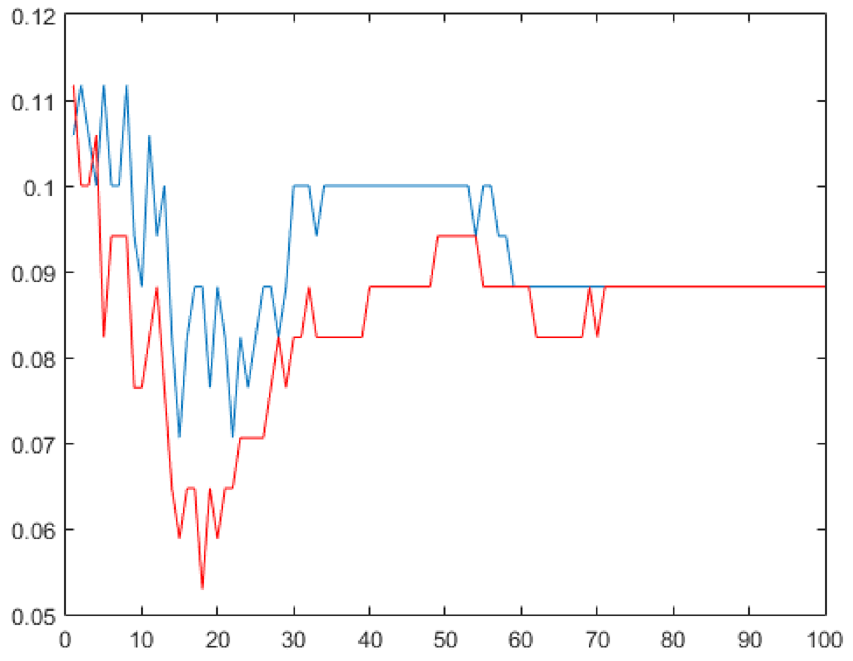
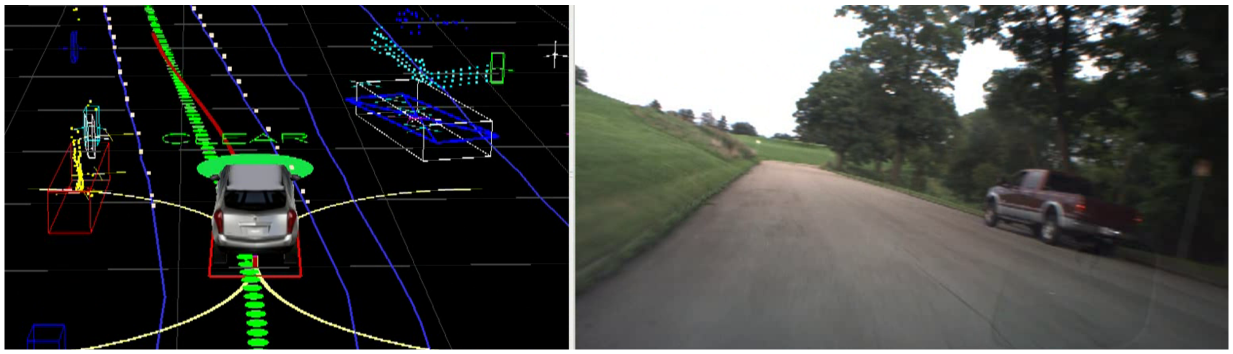


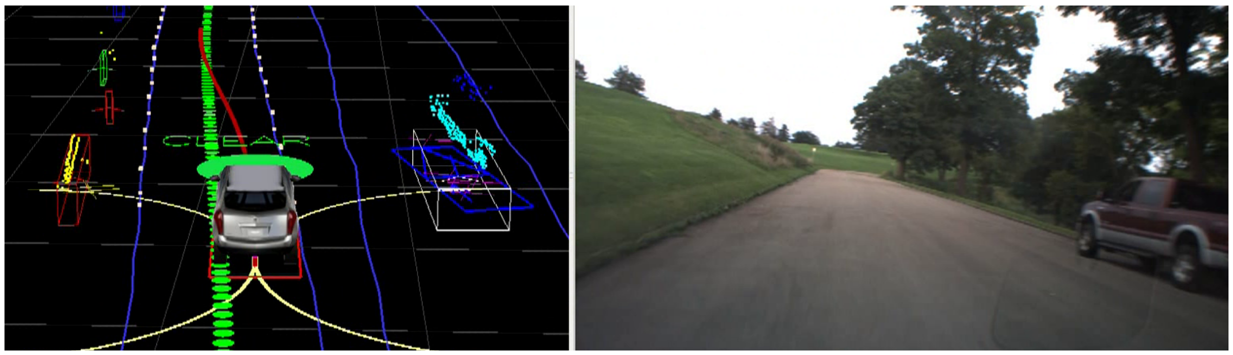
Figure 4.2: Collision rates vs. different numbers of speed nodes with (red line) and without smoothing (blue line).

4.1.3 Individual examples

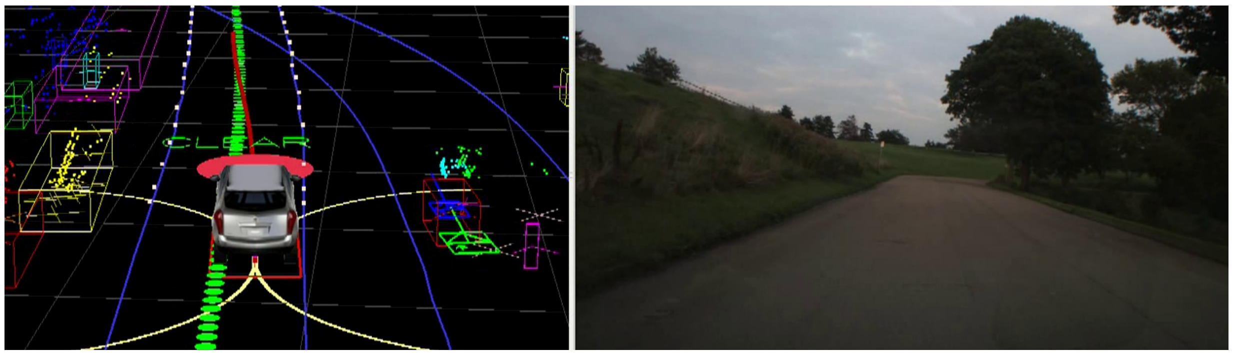
To verify the performance of the intention estimation for the single-merging scenario, an experiment was done with real cars in a closed area. A virtual merging ramp and main road were established in the simulator. Both merging car and host vehicle followed their designated routes. The host vehicle detects and tracks the merging truck (as shown in Figure 4.3a), and estimates its merging intentions. To visualize the estimation results, a colored circle in the left-hand-side figure (in front of the host car) indicates the hosts estimate of the merging car intention. A green circle indicates that the merging car yields, whereas a red circle indicates that the merging car does not yield. To create greater uncertainty in the tested scenario, the merging car suddenly sped up near the end of the ramp. The PGM also estimated that the merging car was overtaking, as the indicator circle turns from green to red in Figure 4.3c. To ensure safety of the test, even though the merging car sped up, the merging car driver did not follow the virtual merging ramp around the merging point. Instead, the tester drove the car out of the virtual map. Since the merging car is no longer visible on the virtual ramp, the indicator turns red to green in Figure 4.3d.



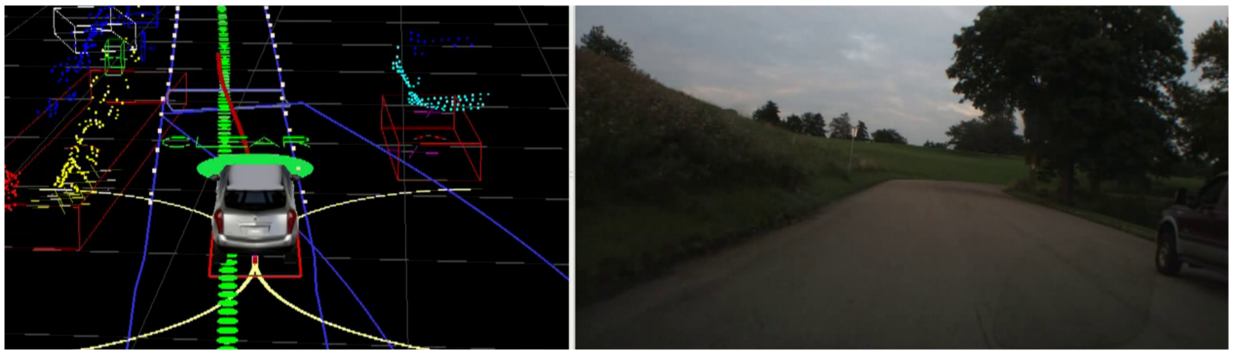
(a)



(b)



(c)



(d)

Figure 4.3: An individual example for single-merging PGM in a closed area in Pittsburgh with a virtual ramp scenario. The merge car (the truck) approached the merging point, and do not yield near the end of the ramp. For safety reasons, the merging car deviated from and exited the virtual ramp at the end.

4.1.4 Tests in Multi-Merging-Leading Merge Scenario

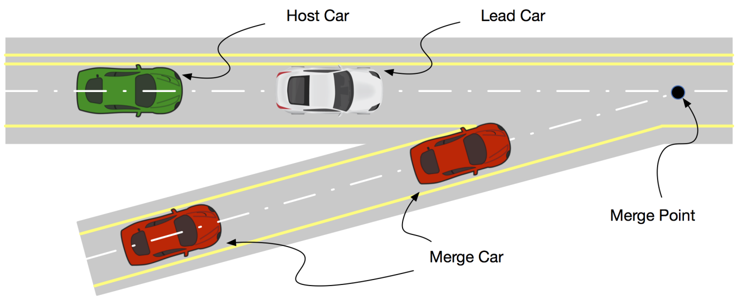


Figure 4.4: An illustration for the Multi-Merging-Leading scenario. The green car is the host car, the white car is the leading car, and red cars are multiple merging cars on the entrance ramp.

Experiments for a multi-merging-leading scenario (shown in Figure. 4.4) were also conducted on the datasets discribed in Table 4.1.

We use collision rate and time-to-collision (TTC) to the vehicle which runs in front of the host as safety criteria. Collision rates for the four algorithms are shown in Table 4.3. In the dataset, human-driven vehicles have no collisions. GeoACC, iPCB and MultiPolicy Decision Making (MPDM [55], which selects an optimal policy from preset ones) have high collision rates relative to MML-PGM. GeoACC only follows the closest merging vehicle without considering its intention, whereas iPCB considers the closest vehicle's intention. MPDM estimates the most likely behavior based on a simple forward simulation model and a manually designed reward function. None of them is easy to design to reflect real maneuver actions in forward simulation and a reward mechanism for selecting a proper trajectory. Note that even the proposed MML-PGM has a non-zero collision rate, which means the algorithm is still imperfect. Compared with 1-on-1 merging results in Table 4.2, the MML-PGM's collision rate is about 2 times higher than the 1-on-1 PGM's lowest collsion rate. The reason is that the 1-on-1 merging scenario only has one merging target, whereas in MML-PGM, collisions with non-merging-target vehicles are also counted. The model is not yet sophisticated enough, since it: 1) does not predict trajectories of merging vehicles, the host vehicle and the leading vehicle; 2) only has a binary output to react to the merging vehicles and the leading vehicle; 3) cannot handle uncertainty, e.g., noise.

In addition to the collision rate, Time-to-Collision (TTC) is also used as a safety criterion (details are introduced in the next paragraph). Having the TTC, Kullback-Leibler divergence [56] (K-L divergence) is applied to quantify the difference between our estimates and real human behaviors.

Table 4.3: Statistical results for different Merging Control Algorithms and human-driven vehicles in Multi-merging-leading scenarios.

Algorithms	GeoACC	iPCB	MPDM	**MML-PGM**	Human
Collision Rate	20.0%	18.9%	15.9%	**7.2%**	0.0%
K-L divergence	0.24	0.11	0.61	**0.02**	—
Rate/car (ms)	0.05	0.20	100.55	**0.08**	—

The second safety criterion considered is the Time-To-Collision (TTC) between the host vehicle (the ego vehicle) and the leading vehicle (which runs ahead of the host vehicle) when the host vehicle reaches the merge point:

$$TTC = (L_{lead} - L_{host})/(v_{host} - v_{lead}) \tag{4.1}$$

where L_{lead} and L_{host} are the distances to the merging point of the leading vehicle and host vehicle, respectively. Note that the leading vehicle always runs ahead of the host vehicle, so the numerator is always non-negative. If the speed of the host vehicle is lower than that of its leading vehicle (either the original leading vehicle or a merged vehicle), the denominator is negative, so the TTC is negative and no collision can occur, since the vehicle ahead is moving faster than the vehicle behind. There will be a collision if the TTC is positive, which means that the host vehicle is moving faster than the leading vehicle. We examine the distribution of the host vehicles' TTC for the four algorithms with the real data. The results are shown in Fig. 4.5.

Fig. 4.5 shows the Probability Mass Functions (PMF on the y-axis) over the TTCs that are generated from the four different algorithms and human-driven vehicles. The peak and majority (about 80%) of the MML-PGM TTCs are negative, whereas the other algorithms' TTCs have a

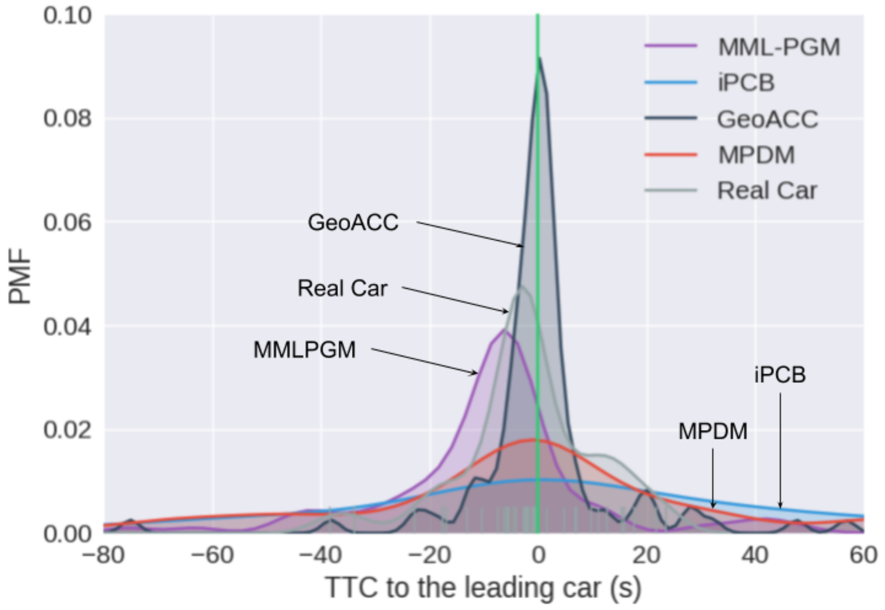


Figure 4.5: TTC to the leading vehicle when the host vehicle reaches the merging point.

peak at zero on the x-axis (indicating collision) and about 50% of the testing cases have positive TTC. The TTC results correspond to what is shown in Table 4.3: in most of the test cases, MML-PGM has negative TTC, which ensures that the gap between the host and its leading vehicle keeps growing, and results in a lower collision rate than that of the other algorithms. Note that the majority of the human-driven vehicles (grey curve) also have negative TTC. But the peak is closer to zero, which means that human drivers are more aggressive than MML-PGM but still retain a zero collision rate.

4.1.5 Individual examples

A simulation example

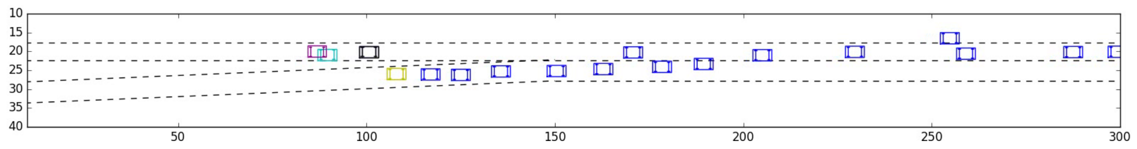
Experiments in simulation were conducted to highlight the features of the proposed discrete intention estimation, in comparison with the previous iPCB model. The sequences of figures (Figure 4.6 and Figure 4.7) show the results of iPCB and MML-PGM in the same scenario. This scenario is one example from the US101 dataset. All surrounding cars replay their trajectories from the dataset. The initial status of the host vehicle (the initial position and speeds) is also the same as the dataset. Then iPCB and MML-PGM run separately to estimate the intentions of all surrounding cars, and generate interactive behaviors for the host vehicle.

Figures 4.6 and 4.7 show the comparison between the iPCB and MML-PGM for an individual example. The black rectangle indicates the leading car; pink is the host car; cyan is the human-driven host;blue designates merging cars; yellow is the following target.

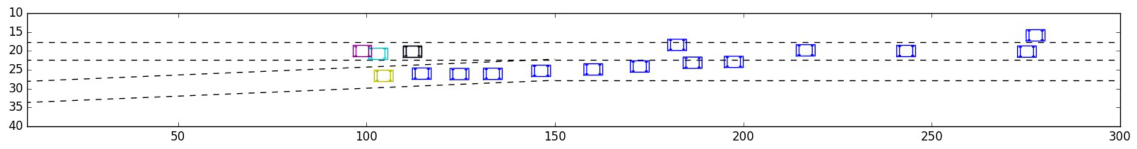
Figure 4.6 shows a sequence for one merging example, with the intention estimation is conducted by iPCB. iPCB only analyzes the intention of the closest vehicle, and does that based on the acceleration. The following target changes frequently, which leads to instability. As shown from Figure 4.6b to Figure 4.6e, the host vehicle repeatedly switches its following target, i.e., the merging car which is most likely to cutin in front of the host vehicle.

MML-PGM analyzes intentions of all merging cars w.r.t. both the host car and its leading car, and then returns a relatively stable following target. Its behavior is similar to that of the human-driven car. From Figure 4.7a to 4.7c, the host vehicle consistently follows its leading car, without identifying a merging car to follow as a target. This means that the merging estimation program on the host vehicle does not believe that there will be a cut-in merging car. It will maintain the distance-keeping behavior to follow its current leading car, until the estimation program detects a merging intention from a car that is preparing to cut in between the host vehicle and the leading car. From Figure 4.7d to 4.7e, it recognizes the merging intention of one of the merging cars, and then sets it as the following target.

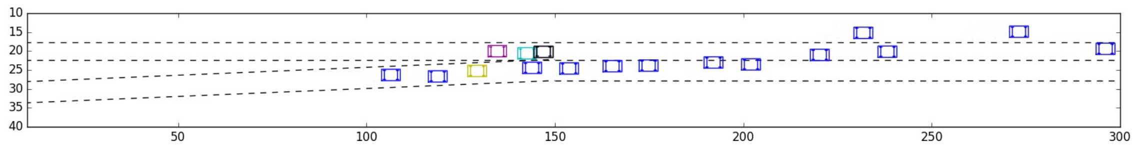
As a result, comparing Figure 4.7e with Figure 4.6e, the host vehicle (in pink color) that is controlled by iPCB is too late to react to the merging car when it finally identifies the last following target, and then collides with the merging car. In the case of MML-PGM, though the host vehicle falls behind its human-trajectory ground-truth, it makes a larger space than the human between the merging car and itself. The larger space in the longitudinal direction grants a safer merging behavior.



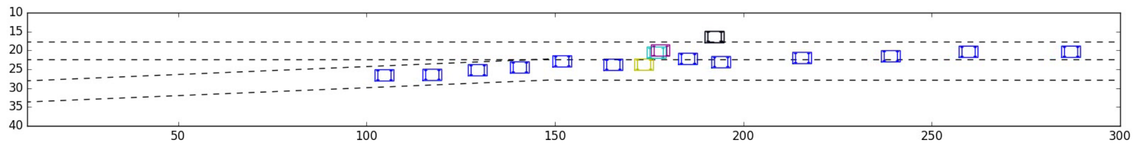
(a)



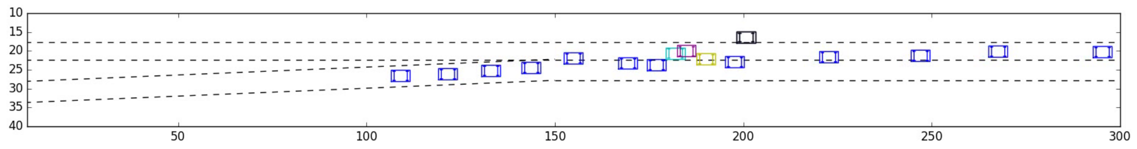
(b)



(c)

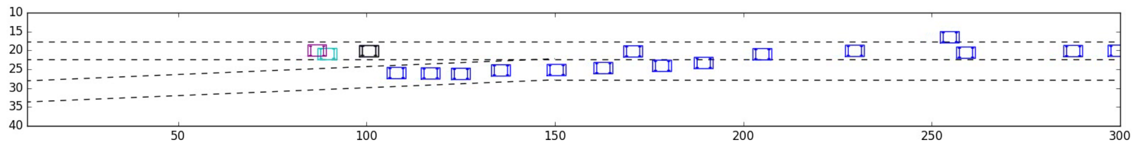


(d)

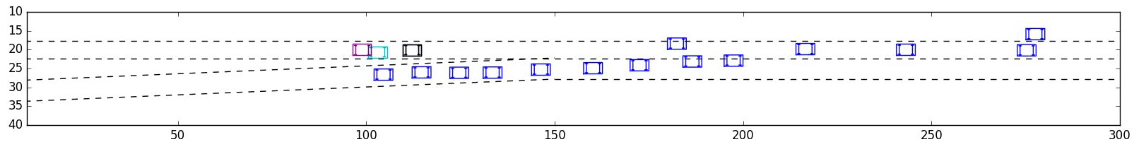


(e)

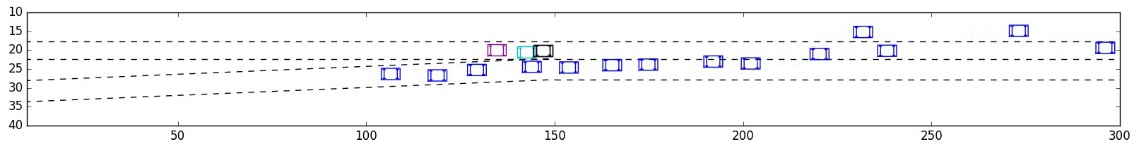
Figure 4.6: An individual example for iPCB in US-101 dataset.



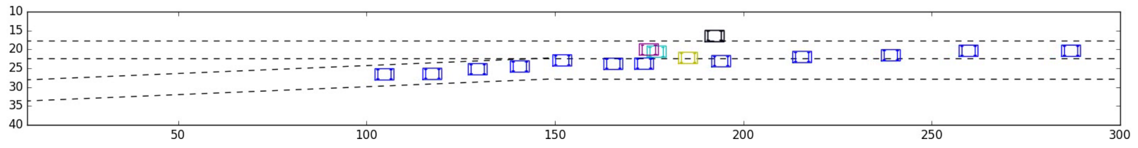
(a)



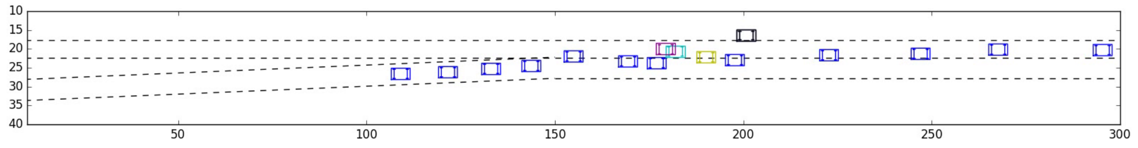
(b)



(c)



(d)



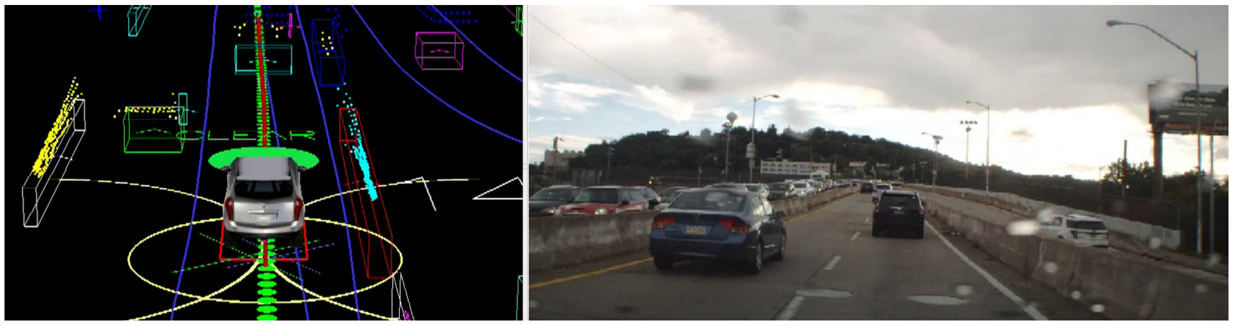
(e)

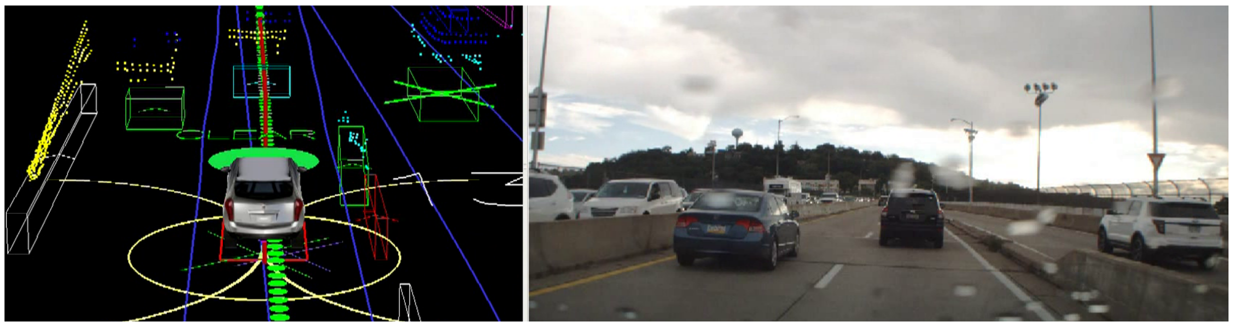
Figure 4.7: An individual example for MML-PGM in US-101 dataset.

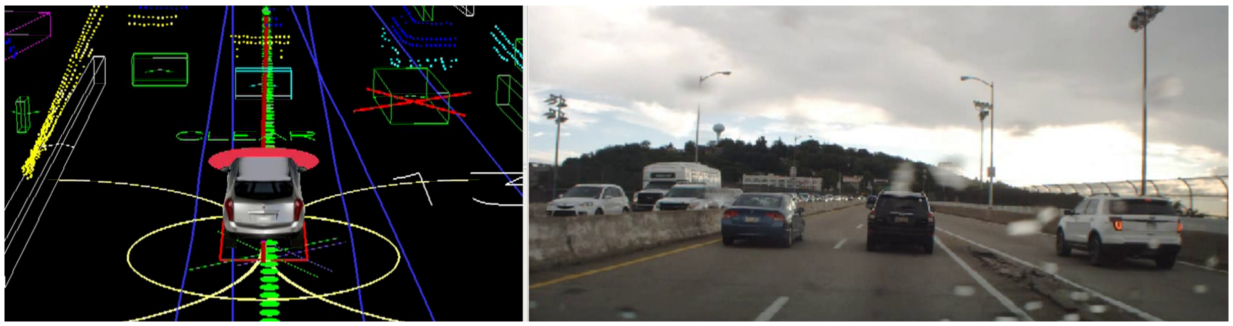
Since both iPCB and MML-PGM use a distance keeping model that limits the distance between the host and leading vehicle, which is larger than 1.5 meters, the program-controlled host vehicle cannot behave as aggressively as the human driver. As shown in Figures 4.7c and 4.6c, the distance between a human driver (the cyan car) and its leading car (the black one) is as small as 0.5 meters. However, the host car (the pink one) has a longer distance to the leading car. This is also one reason that the behavior of the MML-PGM deviates from that of the human in the statistical results (Table 4.3 and Figure 4.5).

Public Road Testing

To further verify the estimation performance, a public road test was conducted in the southbound merging area on Bloomfield Bridge, as shown in the sequence of figures in Figure 4.8. To ensure safety in the public area, the autonomous vehicle runs in open loop: the target selection and accelerate/decelerate decisions are made autonomously, but the lower level control, i.e., the acceleration rate and emergency brake are managed by the safety driver through pedals. Figures 4.8a and 4.8b show that the host car follows the leading car, and keeps track of a merging car (the white SUV on the right merging ramp, indicated by the green cross in the left-hand-side figures). The host car runs MML-PGM to evaluate the merging intentions w.r.t both the host car and the leading car. If it does not consider the leading car, the host car can pass the merging car. However, the leading car running in front of the host car does not allow acceleration. Figures 4.8c and 4.8d show that the host car finally switches the following target to the merging car, and follows the new target. In the left-hand-side figures, the merging indicator turns from green to red and the following target on the ramp is marked as a red cross. Even though, in Figure 4.8c, it is clear from its brake lights that the merging car brakes, combining its location, speed, and historical information, the MML-PGM model does not believe that the merging car is going to yield. The main advantage of the MML-PGM is that it does not depend exclusively on the immediate dynamic information of the merging vehicle.


(a)


(b)


(c)

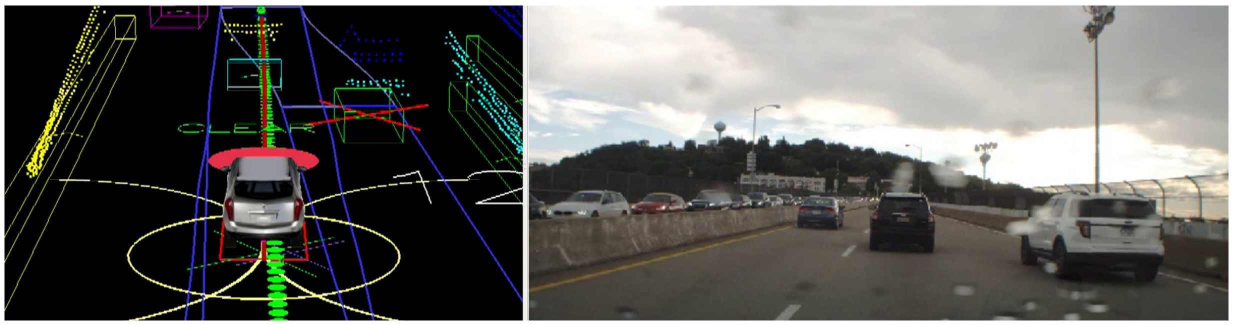

(d)

Figure 4.8: An individual example for MML-PGM on the Bloomfield Bridge Ramp area in Pittsburgh. The merging car is the white SUV on the ramp, shown in the right-hand-side photos. The left-hand-side figures show the scenario in the control software, centered on the host vehicle. The green/red circles are the intention indicator which shows the estimated intention. The red circle indicates that the merging car's intention is "Not Yield", then the host vehicle decelerates and follows the merging car.

4.2 Experimental Results for Intermediate Behavioral Prediction

The intermediate behavioral prediction is the second stage in the behavior estimation system. The output of this stage is semi-continuous behaviors. For example, in the lane change scenario shown in Figure 3.7, it outputs a proper lane change start and end points for an target vehicle. The results for semi-continuous behavioral estimation in lane-change scenarios are shown in Figure 4.9. The method takes a short period of observations of surrounding vehicles around a target vehicle, implicitly analyzes the mutual interactions, and finally results in a prediction. The following results illustrate the prediction of the lane-change start and end points for a target vehicle. The result can be used in the trajectory planner of an autonomous driving vehicle, or can be used to estimate a target vehicle's future behavior.

Table 4.4: Statistical results for different kernels compared with ground-truth, All units are meters (m).

kernels	μ_{start}	μ_{end}	σ_{start}	σ_{end}
Laplacian RBF[45]	-54.90	-116.61	24.10	43.40
Gaussian RBF[45]	-13.55	-31.15	16.20	25.42
IMK with $\|\|\cdot\|\|_F$	-0.95	-18.50	6.38	13.77
IMK with $\|\|\cdot\|\|_S$	1.78	-17.90	5.87	13.10

Figure 4.9 shows the start/end point predictions of six scenarios from the testing group. Highlighted segments are used for prediction, and are the only input to the proposed method. The segments consist of all traffic participants' trajectories in a 3-second time window. The red diamonds are the output of the method, and indicate the start points and the end points of the lane-change behavior. The real lane-change paths generated by human drivers are shown as the black curves on the $Time = 0$ plane. The red diamonds (the outputs) are close to the turn points of the black curves, which indicates that the predicted start/end points correspond to feasible lane-change behaviors.

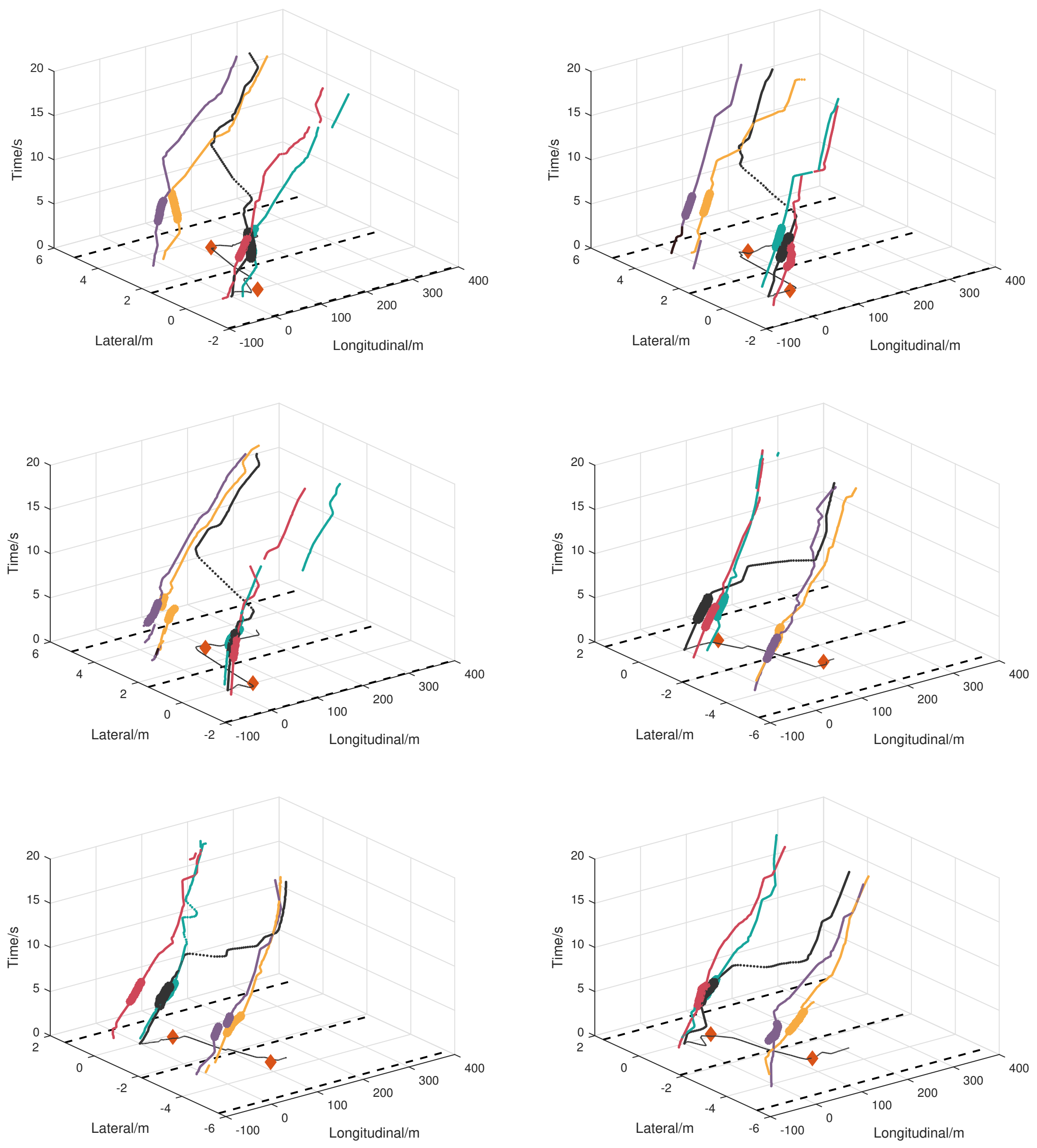


Figure 4.9: Examples and results of the estimation. Vertical axis is time (s). The black curve is the host vehicle's trajectory; colored curves are surrounding vehicles'; the black curve on the $Time = 0$ plane is the projected path of the host vehicle; Dashed straight lines are lane dividers; Red diamonds are predicted start/end points. Highlighted segments on the curves are used to predict the start/end points.

450 groups of trajectories are randomly selected as training sets, and the remaining 93 groups are used for testing. To concentrate on the recent past, training trajectories are pruned and only retain the last 30 steps (3 seconds) before the host vehicle starts the lane-change (when the

heading departs from the orientation of the current lane.).

Four kernels are tested: Laplacian RBFs, Gaussian RBFs which are suggested in [45], and inverse multiquadric kernels (IMK) constructed with the Frobenius norm $||\cdot||_F$ and the Spectral norm $||\cdot||_S$. Results are shown in Table 4.4. μ_{start} is the difference in the start point between the estimate and the ground-truth, and σ_{start} is the standard deviation. μ_{end} is the difference in end point between the estimate and the ground-truth, and σ_{end} is the standard deviation.

Table 4.4 shows that the RBFs have large errors. However, according to the nature of the Frobenius norm and Spectral norm, which work similarly, both significantly outperform the other kernels. In terms of standard deviation, the performance of the Spectral norm is slightly better than that of the Frobenius norm. In the second row, the widely used Gaussian RBF kernel, which is also used in [45], performs worse than the inverse multiquadric kernels with Frobenius norm and Spectral norm.

In summary, the results show that the current framework can handle discrete intention estimation in ramp-merging scenarios and semi-continuous behavior estimation in lane-change scenarios, with lower collision rates, larger safe distances, and similar reactions compared with the ground-truth.

4.3 Experimental Results for the Neural-Process based Trajectory Prediction

The third stage of the system is the estimation of the future trajectory of a target vehicle, which is based on the past observation of its surrounding vehicles. As shown in Figure 4.10, instead of giving semi-continuous points, the full trajectory estimation takes past observations of the target vehicle and its surrounding vehicles and then estimate a full trajectory for the target vehicle.

This section shows the performance of the trajectory prediction via Neural Process. The model (shown in Figure 4.11) also takes the historical trajectories of all surrounding vehicles and

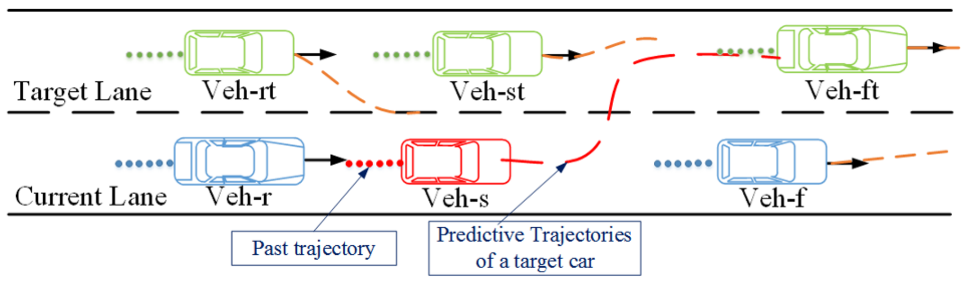


Figure 4.10: A scenario of full trajectory prediction for a target vehicle (red). The predicted trajectory is the red dashed line. Five surrounding cars are taken into consideration. Dot lines are the input of the predictor.

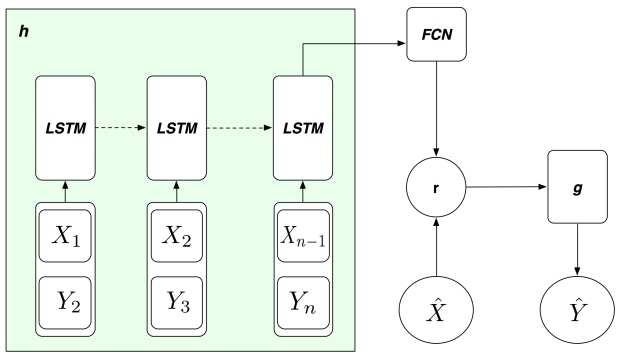


Figure 4.11: A recap of Recurrent Meta Induction Network

the host vehicles as input. A set of historical trajectories is firstly used as input, which results in a "conditional value" $\mathbf{r}$. This value is a representation of the input historical trajectories. The most recent observation is used as the second input. Combining the new observation with $\mathbf{r}$, a trajectory prediction is made. The same dataset is used to learn and evaluate the system, where there are 543 lane changing examples from US-101 and I-80 scenarios in NGSIM dataset. Besides the following car and the leading car in the host lane, three closest cars in the target lane are also recorded and considered in the method.

4.3.1 Baseline Models

Three baseline models are implemented to demonstrate the features and improvements of the proposed algorithm.

1 Continuous trajectory estimation in RKHS [52];

2 Conditional Neural Process (CNP) [49];

3 Plain LSTM trajectory estimation.

Note that all of the baseline algorithms predict the trajectory by considering the interactions between the target and its surrounding cars. Historical trajectories of all involved cars also contribute to the prediction in these baseline methods.

Conditional Neural Process (CNP) is implemented as a baseline model. The main difference between CNP and RMIN is that, to retain the property of a stochastic process, CNP should be permutation-invariant. Therefore, there is an aggregation operation $\bigoplus$ in an intermediate layer between the demonstration and generator sub-net. The implementation also retains this property, and as is recommended, the aggregation operation $\bigoplus$ is implemented by a mean operator, which means that

$$\mathbf{r} = \frac{1}{n}\sum_{1}^{n} r_i$$

In addition, the dimension of the intermediate condition representation $\mathbf{r}$ is 128. The intermediate condition $\mathbf{r}$ is the output of the demonstration sub-network h. h contains a three-layer fully connected network, with increasing dimensions of 32, 64, and 128. Note that the last layer of the fully connected network outputs the condition representation $\mathbf{r}$. Followed by the $\mathbf{r}$, a new observation input x_t is combined. A concatenated tensor which contains $\mathbf{r}$ and x_t is then fed into the generator network g. The generator g also consists of a three-layer fully connected network. The CNP baseline shows the performance of the induction network. However, it does not consider the sequential information. This lack inspired the proposed method, which takes the sequencing of trajectories into consideration.

A plain LSTM sequence-to-sequence trajectory estimation is also implemented; a one-stack LSTM model is used. Instead of using a whole observed trajectory as the input, it feeds individual (x, y) coordinates once per step.

In the design of the proposed method, only one LSTM is stacked for the observer sub-net. The LSTM cell uses an asynchronous structure, which means that it will not output the intermediate condition representation $\mathbf{r}$ until it has processed enough data. We set the number of the demonstrations to three, so that the demonstration sub-net will generate a result $\mathbf{r}$ only once it has processed three segments of past trajectory. However, the output dimension of the LSTM cell is the same as its input. With a possible variable dimension of the input, its output dimension is desired to be fixed. Then a fully connected network follows immediately after the LSTM observer network. To make the structure more comparable to the CNP implementation, the fully connected network only has one layer, containing 128 hidden nodes. The dimension of the condition representation $\mathbf{r}$ is equal to the one in CNP. The generator architecture uses the same design as that in the CNP implementation, i.e., a three-layer fully connected network. Note that in the implementation of the continuous trajectory estimation in RKHS (the first baseline method), scenarios that have different numbers of surrounding vehicles are separately trained and the model is updated by a selection matrix. Otherwise, the result of the RKHS will be poor. However, the induction models (including CNP and the proposed RMIN) are more robust than other methods when facing the uncertainty of the surrounding traffic, as long as the total number of surrounding cars does not exceed the design value. In the current implementation, the upper bound on the number of surrounding cars is five, excluding the target car itself.

4.3.2 Comparison Results

Table 4.5 and Table 4.6 show the prediction errors of the different methods at each timestamp. Mean errors and standard deviation are used in the comparison. To better demonstrate the results, errors are separately shown in longitudinal and lateral directions. All of these methods have increasing mean errors and standard deviations. The RKHS method can only predict over a 2s horizon within an acceptable accuracy.

In the lateral direction, the proposed method achieved the lowest mean error at each examined timestamp. The mean error of the RKHS at 2s is at the same level as that of RMIN at the 5-second mark. At 2s, RMIN has a mean error that is half of CNP's and 36% of the error of

Table 4.5: Mean error and standard deviation of lateral error of the trajectory prediction compared with the ground-truth, in the lateral direction. The units are meters (m).

Methods		1s	2s	3s	4s	5s
RKHS	μ	0.052	0.251	—	—	—
	σ	0.051	0.250	—	—	—
LSTM	μ	0.286	-0.330	-0.588	-0.776	-1.209
	σ	0.776	0.880	0.919	1.020	**1.160**
CNP	μ	0.085	0.181	0.382	0.379	0.444
	σ	0.476	1.268	1.995	2.325	2.472
RMIN	μ	0.019	0.090	0.195	0.235	**0.248**
	σ	0.501	1.299	1.997	2.343	2.492

Table 4.6: Mean error and standard deviation of the longitudinal error of the trajectory prediction compared with the ground-truth. The units are meters (m).

Methods		1s	2s	3s	4s	5s
RKHS	μ	1.425	9.925	—	—	—
	σ	0.871	5.794	—	—	—
LSTM	μ	-2.100	-4.429	-6.050	-9.851	-17.932
	σ	6.356	8.225	10.303	14.608	20.553
CNP	μ	-0.229	-0.565	-0.782	-0.979	1.126
	σ	2.184	4.846	7.685	10.764	**13.995**
RMIN	μ	0.169	0.485	0.666	0.831	**1.091**
	σ	2.606	5.476	8.611	11.946	15.510

RKHS. However, RMIN has the largest standard deviation at that timestamp. At 5s, RMIN has a mean error of 0.248 meter, which is 55% of the CNP error at the same timestamp. Their standard deviations are at roughly the same level. Though the LSTM approach has the lowest standard deviation in the lateral direction, its absolute mean error is almost five times larger than that of RMIN.

In the longitudinal direction, the proposed method also achieves the lowest (absolute) mean error at each examined timestamp. RKHS is significantly worse than other methods in terms of mean error, even looking at the first two seconds. At 2s, the standard deviations are at the same level. However, the absolute mean errors of CNP and RMIN are around 0.5, which is 1/20th of the error of RKHS. One possible reason is that RKHS uses one kernel function to estimate the whole trajectory with 2 dimensions. In detail, only one kernel function is used for the estimates in both lateral and longitudinal directions. At 5s, the mean error of RMIN is 27.8% smaller than that of CNP. The standard deviation of RMIN is only 9.8% larger than CNP's. The LSTM approach performs badly in the longitudinal direction, though its lateral performance is comparable to that of the other methods. Note that its absolute mean error at 1s is already larger than that of CNP or RMIN at 5s. There are several reasons that contribute to the differences: 1) The input of the plain LSTM directly takes individual coordinates at each time step. However, RMIN and CNP take a segment of trajectories at each time step respectively as input to either an LSTM or a fully connected layers-based observer network. The latter provides more data and relations to the observer network. The induction models also use both historical observations and their expected output as inputs to extract the correlations between the observation and output, and the relationship is encoded as the condition tensor. This design makes it easier for induction models to accurately extract the tendency of the trajectory changes. In addition, the speed along the longitudinal direction is larger than that in the lateral direction. 2) The output (generator) network of the induction models uses a fully connected network, whereas the plain LSTM uses one LSTM cell. Since the fully connected network (FCN) describes the relationships among positions in the predicted trajectory more explicitly than the LSTM cell does, with a small set of training data, the FCN in the induction models can work better. 3) Therefore, any subtle changes of the observation (especially for the longitudinal positions) and model prediction error can result in a larger deviation in the longitudinal direction than in the lateral one.

4.3.3 Individual Results

Figure 4.12 shows two individual examples of lane change trajectory prediction. Figure 4.12a is an example for the right lane change prediction and Figure 4.12b is for the left lane change prediction. The target car's trajectory is shown as black lines. Surrounding cars are represented by other colors. The thick parts are the observations (demonstration). The light gray curve that follows the thick black line is the ground-truth, and the red curve that follows the thick black line is the prediction of the target car. To have a clearer comparison, the prediction and ground-truth are projected to the ground plane (z=0). Dashed thick lines on the ground are lane markers.

The individual results show that considering three seconds of demonstration before lane change, the predicted trajectories are close to the ground-truth. The predicted trajectories' curvatures also follow the lane change pattern: The curvature increases at the beginning of the lane change, and then decreases. Once the target car passes the lane marker, the curvature increases and then directs the heading of the car to align with the lane.

4.4 Summary

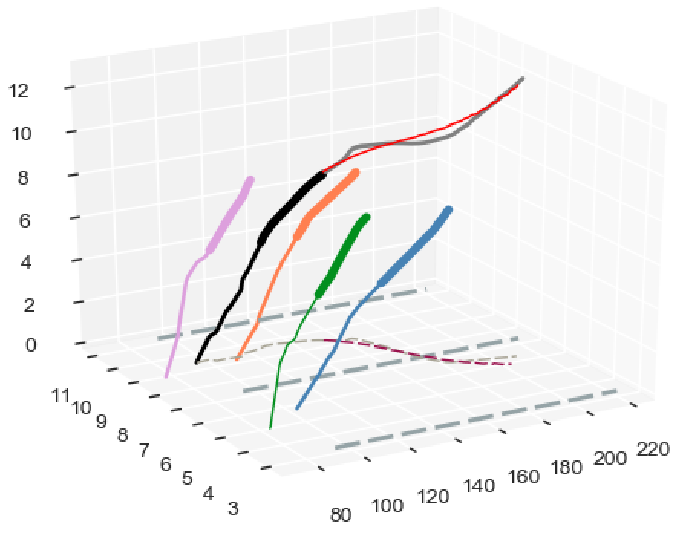
In this chapter, three methods that correspond to three stages of behavioral estimation are tested in various scenarios. The tests are conducted to compare with human-driven ground-truth trajectories, in a realistic simulator which contains more testing cases, and finally, the method is implemented on a self-driving platform to test its performance in real urban traffic. The experiments validate the performance of the methods for different stages:

1. The PGM-based algorithms surpass previous methods for the intention estimation and interaction in merging scenarios: including one-on-one and multi-merging-leading scenarios;

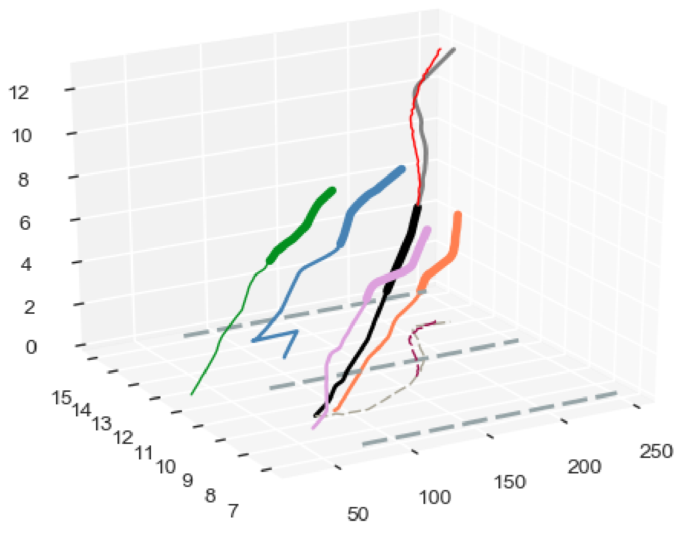
2. The RKHS-based algorithm predicts the end and start points of a lane change behavior of a target vehicle, considering its past trajectory and past observations of the surrounding vehicles;

3. The RCNP-based method outperforms other methods in the prediction of a lane changing trajectory, which is based on the observation of the behavior of surrounding vehicle. By

using an induction neural network, the performance can be achieved by using a small dataset.



(a) A right lane change trajectory prediction



(b) A left lane change trajectory prediction

Figure 4.12: Individual examples for right and left lane change of a target vehicle, with surrounding cars. x-axis is for the longitudinal direction; y-axis is for the lateral direction; z-axis is for the time horizon. Black curves are the trajectories of the target car. Detailed discussion and description are in Section 4.3.3.

Chapter 5

Analysis of Practical Issues

Chapter 3 introduces the behavior system in interactive scenarios. Chapter 4 shows the feasibility of the proposed work in ideal set-ups. In part of the realization of the system, not only the theoretical analysis is performed, but engineering and implementation aspects are also studied in this chapter. These studies include the sensitivity analysis for sensor failures and the discussion of practical integration to the autonomous driving system.

5.1 Sensitivity and Failure Analysis

In this section, we mainly focus on the sensor failure. Without specifying a certain type of sensor, like LIDAR, RADAR or Camera, the study is conducted at the tracker level. Tracker drop-off, drifting and oscillating effects are examined. The following sections will discuss the sensitivity to perception uncertainties. We then propose requirements for the perception system in terms of tracker stability and coverage ranges. This study gives guidelines and criteria for the design of an engineering realization of a perception system on an autonomous driving car. For example, we discuss the desired range of LIDAR/RADARs and resolution of cameras.

5.2 Sources of Uncertainties

Uncertainties come from the driving environment, drivers and perception equipment. To better understand these effects, several main sources are listed in detail below:

- Human intentions: A human driver may change his mind during the negotiation and different individuals have different intentions in the same scenario.
- Road geometry: Different merging ramp angles may lead to various behaviors.
- Vehicle type: Sedans and trucks will behave differently based on their own dynamics.
- Sensing coverage: The longest distance that a tracker can be stably estimated affects how far we can see.
- Sensing stability: The oscillation of the detected tracker in pose and velocity affects how well we can see.

This chapter focuses on the analysis of the sensitivity of the proposed system, with respect to the uncertainty of the perception system, i.e., sensing coverage and stability. The analysis aims to provide a guideline for the design of the perception system for interactive tasks. Since the proposed framework is not limited to a specific type of perception system (e.g., LIDAR, RADAR or cameras), the analysis is conducted on the tracker level. A tracker represents the pose, velocity and dimensions of a surrounding vehicle. The noise and failures will be applied to the tracker. Therefore, the uncertainty can be separated into the following phenomena:

- **Tracker drop-out:** Temporary losing track of one object. Effects of drop-out frequency and period will be studied.
- **Tracker drifting:** A low-frequency offset from the real position of the vehicle.
- **Oscillation:** A high-frequency error in the kinematic information of the tracked objects.

In the test process, other types of tracker failures, such as unexpected tracker divergence or merging are also observed. However, these types of failure have been fixed by the fusion system followed by the tracking system. Since no object can appear or disappear around another one, it is straightforward to identify and filter out these failures if the dimension, pose and kinematics can be detected. The fusion and tracking system is not the main contribution of this dissertation; therefore, only the effects of tracker drop-out, drifting and oscillation will be examined. Details of the fusion framework and segmentation/tracking methods can be found in [57, 58, 59].

5.2.1 Verification factors

In simulation, various scenarios can be generated. These scenarios have different initial conditions, i.e., start point S_0, initial speed V_0 for both the merging and host vehicle. The start positions range from 180 meters to 60 meters to the merge point, and the initial speeds range from 10 to 20 meters per second. Using a resolution of one meter and one meter per second respectively for the initial position and velocity, and applying the initial states to both vehicles, there are 2400 cases in total. To make the comparison clear and straightforward, the three factors are examined separately. Coupled effects are not considered in these experiments.

5.2.2 Drop-out

Bernoulli Dropout

As discussed above, two items need to be considered in the drop-out effects: 1) drop-out frequency and 2) drop-out period. Each item needs a separate set of testing cases. Since the update rate of the whole estimation is set to 10Hz, only drop-out frequencies that are lower than 10Hz will be considered. Note that the synchronization between the perception and the intention estimation system is not the topic here. For the sake of clarity, only the end effect of perception drop-out is discussed. In other words, the dropout effect results in an unobservable tracker at some timestamps.

The drop-out period is the time for a single drop-out. In the view of the estimation system, this results in a number of speed nodes with no observation value. Based on our study about the optimal number of observations in Section 4.1.2, the whole observation period for a single update cycle will take up to 20 observations in total, which is two seconds.

To formulate the dropout uncertainty, the event of dropout D is firstly considered as i.i.d. at each timestamp. In this way, the state of one observation is either dropout or the real perception value, with assignable probability. Therefore the probability of the dropout event can be modeled by a Bernoulli Distribution:

$$D \sim Bernoulli(p) \tag{5.1}$$

where p is the probability of the occurrence of the sensing dropout. Since there are about 100 timestamps in a merging control cycle and the dropout event is i.i.d., the number of dropout events in a merging scenario then follows the Binomial Distribution:

$$N_D \sim Binomial(n, p) \tag{5.2}$$

where n is the total number of timestamps and p is the same p as in the Bernoulli distribution. To test the sensitivity towards the dropout effect, different values of p are studied and tested. In addition, since the period of a dropout event can be longer than one timestamp, the length m of one dropout event is also studied. Since the probability of the start of a dropout also follows the Bernoulli Distribution in (5.1), the whole process follows a more complex distribution than Binomial:

$$\begin{aligned} f(k,n,m,p) = &\binom{n-m-km+1}{k} p^k (1-p)^{n-k(m+1)+1} \\ &+ \sum_{l=1}^{m} \binom{n-l-(k-1)m+1}{k-1} p^k (1-p)^{n-2l-(k-2)m+1} \end{aligned} \tag{5.3}$$

where k is the number of the dropout, m is the length of one dropout, and l is the length of the last dropout if it happens at the end of the observation ($l \leq m$). Details, explanation and derivation can be found in Appendix B. The experiments are to obtain the collision rates with a changing dropout probability p and dropout length m, where $p \in (0, 1)$ with step of 0.05 and $m \in [1, 11]$ with step size of 2. Since the optimal size of the observation is $n = 20$, the value of m cannot exceed n.

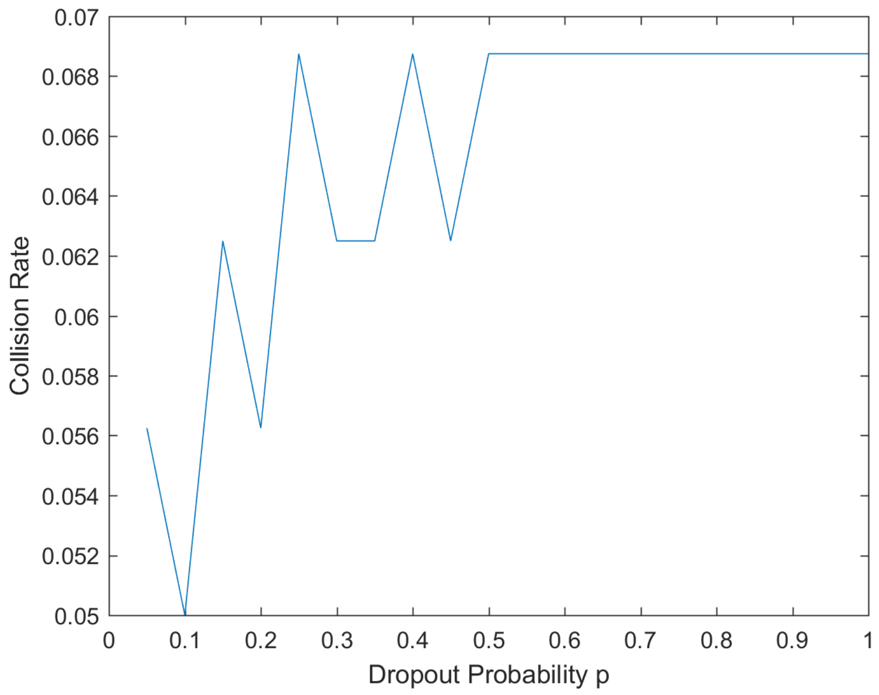


Figure 5.1: Collision rates with varying dropout probability. Dropout length $m = 1$.

Figure 5.1 shows the collision rates with varying dropout probability when the dropout length is one. The collision rates increase in an oscillatory way. Once the probability reaches 0.5, the collision rate become 6.8%, and does not change along with the increasing dropout probability.

Fixed length Dropout with varying locations

The dropout location is also an important factor which leads to different collision rates. We define the dropout location as the ratio of the traveled distance at the beginning of dropout to the length of the whole merging trajectory. The length of dropout is the number of consecutive timestamps during which the perception system fails to give the positions of the merging target.

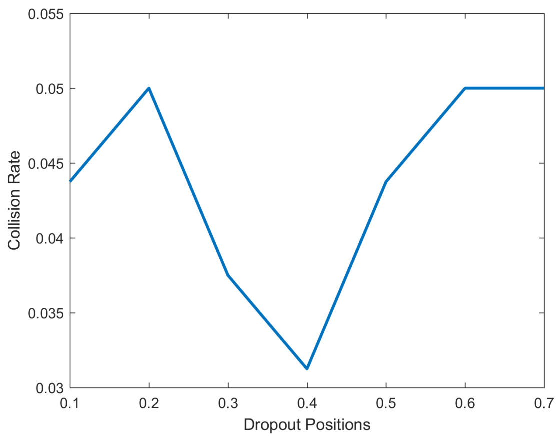
Figure 5.2 shows the change of collision rate with varying dropout location for two different drop lengths. When the observation is unavailable, old data are used without any prediction. The reason why unobservable moving distances is not used is that, normally, the perception will be offline randomly and then recover after a certain period. Since the merging car may move in different speed, which results in different unobservable distances, the dropout period is preferred.

To better illustrate the effects of dropout locations, only one dropout event is considered. Note that one dropout event may block the observations for a certain number of timestamps. The x-axis stops at 70%, since from 70% to 90%, the collision rate remains unchanged. In each test, the dropout location varies from 10% to 90% of the whole merging trajectory. In Figure 5.2a and 5.2b, regardless of the length of the dropout (up to half of the number of total observations in a cycle), the collision rates increase when the dropout location is within the first 20% of the trajectory. From 20% to 40%, the collision rates decrease to about 3.2%. After 40%, the collision rates gradually increase to 5%. If the received sensing data are not enough, the default behavior will slow down the host vehicle and let the merging car go first. This default behavior does contribute to the 5% collision rate, however, the speed of the host vehicle will be unnecessarily decreased, which leads to unreasonable longer time for the whole merging behavior.

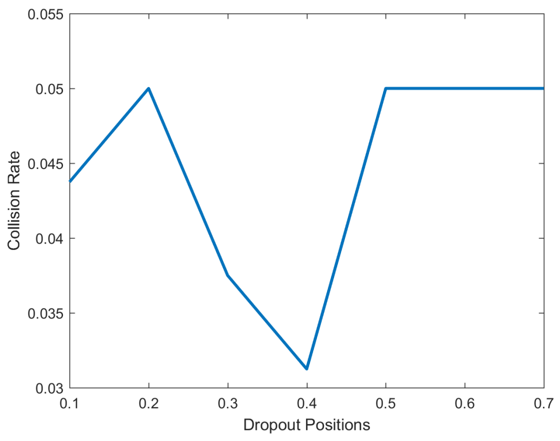
It is easy to understand that the late dropout will lead to more collisions, since it will affect the last-minute decisions. Therefore, the collision rates increases after 40%.

5.2.3 Drifting

A lower-frequency error can exist in the dynamic information of a tracker, and affects the accuracy of the pose (locations, heading) and the speed of the object. This type of error is common in the perception system, not only as a result of the detection and tracking algorithm, but also from the nature of the sensors. Sensors like LIDAR and camera can only observe at most 3 surfaces (in parts) of a vehicle. Using the partial observations will naturally introduce error in the estimation



(a) Drop length $m = 5$.



(b) Drop length $m = 10$.

Figure 5.2: Collision rates with varying dropout location with drop length $m = 5$ and $m = 10$.

of the center of mass and the dimensions of the target vehicle, hence affect the speed estimation through the derivative of the pose. In the case of LIDAR observation, if the object is approaching the ego-vehicle, the tracker position estimate will be closer than the real location; as the distance between two cars shrinks, the pose estimation goes back to the real value; and after the target car passes the ego-vehicle, the error increases again. Figure 5.3 shows the phenomenon of the described low-frequency error. Different curves represent different levels of deviation. Considering most sensors nowadays can detect objects robustly at 100 meters, and limiting the type of vehicle to sedan (5.5 meters in length), the upperbound of the error is ± 2.3 meters (half length of a sedan).

Figure 5.4 shows the resultant collision rates w.r.t. to the different coefficients in Figure 5.3. It is clear that as the coefficient increases, the collision rate tends to increase. In the end, the failure rate is approximate 9.2%. Same as the intuition, this indicates that the low-frequency drifting error results in less failures. However, it is impossible to have a zero drift target position and speed estimation. If the system is required to have a failure rate lower than 8%, the coefficient value should be less than $1.3e^{-4}$, which means that at 100 meter, the drift should be less than 1.3 meters.

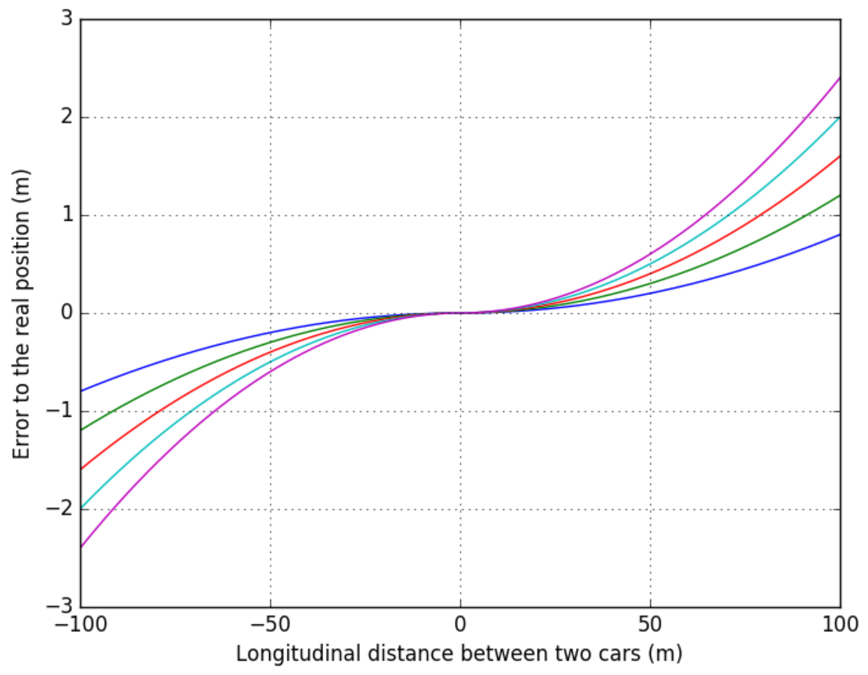


Figure 5.3: A designed pattern for the shape of low-frequency error on the tracker. X-axis is the distance between two cars, negative value means that the target car is behind the ego car. Y-axis is the error to the real pose.

The error follows the equation:

$$E = \begin{cases} - \quad coef \cdot (P_h - P_m)^2 & (P_h - P_m) < 0) \\ \quad coef \cdot (P_h - P_m)^2 & (P_h - P_m) \geq 0) \end{cases} \tag{5.4}$$

where E is the error which is the function of the coefficient and the distance between two cars. $coef$ is the coefficient which changes its value from 1.3×10^{-4} to 2.3×10^{-4}. P_h, P_m are the positions of the host and merging car, respectively.

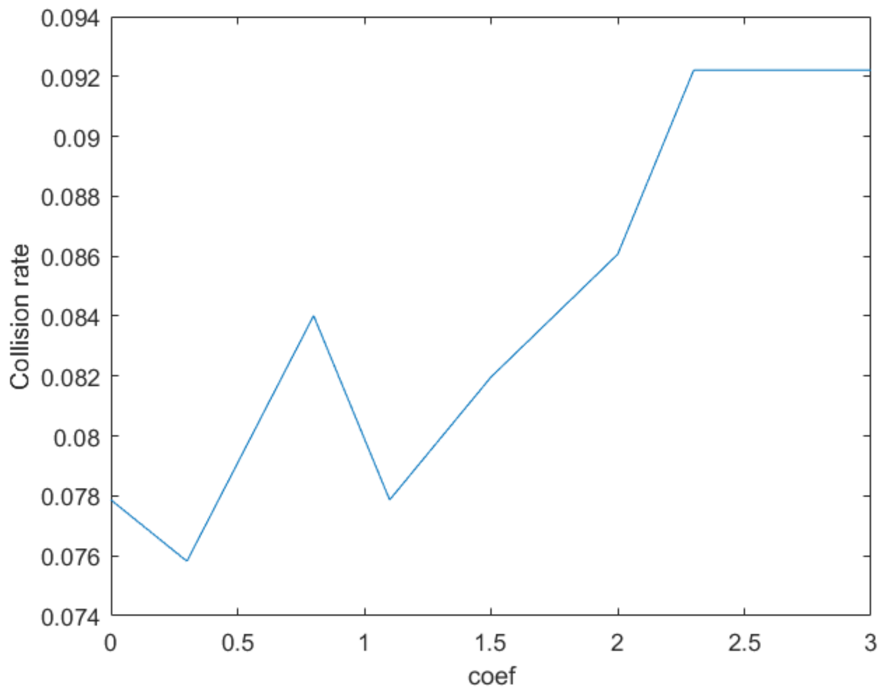


Figure 5.4: Collision rates in a subset of real data, with the given range of coefficients.

5.2.4 Oscillation

Oscillation error is also caused by the nature of the sensor, as well as the algorithm itself. White noise can be used to represent the high-frequency error. Theoretically, white noise covers oscillation at all frequencies. The only free parameter is the standard deviation. Again, as discussed in Section 5.2.3, errors larger than 2.3 meters are unlikely. Therefore, the largest magnitude of the white noise is set to 2.3 meters; in addition, the sum of low- and high-frequency error cannot exceed 2.3. There is a hard cut-off at ± 2.3.

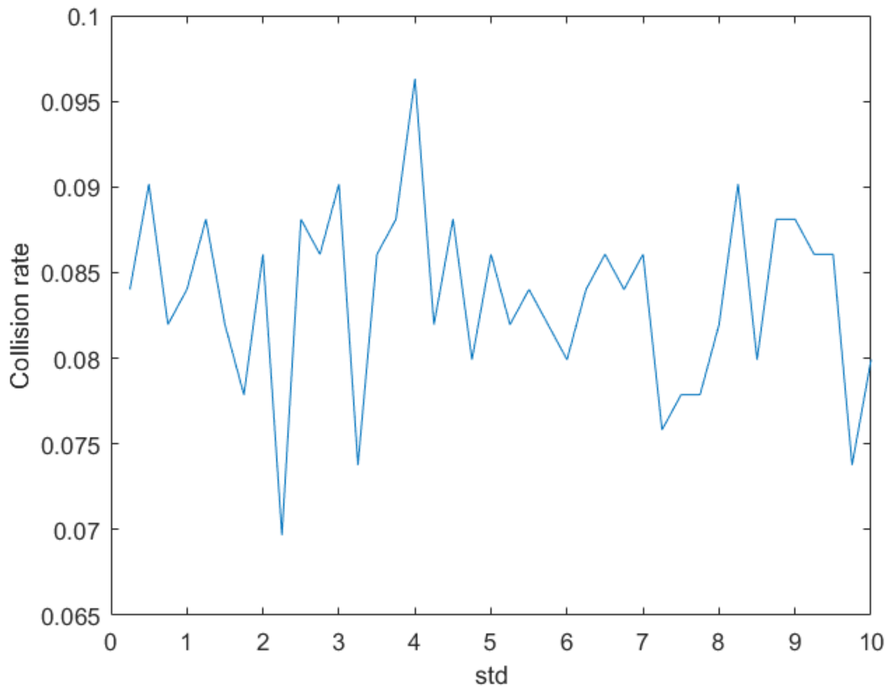


Figure 5.5: Collision rates in a subset (the third part of I-80) of real data, with σ ranging from 0 to 10m/s.

In Figure 5.5, there is no obvious impact of standard deviation magnitude on the change of collision rate. The reason is that the smoothed PGM model uses a RTS smoother, which is based on a Kalman Filter. In the Kalman filtering and smoothing processes, uncertainty is considered as Gaussian noise. And optimal states are guaranteed to be estimated through the processes. Therefore, if only white noise (zero-mean Gaussian noise) is injected into the observation, the smoothing framework can still estimate the optimal states.

5.3 Failure statistic and analysis

Figure 5.6 shows a statistical representation of the failure cases in One-on-One PGM. The failures are plotted according to the initial distance and speed difference between the host and the merging vehicles. The x-axis is the initial distance, and the y-axis is the initial speed difference. The 2-D plot is the joint distribution of the failures w.r.t. the distances and speed distances. The upper and right-hand-side 1-D distributions are the marginal distributions. The most failures occur when the merging car is about 0∼20 meters ahead of the host car and has a similar speed to the host car (or 1m/s faster).

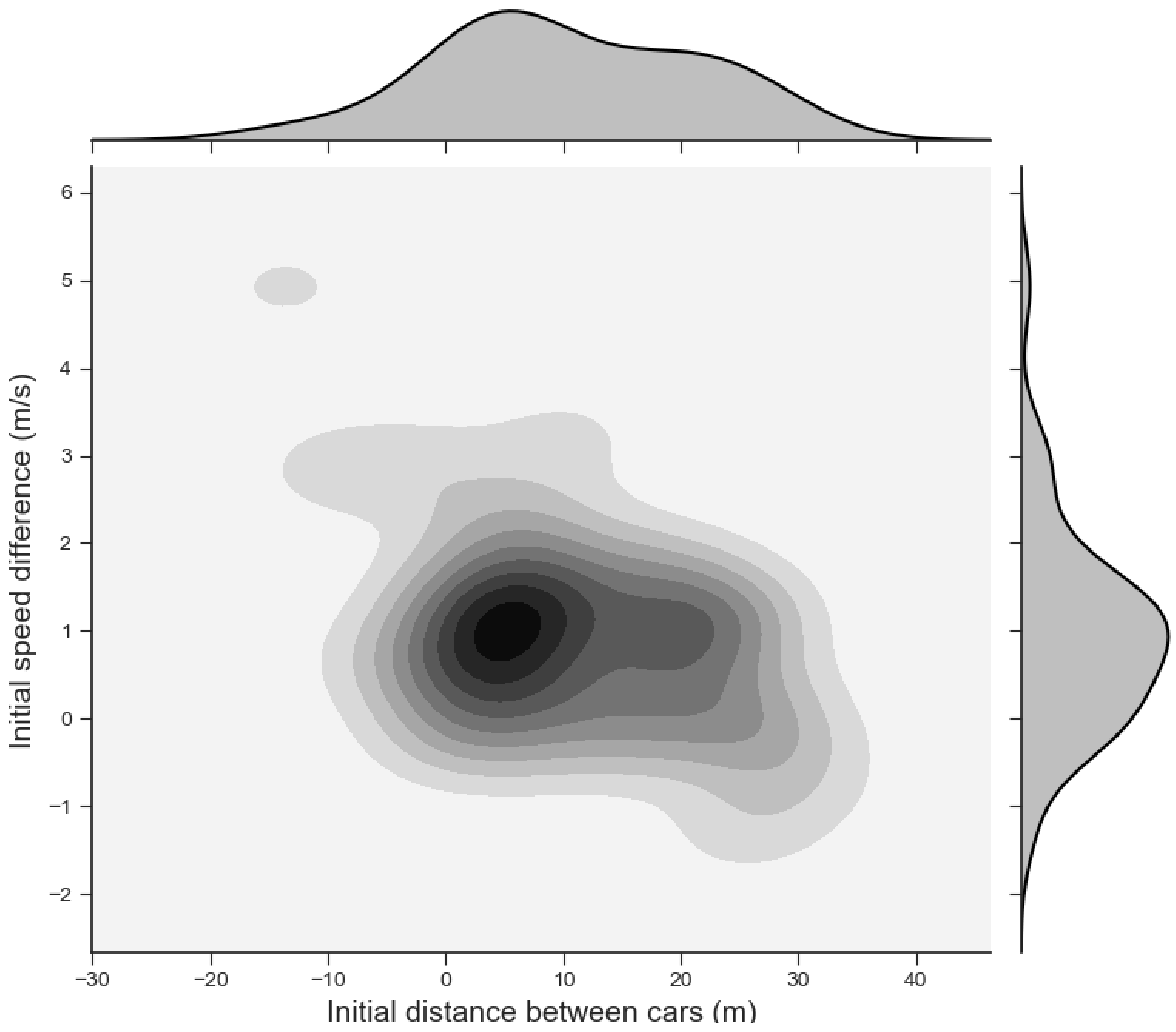


Figure 5.6: Statistical results of the failure cases.

5.3.1 An individual failure example

Figure 5.7 shows a failure example. The initial positions of the host (red) and merging car (green) are 100 meters and 90 meters to the merging point, respectively. The merging car is about 10 meters ahead of the host car. The initial speed of the host car is 4.8 m/s and the initial speed of the merging car is 6.0 m/s, which is 1.2 m/s faster than the host. The differences of the initial positions and speeds are located in the high failure probability region in Figure 5.6. As a result, the host car fails to estimate the merging car's intention, since it reduces speed in the middle of the merging ramp. However, in the last 20 meters of the ramp, it speeds up and the host car still believes that the merging car is yielding. At the end, both of the two cars increase the speed, close the gap in between and finally collide.

5.4 Practical integration issues

In this section, the integration of the behavior framework into real-world autonomous driving system is discussed. In the three-stage behavior system, an upper-level estimator is not necessary to replace the lower-level ones. Though the upper-level prediction contains more detailed information than that of the lower-level ones, the price is greater computational complexity. However, not all scenarios require the detail information. For example, in ramp merging, as is discussed in Chapter 3, there is no need to estimate the full trajectory since the route is determined. In order to save computational resources for other tasks, lower-level estimators are always preferred to the upper-level ones if they are sufficient to manage the given scenario. Compared to non-parametric regression or conditional neural process, the PGM merging control requires less computation and can run as fast as $1.8kHz$. However, the system is artificially limited to update at $10Hz$, instead of operating at the highest cycle rate and draining the computational resources. In addition, since other human drivers do not change their mind rapidly (as fast as 10Hz), it is also reasonable to limit the update rate of the behavioral estimation system.

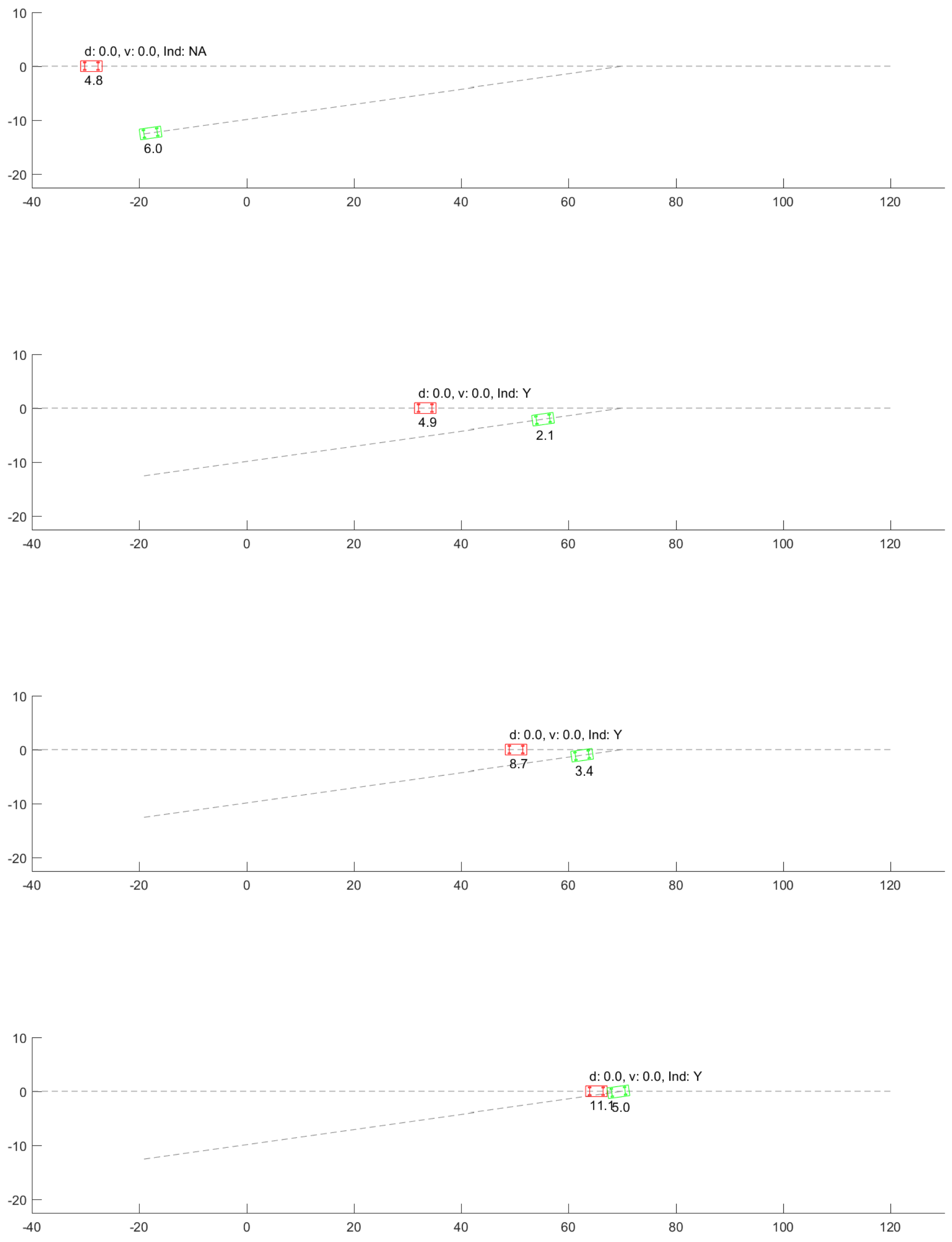


Figure 5.7: An individual example for a failure. In the initial state, the merging car is about 10 meters ahead of the host vehicle, and the speed different is 1.2m/s.

Chapter 6

Conclusion and future work

6.1 Conclusions

This dissertation introduce a system that enables autonomous driving cars to cooperate with human-driven cars in ramp merging and lane changing scenarios. The system makes it possible for the autonomous driving cars to understand the intentions of human-driven cars. The system has three stages, which are suitable for various scenarios with different complexity. Experimental results show that the three-stage system can handle ramp merging and lane changing scenarios and outperform previous solutions in terms of collision rate, distance to other human-driven cars, similarity to human drivers and updating rate. In addition, the proposed methods are capable of running in real time. Individual and real world tests demonstrate the performance of the intention estimation and trajectory prediction. Sensitivity analysis studied the robustness of PGM and its variants in merging scenarios. Multiple types of uncertainty are studied in the research. The results of the sensitivity analysis provide a guideline and requirements for perception systems.

The key contributions of this research are:

1. Introduce an efficient three-stage framework that estimate the future behaviors of one or a set of vehicles considering with their surrounding traffic and the interaction.

2. Valid the proposed system in a simulator with extended interactive scenarios and human-driven trajectories. Valid the methods in a real autonomous driving system in both private and public areas.

3. Conduct sensitivity analysis study the effects of sensing uncertainty and find the failure range. This study gives a guideline for the engineering realization of the perception system in a autonomous driving system.

6.2 Future work

In the future, road geometry and traffic conditions should be taken into consideration, especially for the ramp merging intention estimation, since the road geometry such as the length and angle of the ramp/auxiliary lane can affect the behavior of a merging car. In the non-parametric solution, there is no clear explanation about the nature of different kernels and their effect on the prediction. More sophisticated kernels can be developed to better fit the problem. For example, a network can be used to represent the analytical kernel. Such a network-based kernel will work in a classic way: given two inputs, return the “distance”. More importantly, a kind of mechanism should be identified to captures the interaction between an autonomous driving car and a human-driven car. Current methods “react” to other human-driven cars; however, they do not explicitly consider the reaction of human-driven cars if the autonomous driving car performs a certain behavior. Though PoMDP/MDP is a theoretically sound solution, there are several limitations: 1) Real time solutions of PoMDP are not guaranteed, especially for continuous state space and action space; 2) The dimensions of the state space are fixed, which is unrealistic in real traffic conditions since the number of surrounding cars keeps changing. Therefore, the biggest challenge (not limited to PoMDP/MDP) is to model with a space having a undetermined number of dimensions. Finding real time solutions for this kind of model also remains an open question.

In the practical engineering aspect, the failure modes has been discussed to figure out the boundary of the system. Currently, there is no solid theoretical bound for the whole system. To assure the safety of the system, more simulation experiments should be performed to discover the boundary of the three-level methods. In order to handle the outliers cases which can not be managed by the probabilistic methods, a hybrid system is a promising way to complement the data-driven approach. The hybrid system may switch to rule-based or other approaches from the probabilistic methods, if the scenario and vehicles' states locate in the high-danger region of the proposed approach.

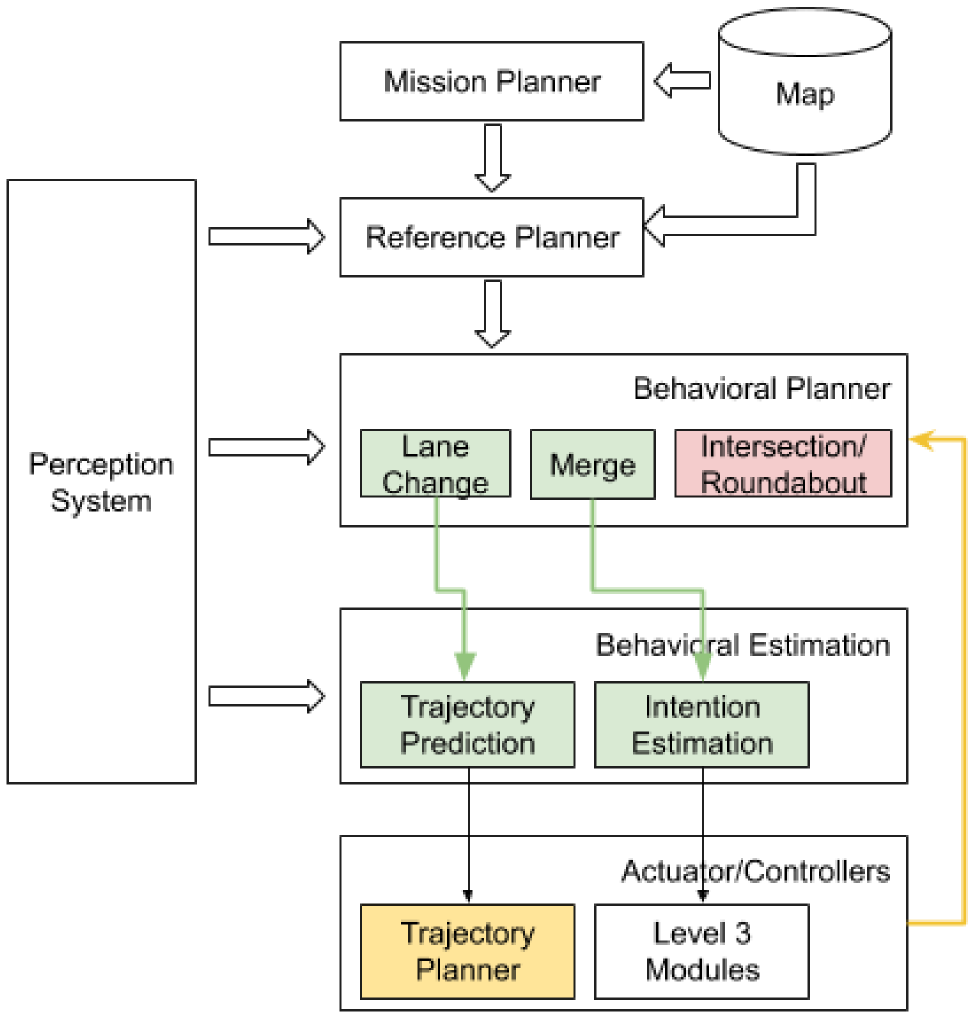


Figure 6.1: An overview of the autonomous driving system and the proposed behavior estimation subsystem.

Figure 6.1 summarized the positions and functions of the proposed method in the whole autonomous driving system. The green parts are the modules established in the dissertation. The system should close loop (marked as the yellow arrow and box following the trajectory prediction module) with a trajectory generator later to produce a feasible interactive trajectory. A monitoring system is also required as a feedback mechanism to tolerate the error from the the behavior estimator or the trajectory generator.

Appendix A

The one dimensional lane/car following model

The distance keeping model uses a Linear-Quadratic Regulator(LQR)-based controller. The state S and control u are defined as a discrete-time system:

$$S^t = \begin{pmatrix} D_t - D_* \\ V_t^{lead} - V_t^{host} \end{pmatrix}, \quad u_t = a_t$$

where D_t is the current distance between the leading and host car, D_* is the desired distance should be kept, V_t^{lead} is the leading car's speed and also the desired speed for the host car, V_t^{host} is the host car's current speed, a_t is the desired acceleration, which is to be estimated. Therefore, the system is:

$$S_{t+1} = A \cdot S_t + B \cdot u_t \tag{A.1}$$

where

$$A = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} dt^2 \\ -dt \end{pmatrix}$$

dt is the timestep.

The goal is to generate a gain K, such that

$$u^t = -K \cdot S^t \tag{A.2}$$

which achieve the minimum value for the quadratic performance index:

$$C = \sum_t S_t^T Q S_t + u_t^T R u_t \tag{A.3}$$

Where Q and R are user-defined penalties. The K in Equation A.2 is estimated by discrete-time LQR solvers, details refer to Zhou et al. [60].

Appendix B

Distribution for dropout more than one timestamp

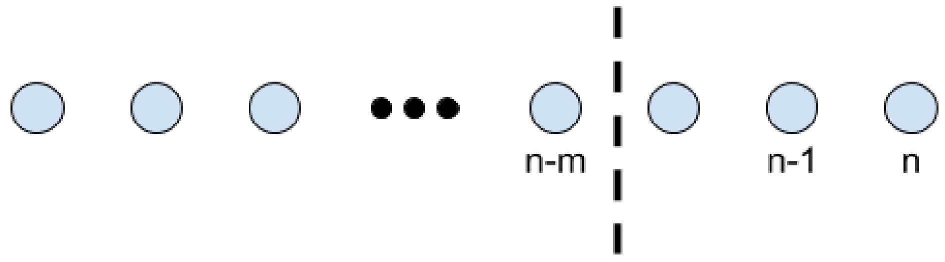
Suppose there are n timestamp, dropout period is m, and for a single timestamp, the start of a dropout is p,

$$\begin{aligned} f(k,n,m,p) = &\binom{n-m-km+1}{k} p^k(1-p)^{n-k(m+1)+1} \\ &+\sum_{l=1}^{m} \binom{n-l-(k-1)m+1}{k-1} p^k(1-p)^{n-2l-(k-2)m+1} \end{aligned} \tag{B.1}$$

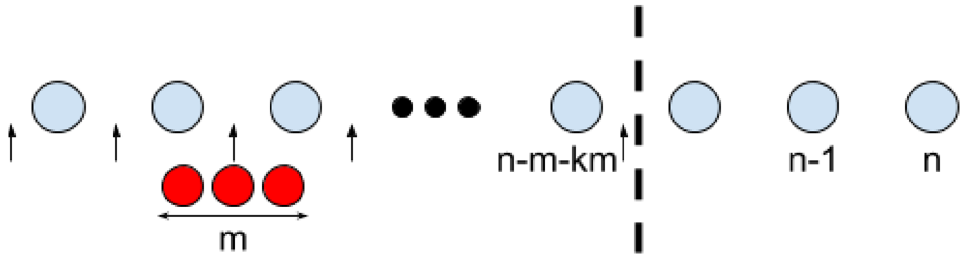
where l is the length of the last dropout period at the end of the observation. Due to the end effect, this distribution becomes more complicated. The process is considered as a sum of several mutually exclusive events. There are two category of the events:

Case 1 The last observation is not dropped, which means that the last m observations are not dropped.

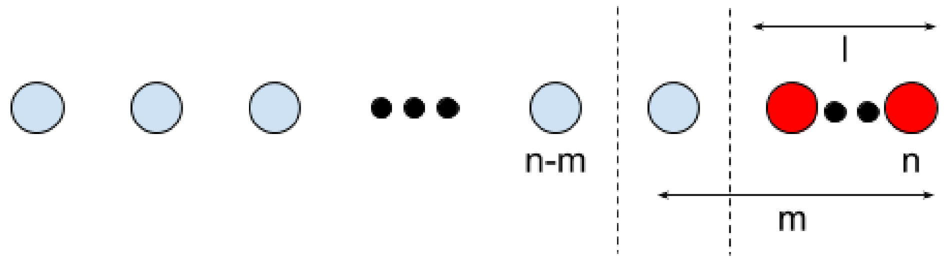
Case 2 The last observation is dropped, which means that from the end of the observation list, at most m consecutive observations are dropped. Further more, in this category, there are m mutually exclusive events which corresponding different numbers of consecutive dropped observation at the end. The number of dropout at the end is represented by l, which ranges from 1 to m.



(a) The observations and drops



(b) The first term, drop in the first n-m nodes. Red group is one of the dropout groups of observation nodes. Arrows indicates possible positions for a dropout with length m.



(c) The second term, has dropout in the last l nodes, where $l \leq m$.

Figure B.1: Dropout cases.

Figure B.1a shows one estimation cycle with n possible available observations.

Case 1.

Figure B.1b illustrate the case that is described in the first item. Say if the dropout length is m, i.e., the number of consecutive unobservable nodes, The event A_0 that corresponds to the first item is: all the last m nodes are observable and k groups of m unobservable nodes will be placed in Therefore, k groups of m dropped nodes (markded as red) will select $n - m - k \cdot m + 1$ slots, and the probability is:

$$P(A_0) = \binom{n - m - km + 1}{k} p^k (1-p)^{n-k(m+1)+1} \tag{B.2}$$

Case 2

Similarly to the case 1, but the last m nodes will contain dropout. Say the last dropout length is l, where $l \leq m$, which means that the last l nodes are dropped. The last l nodes is the k-th dropout group. Therefore, there are $k-1$ groups of m nodes still need to select $n - l - (k-1) \cdot m + 1$ slots, and the probability for a single l value is:

$$\begin{aligned} P(A_l) &= p^l (1-p)^{m-l} \cdot \\ &\qquad \binom{n - l - (k-1)m + 1}{k-1} p^{k-l} (1-p)^{n-l-(k-1)m+1} \\ &= \binom{n - l - (k-1)m + 1}{k-1} p^k (1-p)^{n-2l-(k-2)m+1} \end{aligned} \tag{B.3}$$

Since the events A_0 and A_l where $l = 1...m$ are mutually exclusive, the probability of k dropouts with the length of m in n observations is:

$$P = f(k,n,m,p) = P(A_0) + \sum_{l=1}^{m} P(A_l) \tag{B.4}$$

Substituting Equation (B.2) and (B.3) to Equation (B.4), yields Equation (B.1).

Bibliography

[1] National Highway Traffic Safety Administration (NHTSA). Automated Vehicles for Safety, 2017. 1

[2] A Aashto. Policy on geometric design of highways and streets. *American Association of State Highway and Transportation Officials, Washington, DC*, 1(990):158, 2001. 1

[3] Rudolph J Rummel. *The conflict helix.* Sage Publications, 1976. 2.1.1

[4] Aldo Rustichini. Dual or unitary system? two alternative models of decision making. *Cognitive, Affective, & Behavioral Neuroscience*, 8(4):355–362, 2008. 2.1.2

[5] Fritz Strack and Roland Deutsch. Reflective and impulsive determinants of social behavior. *Personality and social psychology review*, 8(3):220–247, 2004. 2.1.2

[6] Christopher R Baker and John M Dolan. Traffic interaction in the urban challenge: Putting boss on its best behavior. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1752–1758. IEEE, 2008. 2.2.1, 2.1

[7] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210, 2018. 2.2.1

[8] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in

urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 2.2.1, 4.1.1

[9] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008. 2.2.1

[10] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008. 2.2.1, 2.2.3

[11] Jose E Naranjo, Carlos Gonzalez, Ricardo Garcia, and Teresa De Pedro. Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver. *IEEE Transactions on Intelligent Transportation Systems*, 9(3):438–450, 2008. 2.2.1, 2.1

[12] Xiao-Yun Lu and KJ Hedrick. Longitudinal control algorithm for automated vehicle merging. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 1, pages 450–455. IEEE, 2000. 2.2.1, 2.1

[13] Julia Nilsson and Jonas Sjöberg. Strategic decision making for automated driving on two-lane, one way roads using model predictive control. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 1253–1258. IEEE, 2013. 2.2.2, 2.1

[14] 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, may 2016. 2.2.2

[15] Changliu Liu, Chung-Yen Lin, Yizhou Wang, and Masayoshi Tomizuka. Convex feasible set algorithm for constrained trajectory smoothing. In *American Control Conference (ACC), 2017*, pages 4177–4182. IEEE, 2017. 2.2.2, 2.1

[16] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000. 2.2.2

[17] Wenda Xu, Junqing Wei, John M Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2061–2067. IEEE, 2012. 2.2.2, 2.1

[18] Wen Yao, Huijing Zhao, Philippe Bonnifait, and Hongbin Zha. Lane change trajectory prediction by using recorded human driving data. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 430–436. IEEE, 2013. 2.2.3

[19] Enric Galceran, Alexander G Cunningham, Ryan M Eustice, and Edwin Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction. In *Robotics: Science and Systems*, 2015. 2.2.3, 2.3, 2.1

[20] Enric Galceran, Alexander G. Cunningham, Ryan M. Eustice, and Edwin Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, 2017. In Press. 2.2.3

[21] Chiyu Dong, John M. Dolan, and Bakhtiar Litkouhi. Intention estimation for ramp merging control in autonomous driving. In *2017 IEEE 28th Intelligent Vehicles Symposium (IV'17)*, pages 1584 – 1589, June 2017. 2.2.3, 2.1, 3.1.5, 4, 4.1.1

[22] Chiyu Dong, John M. Dolan, and Bakhtiar Litkouhi. Interactive ramp merging planning in autonomous driving: Multi-Merging leading PGM (MML-PGM). In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC) (ITSC2017)*, pages 2186–2191, October 2017. 2.2.3, 4

[23] Simon Ulbrich and Markus Maurer. Probabilistic online POMDP decision making for lane

changes in fully automated driving. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2063–2067. IEEE, oct 2013. 2.2.3

[24] Junqing Wei, John M Dolan, Jarrod M Snider, and Bakhtiar Litkouhi. A point-based mdp for robust single-lane autonomous driving behavior under uncertainties. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2586–2592. IEEE, 2011. 2.2.3

[25] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008. 2.2.3

[26] Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. Real-time decision making for large pomdps. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 450–455. Springer, 2005. 2.2.3

[27] Haoyu Bai, David Hsu, and Wee Sun Lee. Integrated perception and planning in the continuous space: A POMDP approach. *The International Journal of Robotics Research*, 33(9):1288–1302, 2014. 2.2.3

[28] K. M. Seiler, H. Kurniawati, and S. P. N. Singh. An online and approximate solver for pomdps with continuous action space. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2290–2297, May 2015. 2.2.3

[29] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3909–3917, 2016. 2.2.3

[30] E. Ward, N. Evestedt, D. Axehill, and J. Folkesson. Probabilistic model for interaction aware planning in merge scenarios. *IEEE Transactions on Intelligent Vehicles*, 2(2):133–146, June 2017. 2.2.3

[31] Stefan Klingelschmitt and Julian Eggert. Using context information and probabilistic classification for making extended long-term trajectory predictions. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 705–711. IEEE, 2015. 2.2.3

[32] Matthias Althoff and Silvia Magdici. Set-based prediction of traffic participants on arbitrary road networks. *IEEE Transactions on Intelligent Vehicles*, 1(2):187–202, 2016. 2.2.3, 2.1

[33] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 204–211. IEEE, 2017. 2.2.3

[34] Yilun Chen. Learning-based lane following and changing behaviors for autonomous vehicle. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, May 2018. 2.2.3, 3.3

[35] Shuang Su, Katharina Muelling, John Dolan, Praveen Palanisamy, and Priyantha Mudalige. Learning vehicle surrounding-aware lane-changing behavior from observed trajectories. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1412–1417. IEEE, 2018. 2.2.3

[36] 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 *Intelligent Vehicles Symposium (IV), 2018 IEEE*, pages 1233–1238. IEEE, 2018. 2.2.3

[37] Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information gathering actions over human internal state. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 66–73. IEEE, 2016. 2.2.3, 2.1

[38] Zhihai Yan, Jun Wang, and Yihuan Zhang. A game-theoretical approach to driving decision making in highway scenarios. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1221–1226. IEEE, 2018. 2.2.3

[39] Wei Zhan, Liting Sun, Yeping Hu, Jiachen Li, and Masayoshi Tomizuka. Towards a fatality-aware benchmark of probabilistic reaction prediction in highly interactive driving scenarios. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3274–3280. IEEE, 2018. 2.2.3

[40] Jiachen Li, Hengbo Ma, Wei Zhan, and Masayoshi Tomizuka. Generic probabilistic interactive situation recognition and prediction: From virtual to real. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3218–3224. IEEE, 2018. 2.2.3

[41] Tianyu Gu, Jason Atwood, Chiyu Dong, John M Dolan, and Jin-Woo Lee. Tunable and stable real-time trajectory planning for urban autonomous driving. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 250–256. IEEE, 2015. 2.1

[42] Junqing Wei, John M Dolan, and Bakhtiar Litkouhi. Autonomous vehicle social behavior for highway entrance ramp management. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 201–207. IEEE, 2013. 2.1, 3.1.1, 4.1.1

[43] Genshiro Kitagawa. Non-gaussian smoothness prior approach to irregular time series analysis. In *Adaptive Systems in Control and Signal Processing 1986*, pages 303–308. Elsevier, 1987. 3.1.5

[44] Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013. 3.1.5, 3.1.5

[45] Zita Marinho, Anca Dragan, Arun Byravan, Byron Boots, Siddhartha Srinivasa, and Geoffrey Gordon. Functional gradient motion planning in reproducing kernel hilbert spaces. pages 1–17, Jan 2016. 3.2.1, 4.4, 4.2

[46] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002. 3.2.1

[47] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012. 3.2.1, 3.2.2

[48] Charles A Micchelli and Massimiliano Pontil. On learning vector-valued functions. *Neural computation*, 17(1):177–204, 2005. 3.2.1, 3.2.2

[49] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 3.3, 4.3.1

[50] Jacob Devlin, Rudy R Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. Neural program meta-induction. In *Advances in Neural Information Processing Systems*, pages 2080–2088, 2017. 3.3

[51] Chiyu Dong, John M Dolan, and Bakhtiar Litkouhi. Smooth behavioral estimation for ramp merging control in autonomous driving. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018. 4

[52] Chiyu Dong, Yihuan Zhang, and John M. Dolan. Lane-change social behavior generator for autonomous driving car by non-parametric regression in reproducing kernel hilbert space. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4489–4494, Sept 2017. 4, 4.3.1

[53] Chiyu Dong, John M. Dolan, and Bakhtiar Litkouhi. Continuous behavioral prediction in lane-change for autonomous driving cars in dynamic environments. In *2018 IEEE 21th International Conference on Intelligent Transportation Systems (ITSC) (ITSC2018)*, 2018. 4

[54] Nicholas J Garber and Lester A Hoel. *Traffic and highway engineering*. Cengage Learning, 2014. 4.1

[55] Alexander G Cunningham, Enric Galceran, Ryan M Eustice, and Edwin Olson. MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1670–1677. IEEE, 2015. 4.1.4

[56] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997. 4.1.4

[57] C. Wang, C. Thorpe, and A. Suppe. Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds. In *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683)*, pages 416–421, June 2003. 5.2

[58] C. Fu, P. Hu, C. Dong, C. Mertz, and J. M. Dolan. Camera-based semantic enhanced vehicle segmentation for planar lidar. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3805–3810, Nov 2018. 5.2

[59] Xiao Zhang, Wenda Xu, Chiyu Dong, and John M. Dolan. Efficient l-shape fitting for vehicle detection using laser scanners. In *2017 IEEE Intelligent Vehicles Symposium*, June 2017. 5.2

[60] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996. A