Reinforcement Learning for Behavior Planning of Autonomous Vehicles in Urban Scenarios

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Department of Electrical and Computer Engineering

Zhiqian Qiao

B.S., Mechanical Engineering, Shanghai Jiao Tong University
M.S., Mechanical Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

August 2021

(Note: This is an unnumbered page)

Acknowledgments

I would like to express my deepest appreciation to my committee: Professor John M. Dolan, Professor Jeff Schneider, Professor Marios Savvides and Doctor Katharina Muelling. I would also like to extend my deepest gratitude to my supervisor Professor John M. Dolan. This thesis would not have been possible without the support and nurturing of Professor John M. Dolan.

I would like to thanks for General Motors-Carnegie Mellon Connected and Autonomous Driving Collaborative Research Lab (GM-CMU CAD-CRL) and Argo AI Center for funding my Ph.D. study. I would like to extend my sincere thanks to all colleagues in CMU Argo AI Center for Autonomous Vehicle Research and colleagues in GM-CMU CAD-CRL: Dr. Priyantha Mudalige, Praveen Palanisamy, Dr. S Bilal Mehdi and Zachariah Tyree.

I also had great pleasure of working with all my labmates: Chen Fu, Yiwei Lyu Adam Villaflor, Qin Lin, Zherui Zhang and my friends: Wenhao Luo and Wenkun Zhen. During the hard time of the last year of my PhD study due to Covid19, we had fun and memorable days together to get through it. I thank my colleagues in the department of Electrical and Computer Engineering, the Robotics Institute and Mechanical Engineering.

Lastly, I'd like to recognize the assistance that I received from my parents' unconditional support, which encouraged me to finish my PhD in general.

Abstract

How autonomous vehicles and human drivers share public transportation systems is an important problem, as fully automatic transportation environments are still a long way off. Behavioral decision making serves as a key link in autonomous driving technology. Within conventional self-driving technology, heuristic-based rules-enumeration methods fulfill major tasks for behavioral decision making. However, for such a complex behavior as driving, the development of a suitable set of rules is a laborious engineering task that does not guarantee an optimal policy. Reinforcement learning (RL) is a decision-making method with strong recent successes that is capable of solving for an optimal policy, and can map diverse observations to actions in a variety of complex situations. However, RL has its own problems, such as exceptionally long training times, unstable training results and difficult reward tuning.

In this thesis, we present a series of behavior planning structures and algorithms that are based on the advantages coming from both reinforcement learning and heuristic-based rules-enumeration. The resultant contributions include:

- Creation of an Automatically Generated Curriculum in order to increase the learning speed for RL.
- Improvement of the policy network of RL with an LSTM module in order to get better performance on a given task.
- Creation of a hierarchical RL structure with hybrid reward mechanism which can accomplish the behavior decision procedure with the help of heuristic-based methods.
- Application of the hierarchical RL structure to a comprehensive range of urban intersection scenarios, to include approaching, observation, and traversing.

Compared to traditional heuristic-based rules-enumeration methods, which need a large amount of effort to design rules which can cover as many scenarios as possible, reinforcement learning can help to learn such an optimal policy automatically. On the other hand, our algorithm can help RL to be more sample-efficient and converges to an optimal policy faster than competing algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 From human driver to autonomous vehicle

An autonomous vehicle (AV) consists of three main systems: 1) Perception, 2) Planning and 3) Control. By analogy with a human driver, perception can be compared to the human's eyes, planning to the human's brain and consciousness, and control to the human's arms and legs. Figure 1.1 presents the architecture of an AV system and a human driver. Generally, the whole driving system needs the sensors or human eyes to detect the environment and localize the vehicle. Then according to the map information and the prediction of the events happening, the route planner

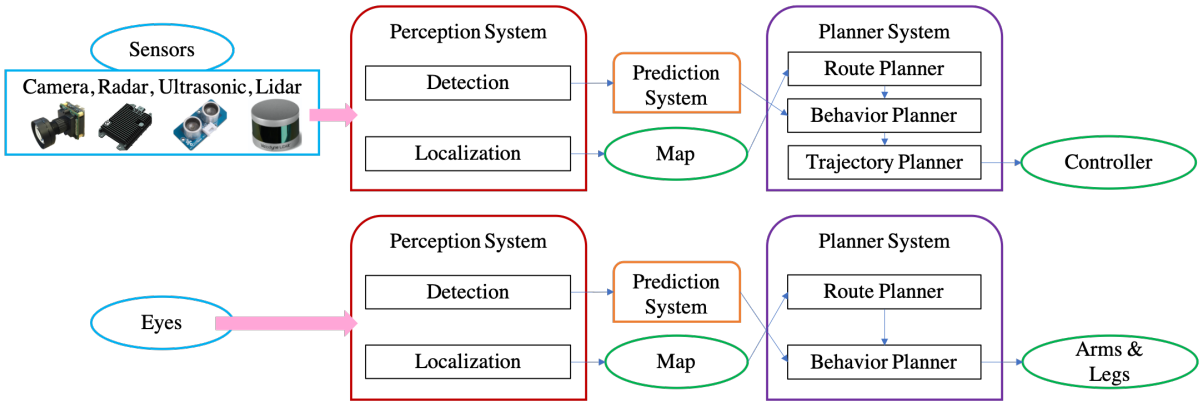

Figure 1.1: Comparison between autonomous vehicle architecture and human driver

specifies the route from starting point to destination. The behavior planner then makes behavior-level decisions for driving, like merging into the exit lane, turning left, etc. Then the autonomous vehicle needs to generate the trajectory for the control system to follow, whereas human drivers can control the car directly using their limbs, perceptions, and experience. The human brain and consciousness are mysterious in many ways and contain various phenomena and characteristics that are difficult to explain. Similarly, the planning part of driving is difficult, especially for new human drivers, due to lack of experience.

In order to design a good behavior decision-making system for autonomous vehicles, an understanding of how human drivers succeed or fail in making decisions while driving is helpful. An economics perspective, in which the decision making model is based on heuristics and bias [1], can be helpful here. This model identifies the cognitive biases of decision-makers, in this case drivers. It assumes that decision makers develop heuristic rules based on their past experience. These heuristics help to overcome the pressure and constraints coming from limited time in which to make a decision, which will lead to biased decisions based on the indidividual personal experience. As a result, driving is a skill for which practice makes perfect. Most decisions made during driving rely on experience or "prior knowledge" coming from what drivers have been taught or what they have learned during practise while encountering a fast-changing driving environment. For a healthy new driver who can perceive the surrounding environment clearly and operate the steering wheel and pedals smoothly, the difficulties of driving normally come from two aspects:

1. How to get a large amount of experience so that the drivers can use the correct heuristic skills when encountering various situations.
2. How to apply prior knowledge to generate the correct response even under the pressure of limited time.

In general, when we design the decision making module of an autonomous vehicle, it needs to overcome these two difficulties for human drivers so that the system is capable of generaliz-

ing the given prior knowledge in order to tackle a rapidly changing and unpredictable driving environment.

1.2 Behavior decision making of autonomous driving

For the reasons described, a good autonomous vehicle decision-making system should have:

- adequate prior knowledge to deploy driving skills, handle various possible driving scenarios, and obey traffic rules;
- flexible behavioral response in a complex and fast-changing environment;
- computational efficiency.

In previous work, researchers have proposed various methods for autonomous vehicle decision making which fit into three main categories: 1) rules enumeration based on heuristics; 2) imitation learning based on prior knowledge of expert drivers; and 3) reinforcement learning through trial and error in simulation. The three types of approach gradually decrease the dependence on pre-collected driving data from expert human drivers.

Heuristic-based rules development. This method entirely depends on prior knowledge gained from expert drivers. Researchers generalize a series of driving rules from human experience to be a basis for autonomous vehicle behavior. Some of the rules are at the behavior level, such as slowing down and stopping at a stop-sign. Others are more detailed, such as trajectory and controller planning in order to drive smoothly when distance-keeping to a leading vehicle. It is laborious and time-consuming to create a set of rules that can make the autonomous vehicle capable of successfully handling the entire range of relevant driving situations.

Imitation learning. In order to lessen the work of designing a large number of rules, an alternative method is to use imitation learning to create a driver model that is similar to human experts. This method can significantly reduce the work of rules enumeration by training a model automatically. With a sufficiently large dataset, the autonomous vehicle can perform decision

making in a way similar to that of the human driver model which is derived from the data. However, this method is not flexible enough to handle scenarios not included in the training dataset. For example, how to perform a lane-change and how to traverse an intersection requires two separate datasets and training processes.

Reinforcement learning. The third method uses reinforcement learning (RL), which through trial and error is able to solve for an optimal policy that can map various observations to corresponding actions in complicated scenarios by training in simulation. This method also relies on prior knowledge of driving from human experts in order to design the reward function, but this requires small effort compared to rules development. The main advantage is the algorithm's flexibility in a wide range of scenarios, since the simulation can generate various scenarios during the training process. However, low stability and heavy computational requirements make RL difficult to use for general tasks in the autonomous vehicle domain. In most previous work applying RL to autonomous driving, RL is used for learning throttle and steering angle for various scenarios or only learning behavior decisions at the higher level of a planning system and then applying traditional control algorithms at the lower level.

The shortcomings of reinforcement learning methods can be summarized as:

- Low stability: Most existing reinforcement learning methods can only be applied to simple autonomous driving scenarios. When the driving scenarios become more complicated, it is hard to get stable performance, i.e., different and inconsistent actions may be generated in the face of similar conditions.
- Single-scenario: Transferring reinforcement learning results from one autonomous driving scenario to a totally different scenario is difficult. In particular, reward functions need to be re-designed from scratch. Transfer learning algorithms can help to expand a given scenario to another one with similar attributes, such as expanding single-lane driving behavior to two lanes. However, transferring between lane-change and yield-to-obstacle is typically not feasible.

- Inefficiency: For driving scenarios with changing environments which include unpredictably interacting vehicles, learning from scratch with RL requires a large amount of learning effort to converge to a satisfactory performance. In some cases, even thousands of trials may not give acceptable performance.

In order to overcome these shortcomings, and meanwhile to prevent dependence on human driver data collection, our work proposes corresponding solutions:

- Curriculum learning: In order to ensure good stability for RL algorithms as well as flexibility in handling complicated scenarios under fast-changing environments like urban intersections, we propose to use curriculum learning to initially learn simple scenarios, then progress to more complex scenarios during the learning procedure.
- Hierarchical structure: We propose a hierarchical hybrid RL structure which uses RL to generate optimal decisions at different planning levels, which can help to address complex driving scenarios, including various sub-scenarios. For example, if the ego vehicle is blocked by a front vehicle, the system is capable of simultaneously making higher-level behavioral decisions such as making a lane change or slowing down behind a leading vehicle and low-level throttle, braking, and steering angle actions based on the behavioral decision and environments. This kind of modularity makes the learning process more efficient and makes it easier to validate the safety of each module.
- Heuristic exploration: We design heuristic-based exploration for reinforcement learning-based methods such that the autonomous vehicle can quickly access successful policies in the initial learning stages, which significantly reduces learning time.

Scenarios arising at urban intersections are particularly suitable for the application of the resultant RL techniques. According to research, every year 40% of all crashes in the USA occur at intersections. Especially at intersections that are not controlled by a traffic light, the driver has to monitor all vehicles continuously and to estimate their intentions and velocities. Decision making in such scenarios is hard even for expert human drivers. Multiple decisions need to be

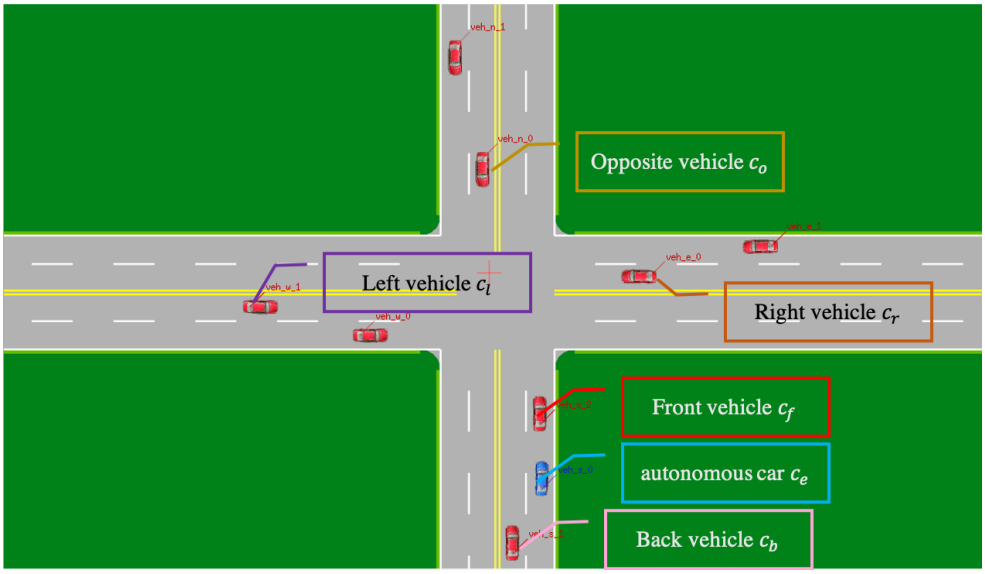

Figure 1.2: Intersection scenario for autonomous driving

made in a flash. In Figure 1.2, for example, when the blue vehicle is approaching an intersection and wants to go straight, the driver needs to maneuver appropriately to avoid crashing into the vehicle in front of it. Sometimes during rush hour, vehicles which want to turn right or left may block the traffic so that a car waiting behind wanting to go straight needs to make a lane-change or yield in order to go through. Meanwhile, the driver then needs to stop at the stop-line if necessary and then slowly approach the intersection so that it will not block it or crash into other vehicles approaching from other directions. Subsequently, the driver needs to decide when to begin traversing the intersection. As the traffic increases and the desire of the drivers to make fast progress grows, it is easy to make a mistake in any of these decisions that can lead to an accident.

1.3 Problem statement and expected contributions

Previous work has presented various solutions to the problem of autonomous driving behavior planning. Most of them can be categorized as heuristic-based or learning-based methods. The advantage of heuristic-based methods is that one can design straightforward rules intuitively and

can easily determine and explain the reasons for failure cases. However, designing and refining such an algorithm is labor-intensive. On the other hand, a learning-based algorithm doesn't depend on rules, but instead automatically learns a policy. However, most machine-learning algorithms, especially end-to-end methods, have problems with stability and universality for complicated scenarios.

In this work, we want to build a behavior planner system which can combine the clarity and explainability of rule-based methods with the power and flexibility of hierarchical learning-based methods. As a result, we propose methods for addressing the three main problems in previous work:

1. How does one make architectural changes to existing reinforcement learning algorithms in order to improve the learning performance as well as to be more computationally efficient?
2. How does one embed heuristic-based rules into a learning-based system in order to reduce the time and labor of designing and generalizing all the rules?
3. How does one design a decision making system that is able to deal with the broad range of driving scenarios which may be encountered, especially at urban intersections?

The learning-based method used in this thesis to solve the behavior planning problem is reinforcement learning (RL). Using reward functions designed based on drivers' experience, the resultant algorithms can automatically improve the behavior planner policy during the training process. The main topic of this dissertation is to determine such policies for specific urban scenarios, leading to the following contributions:

1. Creation of an Automatically Generated Curriculum in order to increase the learning speed for RL.
2. Improvement of the policy network of RL with an LSTM module in order to get better performance on a given task.
3. Creation of a hierarchical RL structure with hybrid reward mechanism which can accomplish the behavior decision procedure with the help of heuristic-based methods.

4. Application of the proposed hierarchical RL structure to a comprehensive range of urban intersection scenarios, to include approaching, observation, and traversing.

The remainder of the thesis is structured as follows. Chapter 2 describes previous related work, including work on various autonomous vehicle behavior planners. Chapter 3 describes the thesis' proposed algorithms, which implement effective behavior planning for self-driving. Chapter 4 presents simulation experiments based on the methods described in Chapter 3 and corresponding results. Chapter 5 covers the real-world data collection system and the experiments extracted from that. Finally, Chapter 6 presents conclusions.

Chapter 2

Related Work

2.1 Decision making of Human drivers

Understanding how human drivers make decisions while driving can provide heuristics for autonomous vehicles. For humans, the rational approach to decision-making emphasizes the use of logic and complete rationality [2]. As described in Chapter 1, the behavioral approach towards decision-making emphasizes that the decision makers are prone to varying degrees of rationality while making decisions, which makes it hard to ensure complete rationality at all times. The different levels of rationality in decisions lead to various models of behavioral decision-making.

According to economic theory, the model with the highest level of rationality is economic rationality [3], which assumes that the decision-maker is perfectly rational and is aware of all the available alternatives and their permutations and combinations while making decisions. This model's requirements are too stringent to apply to human drivers in realistic, real-world driving situations. A less rational model is the bounded rationality model [4], which attempts to deal with less ideal circumstances. The decision maker gets by with simple understandings and perceptions of the problem, rather than fully analyzing all the possible situations. This process still ignores the personal bias of the decision makers. In the next level, a decision under pressure of limited time and changing environment can be applied, the heuristic and biases model [1], which

identifies the cognitive biases of decision-makers. It assumes that the decision makers develop heuristic rules based on their past experience. These heuristics help to overcome the pressure and constraints coming from the limited time which will lead to biased decisions based on personal experience.

The heuristic-based human decision making model provides an intuitive basis for an autonomous vehicle behavior planner. In our work, we apply this kind of heuristic-based decision making model for rules-enumeration as well as an exploration basis for the reinforcement learning which can be used to speed up the learning process.

2.2 Behavior Planner of Autonomous Vehicle

As introduced in Chapter 1, the autonomous driving planner module can be broken down into three planners:

1. **Route planner**: According to the starting point or current position A and the destination B, the planner uses the map information to determine a route based on requirements like: fastest route, shortest route or highway-preferred.
2. **Behavior planner**: When following the given route, the vehicle's behavior planner needs to make some behavior-level decisions like turn-left, lane-change, follow-front-vehicle, etc.
3. **Trajectory planner**: After making a behavior decision, the vehicle needs to plan a detailed trajectory for some period of future time so that the controller can control the vehicle to follow the trajectory.

This thesis focuses on the behavior planner of the autonomous vehicle, and this chapter describes related work. As introduced in Section 1.2, previous approaches proposed for the decision making process of autonomous vehicles can be categorized into three types:

- Heuristic-based rules enumeration

- Imitation learning
- Reinforcement learning

2.2.1 Heuristic-based rules enumeration

Some previous work related to behavioral decision-making applied heuristics to the behavior planner of autonomous vehicles. For example, [5] proposed a slot-based approach to check if it is safe to merge into lanes or across an intersection with moving traffic. This method is based on the information of slots available for merging, which includes the size of the slot in the target lane, and the distance between the ego-vehicle and front vehicle. Time-to-collision (TTC) [6] is a heuristic-based algorithm which has normally been applied to intersection scenarios as a baseline algorithm. Fuzzy logic is also a very popular heuristic-based approach to model the decision making and behavior planning of autonomous vehicles. In contrast to vanilla heuristic-based algorithms, fuzzy logic allows adding the uncertainty of the results into the decision process. [7] used a fuzzy logic method to control the traffic flow in urban intersection scenarios, where the vehicles have access to environment information via a vehicle-to-vehicle (V2V) system, which has only been applied to a small number of public roads and vehicles. In [8], the researchers developed a fuzzy logic method for the application of steering control in roundabout scenarios. The proposed heuristic-based methods rely heavily on the parameter tuning and each set of parameters is restricted to the corresponding scenarios and environments. The use of heuristics alone makes it hard to make the algorithm sufficiently general when designing a high-performance autonomous vehicle behavior planning system. Especially for complex urban scenarios, it is laborious and time-consuming to develop a set of rules with or without advanced technology which can cover all possible cases.

2.2.2 Imitation learning

Imitation learning is an alternative method for a self-driving behavior planner that requires a large amount of data collected from human expert drivers. The fundamental imitation learning algorithms can be categorized into behavior cloning, direct policy learning and inverse reinforcement learning. Most behavior cloning approaches for self-driving involve collecting sensor data from an autonomous vehicle and mapping them to actions such as throttle, brake and steering angle by expert human drivers. [9] and [10] are examples of using behavior cloning in order to mimic human driver data collected from real-world cars or high-fidelity simulations. This kind of supervised learning algorithm can work well if the state-action pairs during testing are similar to those during training and the assumption is met that the state-action pairs are independent and identically distributed. However, for most driving scenarios, the driving process is a Markov Decision Process in which the current state relies on the previous state, so that the error for generating an action will continuously increase for an action generated by behavior cloning only. Direct policy learning can improve behavior cloning by adding an interactive demonstrator during training time. DAgger (Dataset Aggregation) [11] was proposed to train the actual policy on all the previous training data and policy aggregation trains on-policy on the training data received in the last iteration, then blends them together during the roll-out. [12] and [13] applied DAgger in two different autonomous vehicle planning simulations. A more advanced method is inverse reinforcement learning (IRL) [14], which tries to learn the reward function of the environment based on the expert's demonstrations instead of the expert's actions directly. Reinforcement learning is then applied to learn the optimal policy based on the learned reward. [15] modeled the interaction between autonomous vehicles and human drivers by the method of IRL in a simulated environment. The work simulated autonomous vehicles to motivate human drivers' reactions and acquired reward functions in order to plan better decisions while controlling autonomous vehicles. It is obvious that IRL can be a good method for autonomous vehicle planning. However, the method needs two processes of reward learning and reinforcement learn-

ing, which leads to difficult convergence during the training process. If the reward function is not difficult to get from experts or it can be easily gotten from the heuristic rules described before, reinforcement learning is easier to apply to the self-driving problem.

2.2.3 Reinforcement learning

Reinforcement learning (RL) has been applied to transfer multiple rules into a mapping function or neural network. According to the previous section, this work will focus on the application of RL to the behavior planner of the autonomous vehicle. Recently, various DRL algorithms have shown advantages in handling high-dimensional sensory inputs with discrete or continuous actions. Deep Q-Networks [16] (DQN) and its variants have been successfully applied to various fields. However, DQN is restricted to the discrete action domain, which makes it hard to apply to autonomous driving. DDPG [17], which was proposed by Lillicrap et al., adapted the ideas underlying the success of DQN to the continuous action domain. The algorithm has shown excellent performance in finding optimal policies and is competitive with other planning algorithms which need full access to the dynamics of the system. All these solutions have in common that they model the problem as an MDP, i.e., they assume full knowledge of the environment, which is often not true in the real world. As a result, based on the requirements for autonomous vehicles, some advanced techniques based on RL have been proposed which can be be summarized under three headings:

1. Acceleration of RL
2. LSTM-based RL structure
3. Hierarchical RL

1. Acceleration of reinforcement learning

For most reinforcement learning algorithms, a major challenge is the long training time needed for the learning process. The goal is for the computational time for training to be less than

the effort required to tune rules one by one in heuristic-based rules enumeration. Curriculum learning [18] was proposed to speed up the learning process by first training the system on easy tasks and then gradually increasing the complexity of the tasks presented to the learning agent. Most of the work on curriculum learning has focused on simple scenarios which can be solved by hand-designed curricula [19]. For more complex tasks, however, hand-designing curricula is a difficult problem. Florensa et al. [20] proposed a curriculum generation method for reinforcement learning which trains the robot in reverse sequence: starting the learning process from an initial position close to the destination, it gradually increases the distance towards a random start configuration. However, the results are mainly based on static scenarios and not on changeable environments. In contrast, the complexity of autonomous vehicle problems is based on the different scenarios resulting from other vehicles' interactions. Therefore, the reverse curriculum approach cannot be applied directly in our work. [21] introduced an approach which can generate a curriculum as a directed acyclic graph instead and did experiments on the agents using RL. [22] designed a Teacher-Student Curriculum which learns a curriculum by supervised learning or reinforcement learning in order to complete some tasks which cannot be finished if trained directly without a particular curriculum. Most of the reported work hasn't been applied to the autonomous vehicle planning problem, which is more complicated and contains greater numbers of states and action choices. We therefore design some difficult cases at urban intersection scenarios for autonomous vehicles and propose a novel approach, Automatically Generated Curriculum (AGC)-based RL, which can help to solve tasks with high-dimensional state spaces with fewer training iterations based on DRL. We use a neural network to approximate the policy function and apply an automatically generated curriculum-based RL method to learn the optimal behavior policy and decrease the training time and number of training iterations.

2. LSTM-based RL structure

Some previous work has shown the capability of controlling the autonomous vehicle to finish some simple tasks via DQN or DDPG. However, in the same scenarios, if the observation of the AV is occluded during navigation for some time, the results worsen significantly. Modeling the problem as a Partially Observable Markov Decision Process (POMDP) [23] allows the algorithm to consider these uncertainties in the decision process, which makes it more robust to real sensor characteristics. [24] proposed to extend the DQN method for POMDP problems by learning the optimal policy of a model-based RL problem. They used a fully-connected network to approximate the mapping function from observations and belief vectors to Q-values for different actions in order to solve a POMDP problem through DQN. [25] proposed a method called Deep Recurrent Q-network (DRQN) which can be applied to a model-free representation by adding LSTM layers in the policy network. The trained policy network is capable of capturing all the historical information in the LSTM layer and the output actions are based on all the historical observations with the help of LSTM layers. Based on DRQN, [26] proposed an algorithm called Deep Distributed Recurrent Q-networks (DDRQN) which not only used the previous observations but also previous actions as the inputs. The results show the first success in learning communication protocols by RL. [27] is a recent method called Action-specific Deep Recurrent Q-Network (ADRQN), which also used historical observations and actions as input to get an optimal policy. However, instead of inputting all historical information into the LSTM layer, they used paired last-step action and current-step observations as inputs into the LSTM layer for training, which showed improved results on some POMDP tasks. [28] formulated the decision-making problem for autonomous vehicles under uncertain environments as a POMDP and trained a Bayesian Network representation to deal with a T-shape intersection merging problem. [29] dealt with the traversing problem via Deep Q-Networks combined with a long-term memory component. They trained a state-action function Q to allow an autonomous vehicle to traverse intersections with moving traffic. [30] used Deep Recurrent Q-network (DRQN) with states from a bird's-eye

view of the intersection to learn a policy for traversing the intersection. [31] proposed an efficient strategy to navigate through intersections with occlusions by using the DRL method. The previous algorithms have not extended from discrete to continuous actions, especially while controlling the pedal and steering of the AV. In part of our work, we extend the original POMDP with LSTM [30] structure to an application that can generate continuous instead of discrete actions and modify the LSTM-based actor-critic network structure of reinforcement learning so that it applies continuous actions during the training process.

3. Application of hierarchical RL

Based on the context of RL, compared to the POMDP model, a hierarchical model is easier to train with less complicated model structure and meanwhile can handle more complicated scenarios than can the vanilla RL by breaking the initial large network into several smaller networks. [32] proposed the idea of options to generate actions at different levels. The options are used to define a policy governing when the action policy is initialized and terminated. [33] introduced the concept of hierarchical Q learning called MAXQ. They proved the convergence of MAXQ mathematically and showed lower computing time than the original Q learning experimentally. [34] proposed an improved MAXQ method by combining the R-MAX [35] algorithm with MAXQ. It has both the efficient model-based exploration of R-MAX and the opportunities for abstraction provided by the MAXQ framework. [36] used the idea of the hierarchical model and transferred it into parameterized action representations. They use a DRL algorithm to train high-level parameterized actions [37] and low-level actions together in order to get more stable results than by getting the continuous actions directly. [38] presented hierarchical DQN (h-DQN), a method which is based on the structure of DQN. H-DQN adds another high-level Q-value function to learn a high-level policy while the DQN is responsible for the low-level policy. The algorithm is effective on the sparse and delayed feedback of ATARI games.

Most previous work designs a single hierarchical structure which can be used to solve the en-

tire problem. However, in most real-world cases, a complicated task such as behavior planning for an autonomous vehicle may be composed of several sub-tasks. As a result, we propose to build the HRL-structure according to the heuristic method so that the system can more easily figure out the local optimal policy based on local option choice and environment state. Meanwhile, it can allow the validation of different local policies as a sub-function with full capabilities within the hierarchical system instead of presenting a monolithic neural-network black-box policy.

2.3 Summary

Table 2.1 summarizes the advantages and disadvantages of former heuristic-based and learning-based methods and compares them with our proposed approach to the autonomous driving behavior problem. The four respects in which the methods are compared are as follows:

- **Validation**: ease of validation among different situations.
- **Labor Efficiency**: degree of human design effort needed to implement the method.
- **Computational Efficiency**: degree of computational efficiency in getting the algorithm results.
- **Universality**: degree of applicability to various scenarios.

We rank the different algorithms by number, where 1 represents worst performance and 5 means best performance. On the one hand, heuristic-based methods always involve the biases inherent in different decision-makers and are constrained by the labor needed to enumerate more and more rules in order to improve the approaches. On the other hand, learning-based methods also involve human biases during the reward function design process and are hard to validate among various situations because the policy network is more like a black box whose operation is difficult to explain.

In order to alleviate the disadvantages of previous methods, instead of using demonstration data directly, we include a heuristic-based rules-enumeration policy during the exploration pro-

Table 2.1: Comparisons between different methods. 1 means worst and 5 means best for the corresponding criteria.

Method	Validation	Labor Efficiency	Computational Efficiency	Universality
Heuristic-based	5	1	5	4
RL [16][17][39]	1	3	2	1
RL with curriculum (ours)	4	3	3	2
RL with LSTM (ours)	1	4	1	2
RL with hierarchical network (ours)	3	4	4	4
HRL	3	4	4	3
Hybrid HRL (ours)	5	5	5	5

cess which can save a large amount of computation time for human model extraction, as well as initiate the learning process more quickly. We build the HRL-structure according to the heuristic method so that the system can adjust the exploration rate according to the training results in real time and meanwhile can more easily figure out the local optimal policy based on the environment.

Chapter 3

Methods

3.1 Overview

In this chapter, we introduce the methodologies used in this thesis for the autonomous vehicle behavior planner system. Both a heuristic-based rules-enumeration system and reinforcement-learning-based methods are described and applied to the decision making system of autonomous vehicles.

Previous work makes clear that rules-enumeration systems are able to deal with particular scenarios in a straightforward way. A large amount of experience from human drivers can be transformed into multiple rules that the behavior planner of the self-driving car will follow. A challenge of this kind of method is how to transfer the vast experience efficiently into a set of enumerable rules. This is where reinforcement learning (RL) can come into play. An RL-based system is capable of automatically transferring driving experience into a general policy which can avoid the enumeration process of the rules-based system. But the learning-based method also brings new challenges such as computational efficiency, learning speed, and how the transformation works.

As a result, our approach is to combine the two in a way that keeps the advantages and lessens the disadvantages of each method. In response to the shortcomings of learning-based systems

described in the previous chapter, this chapter describes approaches which improve current reinforcement learning algorithms in three main respects:

1. **Automatically Generated Curriculum (AGC)** RL methods still have convergence difficulties with high-dimensional state space problems, such as autonomous vehicle decision-making in urban environments. The main objective of our AGC approach is to apply deep reinforcement learning (DRL) methods which can generate a curriculum sequence to accelerate the training process for high-dimensional reinforcement learning problems with pre-defined tasks.

2. **LSTM-embedded module** For some self-driving scenarios, POMDP can describe the problem better then MDP. In such cases, a LSTM-module in the policy network of the reinforcement learning problem can help the agent to memorize part of the historical states and output an action under the current state with more prior knowledge. The proposed method designs a LSTM-embedded actor-critic network which can generate continuous actions based on the idea of DDPG.

3. **Hierarchical structure with sub-goals** Applying a neural network as a policy network or value function network in reinforcement learning algorithms can simplify policy design or expected rewards prediction. However, the network is like a black box when attempting to validate different situations. For a similar alternative task, traditional RL needs to train unique policies. As a result, our goal is to construct a single planning algorithm based on hierarchical reinforcement learning (HRL). This can accomplish behavior planning in an environment where the agent can pursue multiple sub-goals, and do so in such a way that sub-goal policies can be reused for subsequent tasks in a modular fashion.

In Figure 3.1, according to the three main problems mentioned in Chapter 1, we depict the corresponding proposed methods and what specific issues each of them addresses. In this chapter, we will introduce the heuristic-based decision making system in Section 3.2, followed by fundamental reinforcement learning algorithms in Section 3.3. In Sections 3.4 through 3.7, we

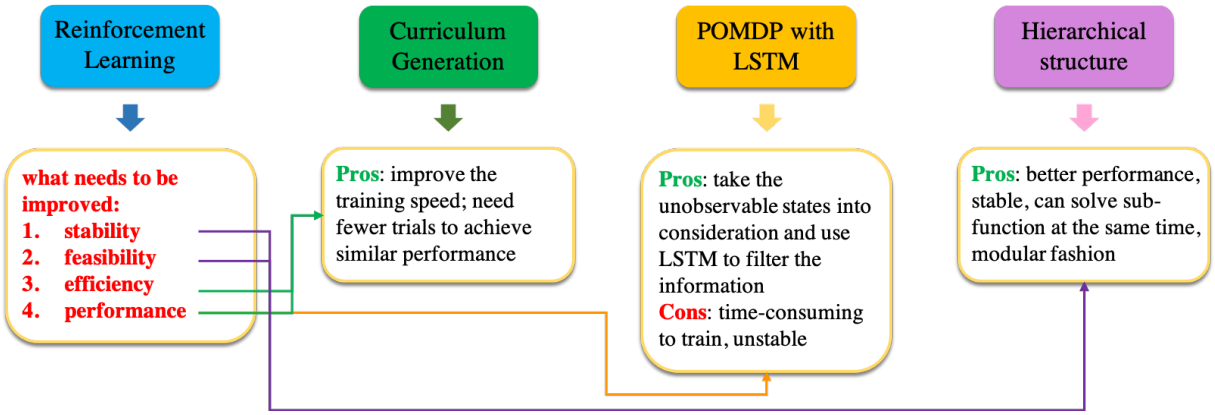

Figure 3.1: Comparison between different RL methods

introduce our proposed approaches for extending the RL structure: Automatically Generated Curriculum, RL with LSTM, and Hierchical RL, and RL with Prior Knowledge.

3.2 Heuristic-based Decision Making System

For the heuristic-based decision making system of an autonomous vehicle, a straightforward method is to enumerate various scenarios that the autonomous vehicle may encounter. For example, Figure 3.2 shows a decision-making structure for an autonomous vehicle under various urban scenarios which are shown in corresponding colors. The system tries to cover as many situations as possible for urban scenarios, but some parameter values still need to be tuned based on the agent's parameters, map information and surrounding human-driven vehicles' behavior. In this example, the decision-making structure is divided into four layers. As one proceeds from higher to lower layers, the decisions become more detailed. Especially for the Action layer, accurate parameter values need to be set according to different situations. For example, in a scenario of following a lead vehicle and approaching an intersection, the exact distance to keep away from the lead vehicle is dependent on the lane, traffic situation and ego agent's intention.

The rule-based method has the problem of an unconstrained number of rules being needed in order to cover as many scenarios as possible. However, expanding such a decision-making dia-

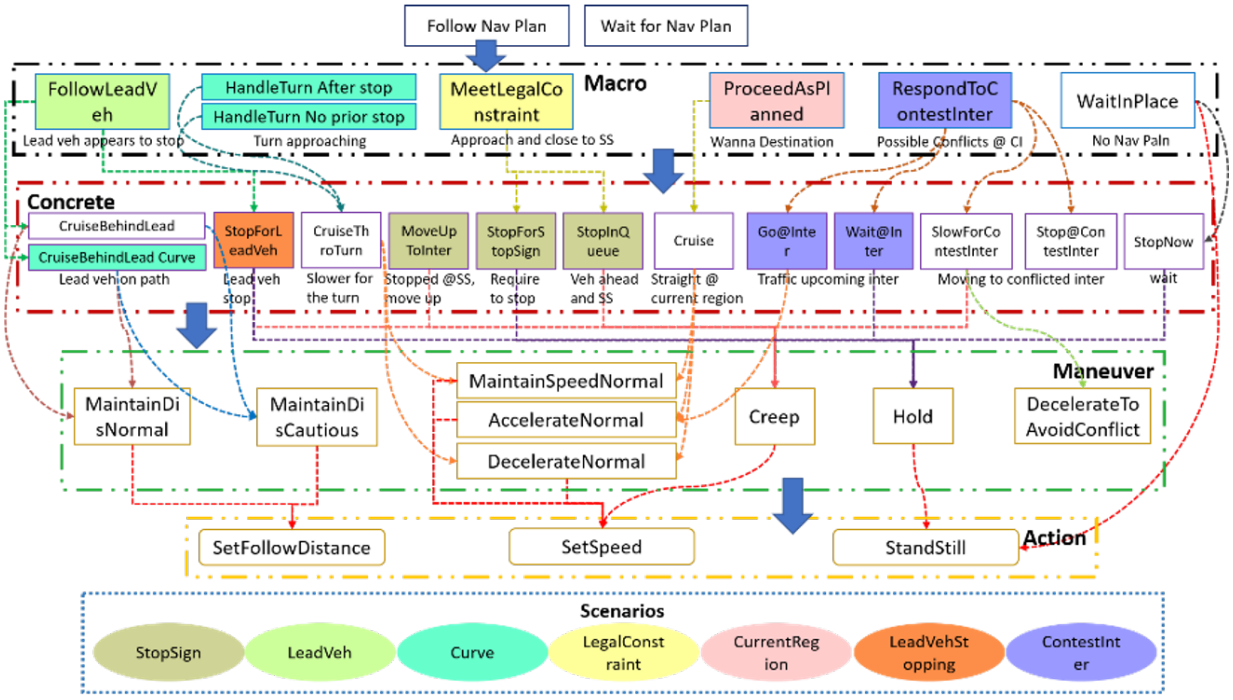

Figure 3.2: Structure of heuristic-based decision making system

gram is labor-intensive. We introduce an AI-based method to avoid the heuristic-based method's rule enumeration process, but meanwhile, the skeleton of the decision-making structure is retained and the machine learning methods are constructed based on such a skeleton so that the new AI-based methods can be stable as well as intelligent.

3.3 Current Machine Learning Algorithms

Current machine learning algorithms can be categorized into three main types :

1. Supervised learning

   Supervised learning is mainly applied to figure out a mapping function $f(X)$ between a set of data X and its label or the corresponding value of features for the data Y.

   $$Y = f(x) \tag{3.1}$$

   By learning the collected data with its label or value of features, the algorithm can auto-

matically predict the label or value of features for the new data which are unseen in the dataset. The two major functions of supervised learning are classification and regression.

2. Unsupervised learning

   Unlike supervised learning, where all collected data have been assigned labels or corresponding values of features, unsupervised learning allows the automated search of a set of data for patterns and information that haven't been detected or identified before. Clustering is a typical application of unsupervised learning. Clustering algorithms can process a set of data and find clusters or groups if they exist in the data.

3. Reinforcement learning Reinforcement learning (RL) is the third category of machine learning methods. It can learn a policy π specifying how an intelligent agent ought to take actions a in an environment in order to maximize cumulative reward r based on current state s. Figure 3.3 shows a block diagram of RL: the algorithm updates the policy $\pi(s_t|a_t)$ while the agent is running in the environment.

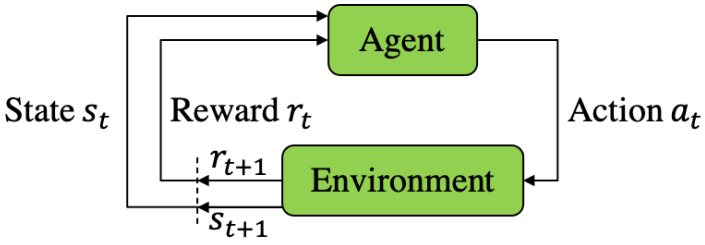

Figure 3.3: Reinforcement learning algorithm

In previous work, mainly supervised learning and reinforcement learning have been applied to improve the decision-making system of self-driving cars. But most supervised learning is taught on limited scenarios and therefore fails when the environment changes. With the help of simulation, RL can create a more robust policy through learning under multiple randomly generated simulated scenarios. In our research, we have shown that RL has the capability to improve based on heuristic-based methods for the decision-making system. In Sections 3.3.1 through 3.3.4, we first introduce the fundamental reinforcement learning approaches (Deep Q-Learning,

Double Deep Q-Learning, Deep Deterministic Policy Gradient, and Hierarchical Reinforcement Learning) that are used in our algorithm and then describe how we improve the heuristic-based method with the help of RL. In Sections 3.4 through 3.7, we introduce the three reinforcement-learning-based approaches (Automatically Generated Curriculum, RL with LSTM, Hierarchical RL and RL with Prior Knowledge) developed in our work.

3.3.1 Deep Q-learning

Since they were proposed, Deep Q-Networks and Double Deep Q-Networks have been widely applied in reinforcement learning problems. In Q-learning, an action-value function $Q_\pi(s, a)$ is learned to get the optimal policy π which can maximize the action-value function $Q^*(s, a)$. Hence, a parameterized action-value function $Q(s, a|\theta)$ is used with a discount factor γ, as in Equation 3.2.

$$\begin{aligned} Q^*(s, a) &= \max_\theta Q(s, a|\theta) \\ &= r + \gamma \max_\theta Q(s', a'|\theta) \end{aligned} \tag{3.2}$$

3.3.2 Double Deep Q-learning

For the setting of Deep Q-learning, the network parameter θ is optimized by minimizing the loss function $L(\theta)$, which is defined as the difference between the predicted action-value Q and the target action-value Y^Q. θ can be updated with a learning rate α, as shown in Equation 3.3.

$$\begin{aligned} Y_t^Q &= R_{t+1} + \gamma \max_a Q(S_{t+1}, a|\theta_\mathbf{t}) \\ L(\theta) &= \Big(Y_t^Q - Q(S_t, A_t|\theta_t)\Big)^2 \\ \theta_{t+1} &= \theta_t + \alpha \frac{\partial L(\theta)}{\partial \theta} \end{aligned} \tag{3.3}$$

For the Double Deep Q-learning setting, the target action-value Y^Q is revised according to another target Q-network $Q^{'}$ with parameter θ':

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a|\theta_t)|\theta_t') \tag{3.4}$$

During the training procedure, techniques such as the ϵ-greedy approach [40] and the prioritized experience replay approach [41] can be applied to improve the training performance.

3.3.3 Deep Deterministic Policy Gradient

For DQN and Double DQN, the policy can only generate a discrete action space, which is less realistic than DDQG, which can generate continuous actions through the policy network. This algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives. The critic-network $Q(s, a|\theta^Q)$ is updated by the same method used in Equation 3.3. The actor-network $\mu\left(s_t|\theta^\mu\right)$ which is used to output an optimal action is updated by using the sampled policy gradient in Equation 3.5:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu\left(s_t|\theta^\mu\right)|_{s_i} \tag{3.5}$$

3.3.4 Hierarchical Reinforcement learning

For the HRL model [38] with sequential sub-goals, a meta controller Q^1 generates the sub-goal g for the following steps and a controller Q^2 outputs the actions based on this sub-goal until the next sub-goal is generated by the meta controller.

$$\begin{aligned} Y_t^{Q^1} &= \sum_{t'=t+1}^{t+1+N} R_{t'} + \gamma \max_g Q(S_{t+1+N}, g|\theta_{\mathbf{t}}{}^1) \\ Y_t^{Q^2} &= R_{t+1} + \gamma \max_a Q(S_{t+1}, a|\theta_{\mathbf{t}}^{\mathbf{2}}, g) \end{aligned} \tag{3.6}$$

3.4 Automated Curriculum Generation

As machine learning-based approaches and especially deep reinforcement learning (DRL)-based approaches have become very popular, the idea of applying DRL to autonomous driving scenarios has gained some attention. Recent work on using deep RL for learning to cross intersections

was able to show that it is able to learn successful policies that are comparable to or can even outperform rule-based systems in terms of successfully reaching the goal [42]. Unlike rule-based algorithms, RL can learn to deal with a continuously changing environment by trial and error. Unlike supervised learning, RL does not need a large amount of labeled data to train a data-based model. Rather than learning a mapping from input to label, RL enables an autonomous agent to learn a mapping from environment states to agent actions from its experience, which is similar to how humans learn to drive.

However, most DRL methods still have difficulty with problems with high-dimensional state spaces such as autonomous vehicle decision-making in urban environments. The basic objective in our work is to apply deep reinforcement learning (DRL) methods to train an agent that can autonomously learn how to approach and traverse an urban stop-sign intersection by collecting information on surrounding vehicles and the road. The problem with DRL algorithms such as Deep Q-learning [16] (DQN) and Deep Deterministic Policy Gradient [17] (DDPG) is that they need a long training period to get an acceptable result. In this section, we introduce an algorithm called Automatically Generated Curriculum (AGC) that can generate a curriculum sequence to accelerate the training process for high-dimensional reinforcement learning problems.

AGC-based reinforcement learning is a curriculum reinforcement learning method which involves two levels of learning. The higher level is responsible for automatically generating a curriculum according to total rewards of test samples after the current training iteration with respect to each task. The lower level applies a traditional DRL algorithm such as DQN or DDPG to train a policy. The actions considered can be either discrete or continuous.

Fig. 3.4 shows a flowchart for ACG-based DQN. The inner rectangle with a red dashed outline is the DRL (DQN or DDPG) process and the outer part is the process for automatically generating the curriculum used for training the DRL algorithm. In the outer part, the policy of the curriculum generation is formulated as a k-armed bandit problem [43], where k is chosen based on the number of tasks. In this work we use an action-value-based incremental method for

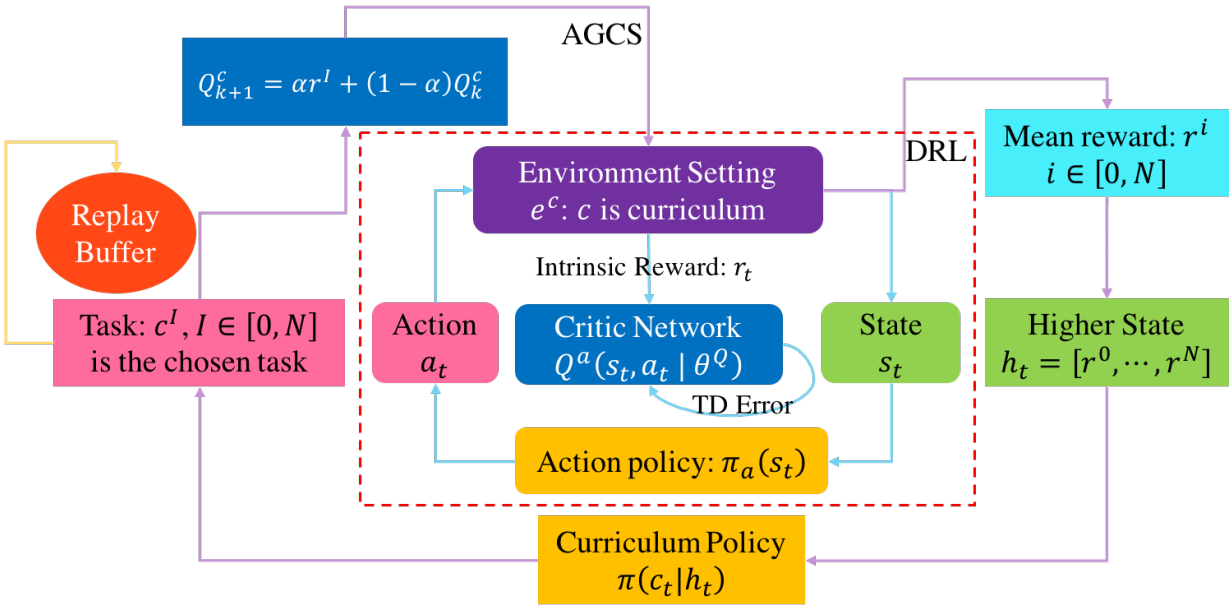

Figure 3.4: Flowchart of Deep Reinforcement Learning with Automatically Generated Curriculum Sequence

the ACG-based RL algorithm. We update the V function of the curriculum generation through Equation 3.7 according to [43], in which $n \in [1, N]$ is the task id and k specifies the training iteration.

$$V_n^{k+1} = \alpha r^n + (1-\alpha)V_n^k \tag{3.7}$$

We use the value function V_i to evaluate the difficulty of each task i. Higher values indicate that this kind of task is relatively easier to get a better performance on than other tasks. AGC-based RL chooses the task to be trained in the next iteration via the Boltzmann distribution exploration method: $\pi(c_n|V_n) = \frac{\exp(V_n)}{\sum_n^N \exp(V_n)}$. An easier task with a higher V_n will result in a higher probability to be chosen as the curriculum for the next training iteration.

DQN [16] and DDPG [17] usually store the last M experiences as tuples of $s_t, a_t, r_t, s_{t+1}, a_{t+1}$ in a replay buffer $\mathbf{B}$. The original algorithm uses uniform sampling, which gives equal importance to all the transitions in the replay memory and may result in an unbalanced memory with more failure cases than success epochs. Here, the experiences during a training iteration are

added to the Replay Buffer $\mathbf{B}$ such that it represents the probability of the current chosen task I to be chosen in the next training iteration, which is $\pi(c_I|V_I^{k+1})$. This means that after training on one task in one training iteration, if the training result improves the policy and gets better performance, the experiences during the training iteration have a higher probability of being added to the replay buffer. Meanwhile, if the transition is not chosen, it will be abandoned without storing it in the buffer. As a result, the buffer only contains transitions which may include more buffers that can help to train out a better result. The complete algorithm is shown in Algorithm 1.

3.5 Reinforcement Learning with LSTM

When applying autonomous driving technology to some real-world scenarios, environmental uncertainties make the development of decision-making algorithms difficult. Modeling the problem as a Partially Observable Markov Decision Process (POMDP) [23] allows the algorithm to consider these uncertainties in the decision process, which makes it more robust to real sensor characteristics. As a result, in this section, we propose to model the decision making problem as a POMDP and will use deep reinforcement learning (DRL) to optimize the policy for generating continuous actions.

Figure 3.5 shows an example of when the POMDP model can be applied. The intersection is four-way with two-way stop-sign traffic control. When the ego car can begin to approach the intersection, the pink ray-trace shows the field-of-view of the ego car and when the car is far away from the intersection, the visibility range is limited to the current lane but has no information about the road to be crossed. With the help of the POMDP model, the ego car can predict information through historical data.

For the partially observed MDP problem, we propose a network based on the ideas of DDPG [17] and ADRQN [27] to generate continuous actions based on the observations coming from previous steps. In the network, instead of using all the previous observations, only a fixed number of previous steps $n_{steps} = 20$ are used as the inputs to the LSTM layer. Figure 3.6 shows the

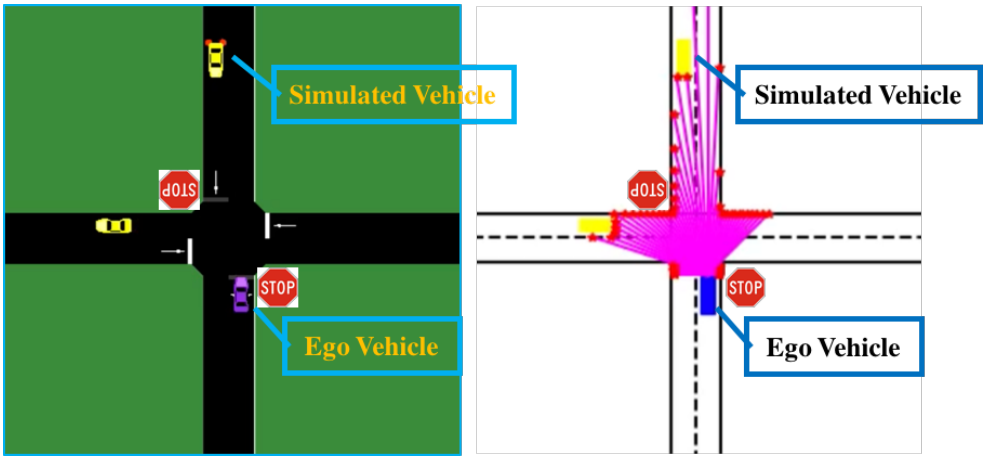

Figure 3.5: The proposed intersection traversal scenario with simulated ray traces. The ego vehicle starts from the stop line with zero velocity and other simulated vehicles run in other lanes using the Krauss Traffic Model.

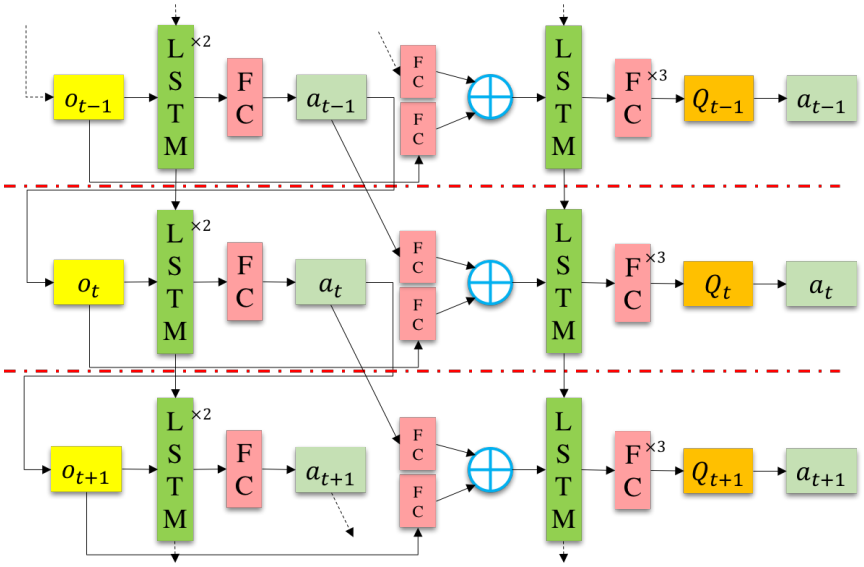

Figure 3.6: POMDP with LSTM network for generating continuous actions.

structure of the proposed network. The input observation o_t is a 126-D vector which generates a 3-D continuous action vector a_t through two 512-D LSTM layers and one fully connected (FC) layer. Meanwhile, the observation vector followed by an FC layer concatenates with the action vector coming from the previous step followed by an FC, and then followed by a 128-D LSTM

layer and three FC layers to produce a 1-D Q-values Q_t corresponding to the continuous action vector.

3.6 Hierarchical Reinforcement Learning

Solving the POMDP with reinforcement learning (RL) [43] often requires storing a large number of observations. For continuous action spaces, the POMDP system will be computationally inefficient. As a result, we address the decision making problem by modeling it as an MDP and learning a policy with RL using a hierarchical structure. Our goal is to construct a single planning algorithm based on hierarchical reinforcement learning (HRL) which can accomplish behavior planning in an environment where the agent pursues multiple sub-goals and to do so in such a way that sub-goal policies can be reused for subsequent tasks in a modular fashion Fig. 3.7 shows the transition between rules-enumeration and reinforcement learning based methods.

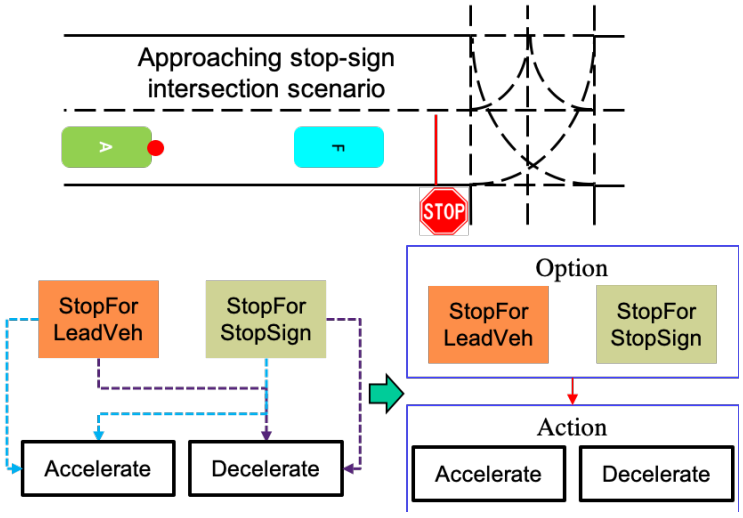

Figure 3.7: Heuristic-based structure vs. HRL-based structure

3.6.1 RL with Hierarchical Options

For MDP with hierarchical options, we use three networks to produce hierarchical options, low-level actions, and Q-values. Unlike the POMDP network, the system does not take observations from previous time-steps into account for the decision process. However, it uses hierarchical options to make a decision on whether the ego vehicle can trust the environment or not. Then, based on the observations and high-level decision, the agent makes the decision on the low-level policies. The structure of the network is shown in Figure 3.8. The 126-D input state vector s_t is followed by three FC layers in order to generate a 2-D Q-values O_t corresponding to two hierarchical option candidates. Meanwhile, the input state vector produces a 2-D continuous action vector a_t through four FC layers. Then, the state vector followed by an FC layer concatenates with the action vector followed by one FC layer. The output produces a 1-D Q-values Q_t which corresponds to the action vector through four FC layers.

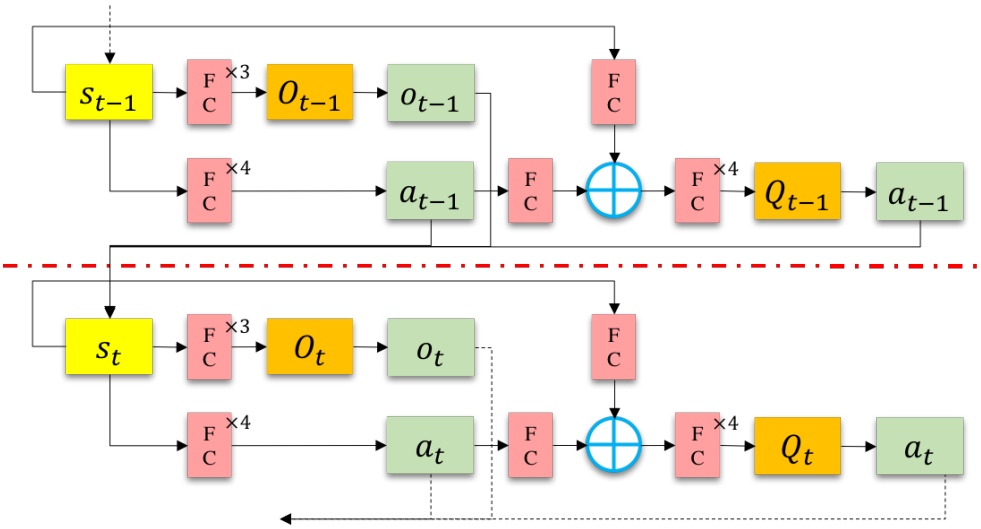

Figure 3.8: MDP with Hierarchical Options network for generating continuous actions.

The lower-level decision here means the acceleration or deceleration the agent takes at the current step. The overall flow of the algorithm is given in Algorithm 2.

3.6.2 Hybrid HRL

In this section we present our proposed model, which is a hierarchical RL network with an explicit attention model, hybrid reward mechanism and a hierarchical prioritized experience replay training schema. We will refer to this model as Hybrid HRL.

Hierarchical RL with Attention

Hierarchical structures based on RL can be applied to learn a task with multiple sub-goals. For a hierarchical structure with two levels, an option set O is assigned to the first level, whose object is to select among sub-goals. The weight θ_t^o is updated according to Equation 3.8.

$$\begin{aligned} O_{t+1}^* &= \arg\max_o Q^o(S_{t+1}, o|\theta_t^o) \\ Y_t^{Q^o} &= R_{t+1}^o + \gamma Q^o(S_{t+1}, O_{t+1}^*|\theta_t^{o'}) \\ L(\theta^o) &= \left(Y_t^{Q^o} - Q^o(S_t, O_t|\theta_t^o)\right)^2 \end{aligned} \tag{3.8}$$

After selecting an option o, the corresponding action set A^o represents the action candidates that can be executed on the second level of the hierarchical structure with respect to the selected option o. Some previous work proposed the Hierarchical Markov Decision Process (MDP) [44], which shares the state set S among different hierarchical levels during the MDP or designs different states for changing sub-goals and applies initial and terminating condition sets to transfer from one state set to another.

In many situations, the portion of the state set and the amount of abstraction needed to choose actions at different levels of this hierarchy can vary widely. In order to avoid designing a myriad of state representations corresponding to each hierarchy level and sub-goal, we share one state set S for the whole hierarchical structure. Meanwhile, an attention model is applied to define the importance of each state element $I(s, o)$ with respect to each hierarchical level and sub-goal and then use these weights to reconstruct the state s^I. The weight θ_t^a is updated according to

Equation 3.9.

$$
\begin{aligned}
A^*_{t+1} &= \arg\max_a Q^a(S^I_{t+1}, O^*_{t+1}, a|\theta^a_t) \\
Y^{Q^a}_t &= R^a_{t+1} + \gamma Q^a(S^I_{t+1}, O^*_{t+1}, A^*_{t+1}|\theta^{o'}_t) \\
L(\theta^a) &= \left(Y^{Q^a}_t - Q^a(S^I_t, O_t, A_t|\theta^a_t)\right)^2
\end{aligned}
\tag{3.9}
$$

When implementing the attention-based HRL, we construct the option network and the action network (Figure 3.9), which includes the attention mechanism as a $softmax$ layer in the action-value network Q^a.

Hybrid Reward Mechanism

For a sequential sub-goals HRL model [38], the reward function is designed separately for the sub-goals and main task. The extrinsic meta reward is responsible for the option-level task, and meanwhile the intrinsic reward is responsible for the action-level sub-goals. For HRL with parameterized actions [36], an integrated reward is designed to evaluate both option-level and action-level together.

In our work, instead of generating one reward function which is applied to evaluate the final outputs coming from both options and actions in one step together, we designed a reward mechanism which can evaluate the goodness of option and action separately during the learning procedure. As a result, a hybrid reward mechanism is introduced so that: 1) the algorithm gets the information of which reward function should be triggered to get rewards or penalties; 2) meanwhile, a positive reward which benefits both option reward and action reward occurs if and only if the whole task and the sub-goals in the hierarchical structure have all been completed. Figure 3.10 demonstrates the idea for the hybrid reward mechanism.

Hierarchical Prioritized Experience Replay

In [41] the authors propose a framework for more efficiently replaying experience during the training process in DQN so that the stored transitions $\{s, a, r, s'\}$ with higher TD-error in the

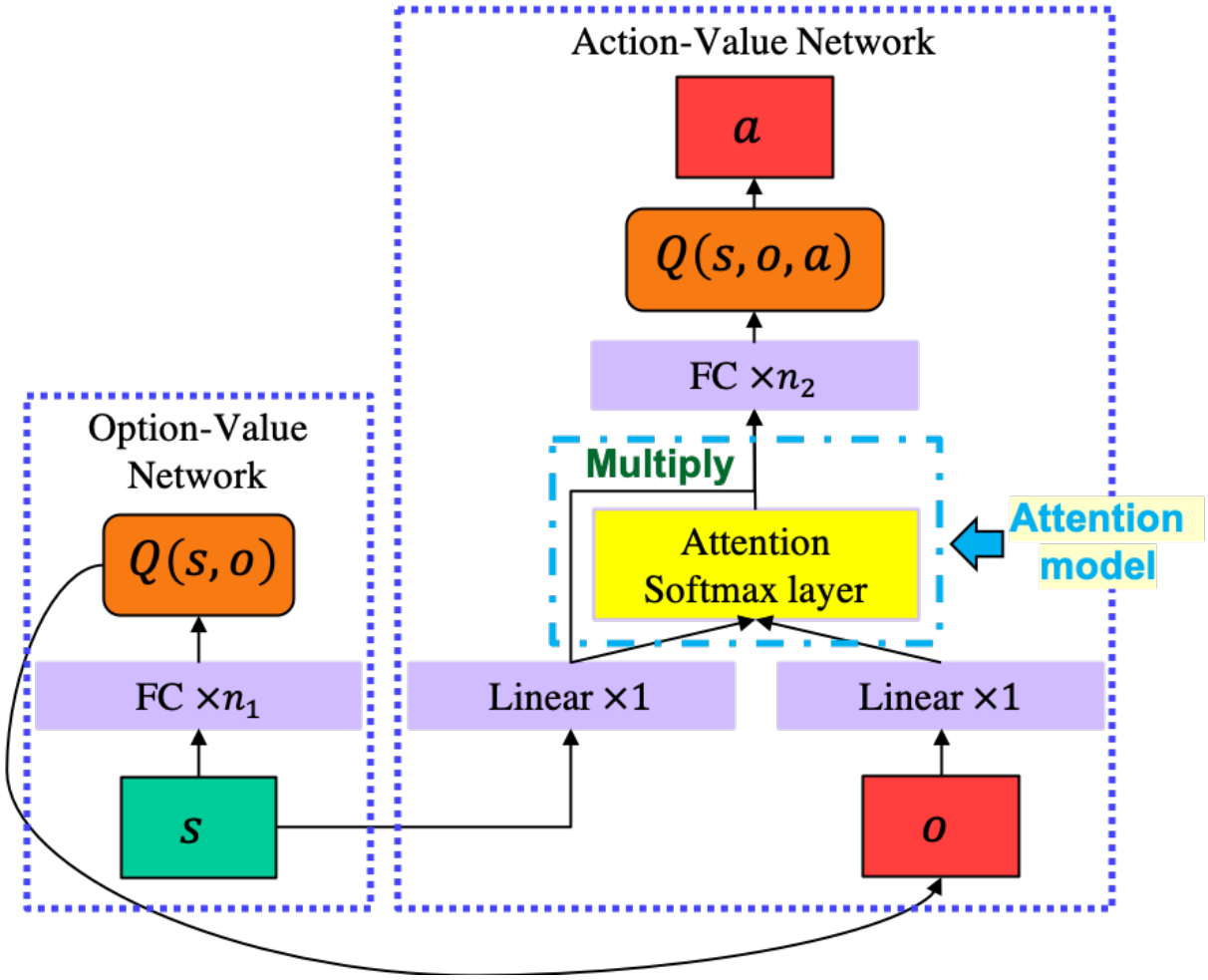

Figure 3.9: Hierarchical RL Option and Action Q-Network. FC stands for a fully connected layer. Within all the FC layers, *Linear* activation functions are used to generate last layers in both Option-Value and Action-Value networks. For the rest of the layers, *ReLu* activation functions are applied.

previous training iteration result in a higher probability of being selected in the mini-batch for training during the current iteration. However, in the HRL structure, the rewards received from the whole system not only rely on the current level, but also are affected by the interactions among different hierarchical levels.

For the transitions $\{s, o, a, r^o, r^a, s'\}$ stored during the HRL process, the central observation is that if the output of the option-value network o is chosen wrongly due to high error between

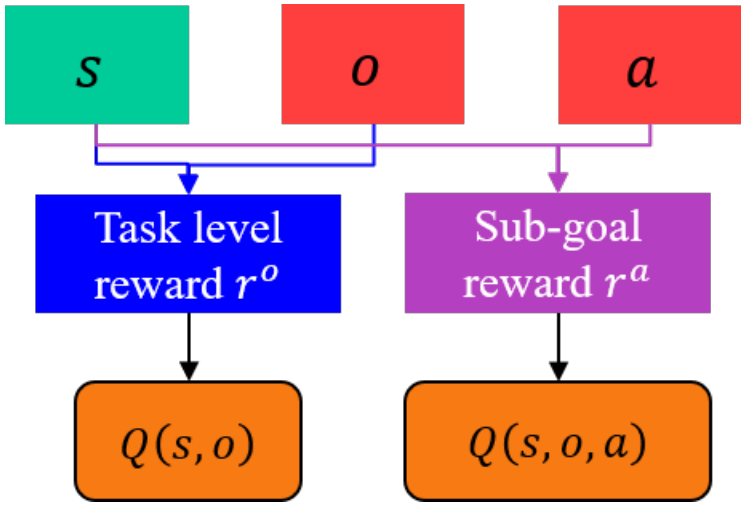

Figure 3.10: Hybrid Reward Mechanism

predicted option-value Q^o and the targeted option-value $r^o + \gamma Q^o(s', o')$, then the success or failure of the corresponding action-value network is inconsequential to the current transition. As a result, we propose a hierarchical prioritized experience replay (HPER) in which the priorities in the option-level are based on error directly and the priorities in the lower level are based on the difference between errors coming from two levels. Higher priority is assigned to the action-level experience replay if the corresponding option-level has lower priority. According to Equations 3.8 and 3.9, the transition priorities for option and action level are given in Equation 3.10.

$$\begin{aligned} p^o &= \left| Y^{Q^o} - Q^o(S, O|\theta^o) \right| \\ p^a &= \left| Y_t^{Q^a} - Q^a(S_t^I, O_t, A_t|\theta_t^a) \right| - p^o \end{aligned} \tag{3.10}$$

Based on the above-described approaches, the Hybrid HRL is shown in Algorithms 4, 5 and 6.

3.7 Reinforcement Learning with Prior Knowledge

In Chapter 1, we mentioned that prior knowledge is important for human drivers to drive effectively. How to incorporate prior knowledge or previously designed rules into the learning procedure is a subject that we introduce in the current section.

3.7.1 Exploration with heuristic

In reinforcement learning, the learning procedure is actually a process to transfer the prior knowledge into policies. Some recent work [45][46] proposes to use demonstration knowledge in the reinforcement learning procedures so that the algorithms can achieve more efficient learning. However, for the autonomous vehicle driving behavior decision-making problem, some heuristic-based methods can provide imperfect but successful solutions to finish assigned tasks. As a result, inspired by the ϵ-greedy approach, we introduce the ϵ-HeuristicExplore (Eq.3.11) approach, which can explore the heuristic-based policies with higher probability during the early training stage and meanwhile embed some random exploration and exploitation with lower probability.

$$a = \begin{cases} \text{rule-based action} & \text{with probability } \epsilon/2 \\ \text{random action} & \text{with probability } \epsilon/2 \\ a* & \text{with probability } 1-\epsilon \end{cases} \tag{3.11}$$

$a*$ is the action received from exploitation. In this kind of method, the agent can get a higher average reward at the beginning of training and less effective policies are stored in the training replay buffer. As a result, the agent can access both high-quality (heuristic-based) and random explorations during the whole training procedure. Meanwhile, the probability of exploring heuristic-based policies decreases during the training procedure. Instead of exploring an unknown environment with totally random actions, the agent gets an idea of what action may bring higher reward based on a rule-based algorithm which helps the agent to learn more quickly.

3.7.2 Adjusted heuristic exploration

Based on the heuristic-based exploration for ϵ-greedy method, we further propose to adjust the decay rate for ϵ according to changing total reward of the current epoch. When the average total reward is higher than a period of previous epochs, ϵ is decreased during the training process.

Otherwise, ϵ will increase to favor exploration over exploitation.

Algorithm 7 describes the main training approach for a three-layer autonomous vehicle planning system. Algorithm 8 describes the method flow of adjusted heuristic exploration during the training process.

3.8 Summary

To sum up, DRL is promising for autonomous vehicle behavior planning problems in which rule-based algorithms may have difficulties. However, DRL always needs a long training period for a good result, and may result in unstable training performance or not be capable of solving various scenarios with a single policy. As a result, based on traditional reinforcement learning algorithms, we proposed three approaches which can improve the performance of RL. ACG-based DRL (Section 3.4) can significantly reduce training time compared to plain DRL. An LSTM-module (Section 3.5) embedded in the network during training can deal with the POMDP problem by generating continuous actions. The hierarchical structure (Section 3.6) can increase the possibility of validation for learning-based methods with modular sub-functions which take advantage of a hybrid reward mechanism and heuristic-based exploration.

Algorithm 1 Automatically Generated Curriculum for DQN

procedure AGC-RL

  Construct an empty replay buffer $\mathbf{B}$

  for $n \leftarrow 1$ to N candidate tasks **do**

    Randomly initialize critic network $NN^n_{Q^a}$ with weights θ^n_Q and the target critic network $NN^n_{Q^{a'}}$ with weights $\theta^n_{Q'}$.

    for $e \leftarrow 1$ to E epochs **do**

      r^n_e, $NN^n_{Q^a}$, $NN^n_{Q^{a'}}$, TB_n = Train($task^n$, $NN^n_{Q^a}$, $NN^n_{Q^{a'}}$, $\mathbf{B}$) and add TB_n into $\mathbf{B}$

    $V^0_n = \frac{1}{E}\sum_e r^n_e$

  for $k \leftarrow 0$ to K training iterations **do**

    $P(n) = \pi(c_n|V^k_n) = \frac{\exp(V^k_n)}{\sum^N_n \exp(V^k_n)}$

    I = sample($[1, \cdots, N]$, prob=$[P(1), \cdots, P(N)]$)

    if $k \geq 1$ **and** sample($[0, 1]$, prob=$[P(n), 1 - P(n)]$) is 1 **then**:

      Add TB_n into $\mathbf{B}$ according to $P(n)$

    for $e \leftarrow 1$ to E epochs **do**

      r^I_e, NN^n_Q, TB_n = Train($task^I$, $NN^I_{Q^a}$, $\mathbf{B}$)

    for $n \leftarrow 1$ to N candidate tasks **do**

      $V^{k+1}_n = \alpha\frac{1}{E}\sum_e r^n_e + (1-\alpha)V^k_n$

      for $e \leftarrow 1$ to E epochs **do**

        Get initial states s_0 of $task^n$

        for $t \leftarrow 0$ to T **do**

          Select $a_t = \arg\max_{a_t} Q^a(s_t, a_t)$ and execute a_t

          $s_{t+1} = T(a_t, s_t)$, $r^n_e = r_{ego} + r_{target} - \sigma_4 \mathbb{I}_{\|p^i_{target} - p_{ego}\|^2 = 0} + \sigma_5 \mathbb{I}_{\|p_{des} - p_{ego}\|^2 = 0}$

procedure TRAIN($task^n$, $NN^n_{Q^a}$, $NN^n_{Q^{a'}}$, $\mathbf{B}$)

  Empty TB_n and get initial states s_0 of $task^n$, $r^n = 0$

  for $t \leftarrow 0$ to T **do**

    Select $a_t = \pi(s_t)$ according to ϵ exploration and execute a_t to get new state $s_{t+1} = T(s_t, a_t)$.

    Get reward r_t and $r^n += r_t$

    Add (s_t, a_t, r_t, s_{t+1}) to the temporary Replay Buffer TB_n and sample random mini-batch of M transitions (s_i, a_i, r_i, s_{i+1}) from the replay buffer $\mathbf{B}$.

    Minimize critic loss: $L = \sum_i (y_i - Q^a(s_i, a_i|\theta^Q))^2$ where $y_i = r_i + \gamma Q^{a'}(s_{i+1}, a_{i+1}|\theta^{Q'})$.

    Update weights: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$

Algorithm 2 MDP with Hierarchical Options via RL

procedure HOMDP

Construct two empty replay buffers $\mathbf{B}_a$ and $\mathbf{B}_o$.

Randomly initialize actor network NN^a, critic network NN^Q and option network NN^O with weights θ^μ, θ^Q and θ^O and the corresponding target actor network $NN^{\mu'}$, critic network $NN^{Q'}$ and option network $NN^{O'}$ with weights $\theta^{\mu'}$, $\theta^{Q'}$ and $\theta^{O'}$.

for $e \leftarrow 1$ to E epochs **do**

Get initial state s_0. Initial option is $o_0 = SlowForward$. $r_o = 0$.

for $t \leftarrow 1$ to T time steps **do**

o_t, a_t = GetAction(s_t, o_{t-1})

$s_{t+1}, r_t, done =$ StepForward(s_t, o_t, a_t)

if o_t is $Forward$ **then**

$r_o += r_t$ and add $(s_t, o_t, r_t, s_{t+1}, done)$ to $\mathbf{B}_o$.

if $done$ **then**

Add $(s_t, o_t, r_o, s_{t+1}, done)$ to $\mathbf{B}_o$.

else

Add $(s_t, o_t, r_t, s_{t+1}, done)$ to $\mathbf{B}_o$.

Sample random mini-batch of M transitions (s_i, o_i, r_i, s_{i+1}) from $\mathbf{B}_o$ and (s_j, a_j, r_j, s_{j+1}) from $\mathbf{B}_a$.

$o_{i+1} = \arg\max_o O'(s_{i+1}|\theta^{O'})$. $y_i^o = r_i + \gamma O'(s_{i+1}|\theta^{O'})$.

Minimize $L^o = \frac{1}{M}\sum_i y_i^o - O(s_i|\theta^O)$ to update NN^O.

$y_j^\mu = r_j + \gamma Q'(s_{j+1}, a_{j+1}|\theta^{Q'})$

Minimize $L^\mu = \frac{1}{M}\sum_j y_j^\mu - Q(s_i|\theta^Q)$ to update NN^Q.

$\frac{1}{M}\sum_j \nabla_{\mu(s_j)} Q(s_j, \mu(s_j)|\theta^Q) \nabla_{\theta^\mu} \mu(s_j|\theta^\mu)$ is the policy gradient and is used to update actor network.

Update the target networks: $\theta^{z'} \leftarrow \tau\theta^z + (1-\tau)\theta^{z'}$ for z in $\{\mu, Q, O\}$.

Algorithm 3 Get Action

procedure GETACTION(s, o)
    if o is SlowForward **then**
        $o \leftarrow \arg\max_o O'(s|\theta^{O'})$ according to ϵ greedy.
        $a = 0$.
    if o is Forward **then**
        $a = \mu(s|\theta^{\mu}) + \mathcal{N}$ where $\mathcal{N}$ is a random process.

Algorithm 4 Hierarchical RL with Attention State

procedure HRL-AHR()
    Initialize option and action network Q^o, Q^a with weights θ^o, θ^a and the target option and action network $Q^{o'}$, $Q^{a'}$ with weights $\theta^{o'}$, $\theta^{a'}$.
    Construct an empty replay buffer $\mathbf{B}$ with max memory length l_B.
    for $e \leftarrow 0$ to E training epochs **do**
        Get initial states s_0.
        while s is not the terminal state **do**
            Select option $O_t = \arg\max_o Q^o(S_t, o)$ based on ϵ-greedy. O_t is the selected sub-goal that the lower-level action will execute.
            Apply attention model to state S_t based on the selected option O_t: $S_t^I = I(S_t, O_t)$.
            Select action $A_t = \arg\max_a Q^a(S_t^I, O_t, a)$ based on ϵ-greedy.
            Execute A_t in simulation to get S_{t+1}.
            $R^o_{t+1}, R^a_{t+1} = HybridReward(S_t, O_t, A_t)$.
            Store transition T into $\mathbf{B}$: $T = \left\{S_t, O_t, A_t, R^o_{t+1}, R^a_{t+1}, S_{t+1}\right\}$.
        Train the buffer $ReplayBuffer(e)$.
        if e mod $n == 0$ **then**
            Test without action exploration with the weights from training results for n epochs and save the average rewards.

Algorithm 5 Hybrid Reward Mechanism

procedure HYBRIDREWARD()
  Penalize R_t^o and R_t^a for regular step penalties (e.x.: time penalty).
  for δ in sub-goals candidates **do**
    if δ fails **then**
      if option $o_t == \delta$ **then**
        Penalize option reward R_t^o
      else
        Penalize action reward R_t^a
  if task success (all δ success) **then**
    Reward both R_t^o and R_t^a.

Algorithm 6 Hierarchical Prioritized Experience Replay

procedure REPLAYBUFFER(e)
  mini-batch size k, training size N, exponents α and β.
  Sample k transitions for option and action mini-batch:

$$MB^g \sim P^g = \frac{p^{g\alpha}}{\sum_0^{l_B} p_i^{g\alpha}}, \quad g \in \{o, a\}$$

  Compute importance-sampling weights:

$$w^g = \frac{[N \cdot P^g]^{-\beta}}{\max_i w_i^g}, \quad g \in \{o, a\}$$

  Update transition priorities:

$$p^o = \left| Y_t^{Q^o} - Q^o(S_t, O_t|\theta_t^o) \right|$$

$$p^a = \left| Y_t^{Q^a} - Q^a(S_t^I, O_t, A_t|\theta_t^a) \right| - p^o$$

  Adjust the transition priorities to be greater than 0: $p^a = p^a - min(p^a)$.
  Perform gradient descent to update $\theta_t^g = \theta_t^g + \alpha \frac{\partial L(\theta^{\mathbf{g}})}{\partial \theta^{\mathbf{g}}}$ according to sample weights w^g, $g \in \{o, a\}$.
  Update target networks weights $\theta^{g'} = \theta^g$, $g \in \{o, a\}$.

Algorithm 7 HybridHRL for three-layer planning system

procedure HYBRIDHRL()

Initialize behavior-layer and trajectory-layer network Q^b, Q^p with weights θ^b, θ^p and the target behavior and trajectory network $Q^{b'}$, $Q^{p'}$ with weights $\theta^{b'}$, $\theta^{p'}$.

Construct an empty replay buffer $\mathbf{B}$ with max memory length l_B.

$\epsilon = 1$, k is a predefined training period number.

for $e \leftarrow 0$ to E training epochs **do**

Get initial states s_0.

while s is not the terminal state **do**

Select behavior decision B_t and P_t based on $AdjustedHeuristicExploration()$.

Apply PID controller to trace the trajectory point P_t in simulation to get corresponding throttle, brake and steering angle and results in the next state S_{t+1}.

$r^b_{t+1}, r^p_{t+1} = HybridReward(S_t, B_t, P_t)$.

Store transition T into $\mathbf{B}$: $T = \left\{S_t, B_t, P_t, r^b_{t+1}, r^p_{t+1}, S_{t+1}\right\}$.

$R_e = \sum_t r_t$

if $\sum_{e-4k}^{e-2k} R_e < \sum_{e-2k}^{e} R_e$ **then**

$\epsilon = \eta\epsilon$, $\eta \in [0, 1]$

else

$\epsilon = \epsilon/\eta$, $\eta \in [0, 1]$

Train the buffer.

Algorithm 8 Adjusted Heuristic Exploration

procedure ADJUSTEDHEURISTICEXPLORATION()

if $(e/k) \bmod 2 = 0$ **then**

if $random() > \epsilon$ **then**

$action = FollowHeuristicRule()$

else

$action = FollowHeuristicRule() + \mathcal{N}(\mu, \sigma)$

else

$action = \arg\max_{action} Q(state, action)$

return action

Chapter 4

Simulation Experiments

4.1 Overview

In this chapter, we describe experiments corresponding to the approaches proposed in Chapter 3. Compared to the highway situation, urban scenarios are much more complicated with more corner cases for self-driving cars. On the highway, both road structures and human vehicle behavior are more well-traced and more predictable. In urban situations, road structure and shape have more variation and owing to the range of different possible situations, human vehicles' behavior is hard to predict, which makes it difficult for rule-based algorithms to generate a tidy set of rules covering all situations. As a result, in the experiments, the proposed algorithms are applied to behavior decision making of self-driving cars at urban intersections. For urban intersections with traffic lights or signs only, heuristic-based rules-enumeration needs a large amount of effort to design rules covering different situations like traffic-jam, unexpected lane-change, stopped vehicles, etc. RL has the advantages of learning automatically and showing better performance in the face of scenarios unseen during testing.

For each proposed algorithm in Chapter 3, we trained an agent in the simulation so that it can learn how to finish different tasks according to designed situations in which heuristic-based methods find it difficult to obtain good performance. Depending on the method, we tested both

on discrete and continuous actions for various scenarios. Among different intersection types in the US, stop-sign intersections need the most interaction between drivers. For example, for a two-way (i.e., with only two directions governed by a stop sign) intersection (shown in Figure 4.1), the ego car is not allowed to interfere with the cars on the east-west road that has the right of the way. As a result, heuristic-based methods may have difficulty, especially when the ego car needs to negotiate with human drivers a lot. For the Automatically Generated Curriculum and RL with LSTM-module experiments, we choose a four-way intersection with two-way stop-signs in which the ego car starts from a lane with a stop-sign without front vehicles. The agent needs to deal with the approaching vehicles from the crossed road in order to transverse the intersection. That is also the scenario in which we initially tested the hierarchical structure. Moreover, we added front vehicles in front of the ego car to increase the difficulty in the test scenarios and also tested a two-lane intersection with traffic lights. In the next chapter, we discuss more scenarios in the real world and apply the hierarchical structure to them.

4.2 Experiments for Automatic Curriculum Generation

We applied the automatic curriculum generation methods to the two scenarios which are presented in Figure 4.1. The scenario setup considered in this work is a four-way intersection with two-way stop signs. In each scenario, the AV (green solid rectangle with the letter "A") has to reach a pre-defined destination (green hollow rectangle with the letter "A"). Vehicles shown as blue rectangles with the letter "T" (front vehicle or approaching vehicles according to different scenarios) are target vehicles within the ego vehicle's visibility range whose information is included in the state space. All the simulated vehicles in the scenario stay in their lanes with constant velocities. Furthermore, all traffic participants except the AV will not change their trajectory in response to the AV. According to human drivers' knowledge, when approaching a two-way stop-sign intersection, if the vehicle coming from the left is very close to the ego car, the ego car should stop without hesitation.

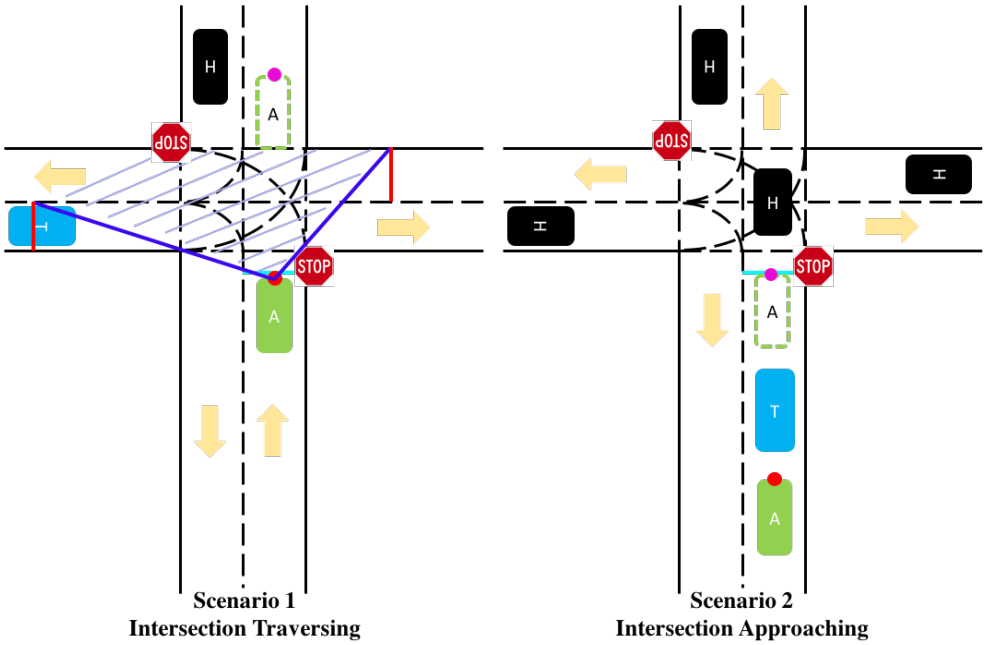

Figure 4.1: Two proposed intersection scenarios. Scenario 1 is for intersection traversing problem and Scenario 2 is for intersection approaching problem. In the plots, the cyan lines are stop lines of the intersection corresponding to the red stop signs.

4.2.1 Experimental Setup

We modeled both scenarios as MDP and trained the Q-network with DQN and DDPG respectively for intersection approaching and traversing scenarios. We trained an agent using a random curriculum and compared the performance and the training time with an agent trained with the Automatically Generated Curriculum approach.

By modeling the problem as an MDP, we solved the Intersection Traversing and Intersection Approaching problems using Deep Q-learning [16] and Deep Deterministic Policy Gradient [17], respectively. All the critic network and actor networks for DQN and DDPG are constituted by two hidden layers with 600 and 300 nodes. The "ReLU" activation function is used for all hidden layers and the "tanh" activation function is used for the actor output in DDPG. We set up a replay buffer with a size of 500,000.

For every training epoch, a successful case means that there is no collision between the ego

vehicle and other vehicles and the whole process can be finished within 2000 steps (200 seconds). We use collision rate and number of steps to finish as metrics to evaluate the tasks.

For the intersection traversing problem, we created a set of six tasks according to different initial positions of the first approaching vehicles and for the intersection approaching problem; four tasks were generated according to the initial positions of the ego vehicle. The initial positions of the approaching simulated vehicles or ego vehicle are randomly generated for initialization at the first epoch and different numbers in Fig. 4.2 correspond to different task IDs.

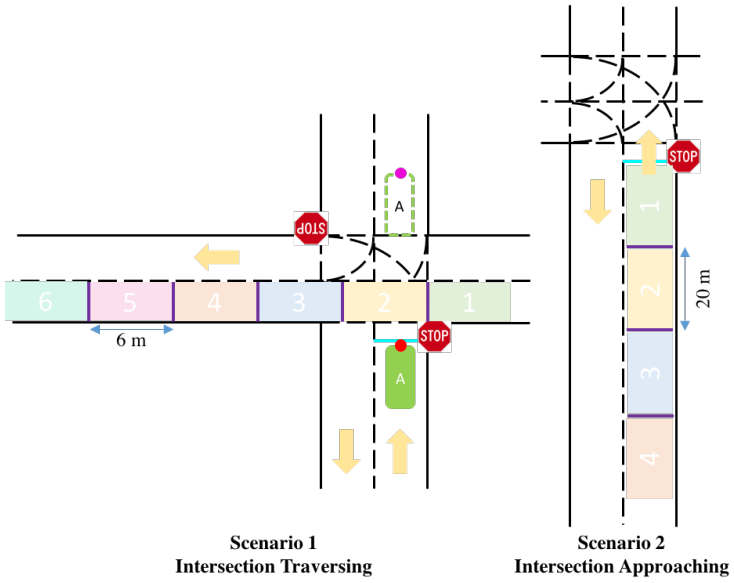

Figure 4.2: Intersection traversing problem is divided into six tasks according to initial position of approaching human vehicles.

4.2.2 Results

In this section, we compare our automatic generated curriculum method with random curriculum (original DQN and DDPG) and manually designed curriculum methods. We tested and compared three methods in scenarios 1 and 2 and provide the results here.

Random Curriculum

We applied the DQN or DDPG algorithm by generating curricula randomly, which means the probability of each task to be chosen obeys a discrete uniform distribution. After training for 250,000 iterations, we can get an 80% success rate for the Traversing scenario and for the Approaching scenario, it takes 7,000 iterations to get to a success rate of 60%.

Manually Designed Curriculum

In order to manually design the curriculum, we firstly train DQN on each task separately. According to the different performance for each task, we manually design the curriculum, beginning with the task which achieves a higher success rate more quickly.

Automatically Generated Curriculum

1. **Intersection traversing problem**

   Applying the AGC-based approach, the probability of each task's being chosen for the next training iteration varies according to the mean rewards the task can get during the current training iteration. Fig. 4.3 shows the probability density function for each task to be chosen for the next training iteration for different tasks according to the AGC-based model for the intersection traversing case. We see that in the first period (roughly before the 100th training iteration), the system prefers Task 2 because training Task 1 can reach an acceptable result easily and Task 2 is relatively simple compared to other tasks. However, in the later training period, the most difficult tasks, Task 4 and 5, are preferred over other tasks because they get the lowest scores when the other scenarios perform well.

   We compare the success rate and mean reward values using random curriculum and AGC-based DQN in Fig. 4.4 and Fig. 4.5. With the help of the AGC structure, the system can reduce training time by a factor of six to reach a similar and more stable performance compared to the vanilla DQN algorithm. In Fig. 4.5, we compared AGC with Random

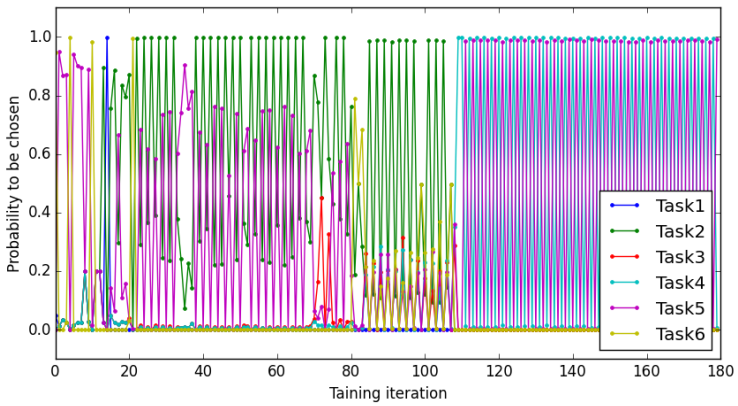

Figure 4.3: Probability of being chosen for next iteration of training for the intersection traversing case

Curriculum and Manually Designed Curriculum (MDC). Especially for MDC, the system performs well for some easy-to-handle tasks; however, when the difficulty increases and the curriculum is not designed well, it may not achieve the expected result. Random Curriculum can achieve an acceptable result after a long time of training.

Fig. 4.6 shows the critic loss of lower-level DQN. It shows that if DQN is trained directly via random curriculum, the system is not stable, and as training time increases, the system may have a divergent critic loss. However, the use of AGC helps the critic loss to gradually decrease and although the chosen task keeps switching, the critic loss is bounded all the time and tends to decrease with the increase of training time.

2. **Intersection approaching problem**

   For the intersection approaching problem, we created a set of four tasks. Each task corresponds to a different range of initial positions for the ego vehicle. The initial distance between the front vehicle and the AV is randomly generated and is greater than 10 meters. The front vehicle always stops at the stop sign first and then traverses the intersection. The initial velocity of the ego vehicle is randomly generated and is between 8 m/s and 12 m/s and the destination of the ego vehicle is to stop completely at the stop line. Fig. 4.7 compares the results between Random Curricula, Manually Designed Curriculum and

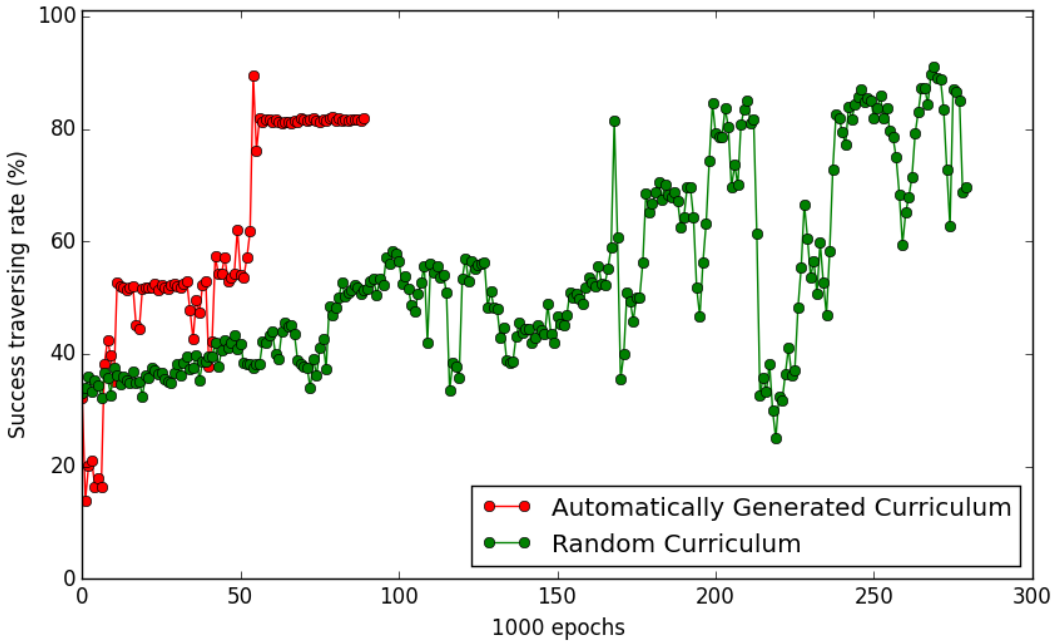

Figure 4.4: Success rate of Random Curricula and AGC-based Model for the intersection traversing case

Table 4.1: Results comparison for different methods with the same number of training iterations

		Steps	Collision	Unfinished	Not stop	Success	Mean Reward
Intersection Traverse	TTC	365	2.2%	10.5%	N/A	87.3%	22.1
	Random Curricula	294	25.6%	16.2%	N/A	58.2%	67.3
	AGC-based Curricula	206	13.5%	4.4%	N/A	82.1%	**132.1**
Intersection Approaching	Random Curricula	183	22.5%	20.7%	15.6 %	41.2%	150.23
	AGC-based Curricula	130	0.21%	0.10%	1.0%	98.69%	**480.39**

the AGC-based model. The AGC model takes fewer iterations to achieve a 99.2% success rate, whereas the Random Curricula method has a success rate less than 80% after twice as much training time and the manually designed curriculum is quite unstable when we transfer from one task to another.

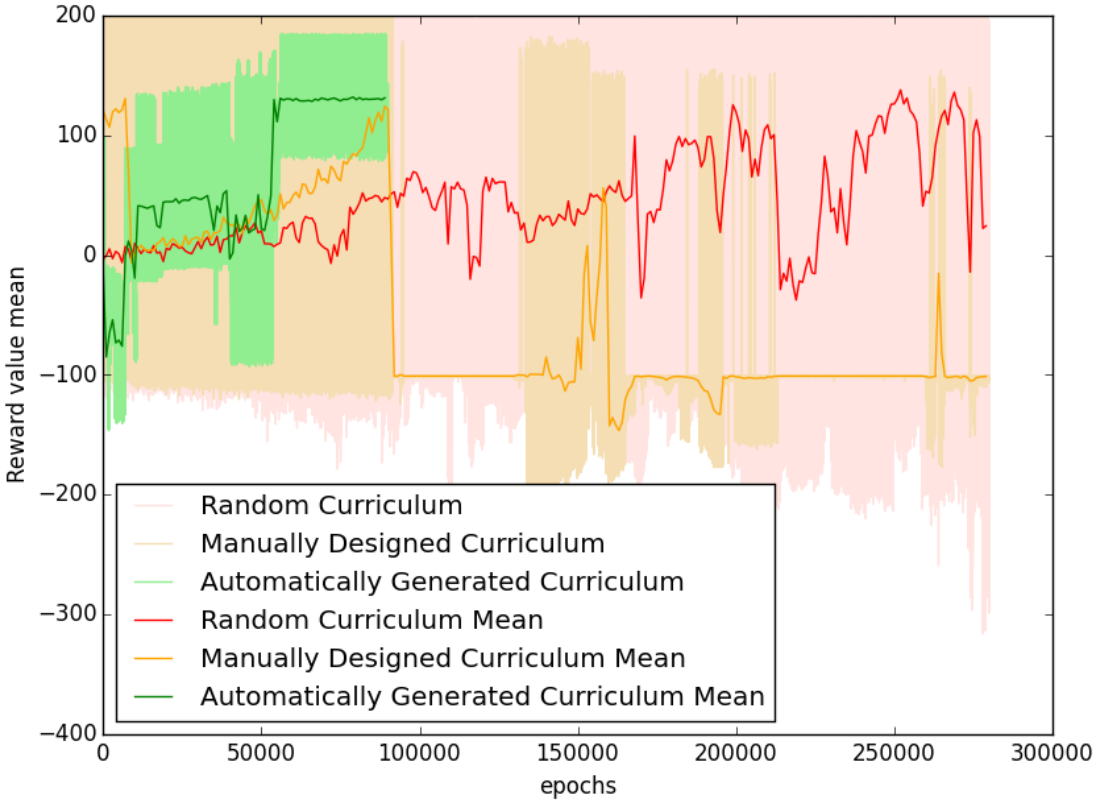

Figure 4.5: Reward value of three methods for the intersection traversing case

4.2.3 Discussion

For testing the performance of the different methods, we train each model with the same number of iterations and then use the result to test for 1,000 episodes with the task chosen at random in each episode. The results of the test with different methods are shown in Table 4.1. Steps in the table means the average number of time steps needed to finish the whole task. For the intersection traverse scenario, we train the model for 100,000 iterations. TTC can have a higher success rate than the RL-based methods, but from the steps metrics we can see that TTC uses a much more conservative strategy than the AGC-based RL does. As a result, the number of time-steps needed by the TTC-based method is higher than necessary to traverse the intersection. This resulted in several unfinished episodes where the TTC method failed to traverse the intersection. For the

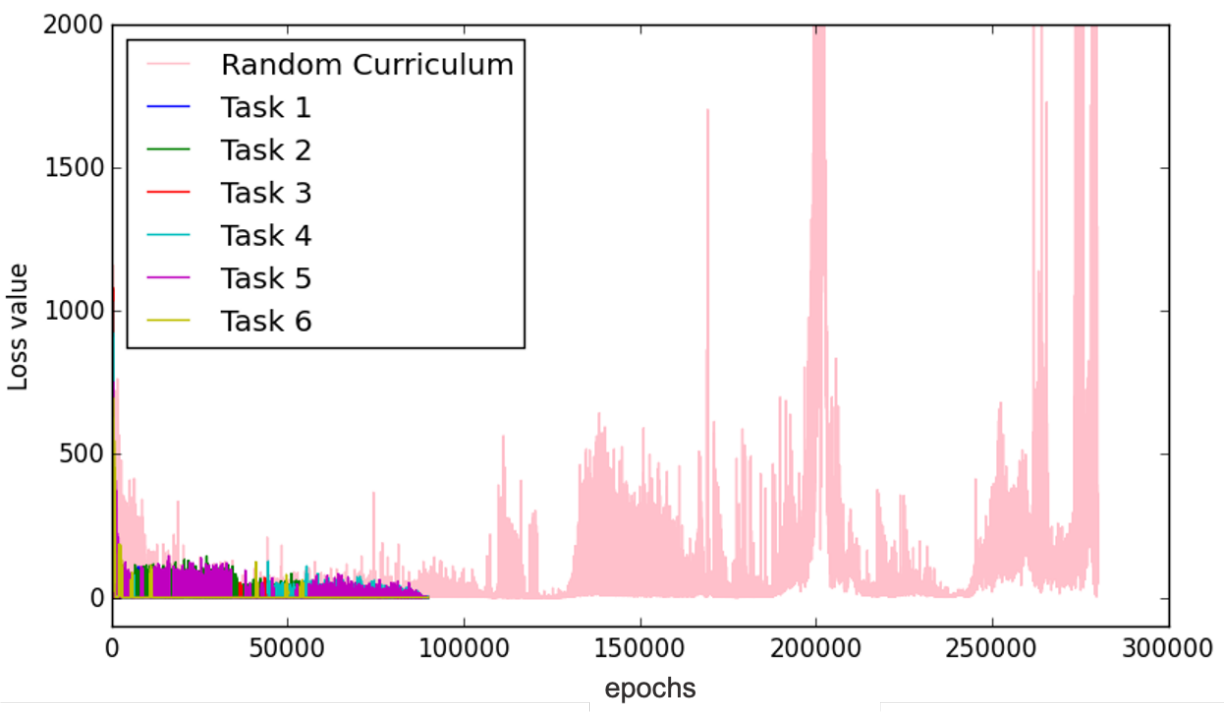

Figure 4.6: Loss Function of Random Curricula and AGC-based Model for the intersection traversing case

intersection approaching scenario, we train the model for 40,000 iterations and the AGC-based model can get a much better performance on the test tasks. Except for the performance of the mean reward, the steps metric also shows better results. We can see that the AGC-based methods can finish the task in fewer time-steps, so that the ego car achieves the destination sooner than with other methods.

4.3 Experiments for RL with LSTM-module and Hierarchical Option MDP

In this section, we present experiments and results corresponding to the proposed RL with LSTM-module policy network (Section 3.5) and Hierarchical Option RL (Section 3.6.1). The method will use multiple historical steps of states to generate actions that have memory of pre-

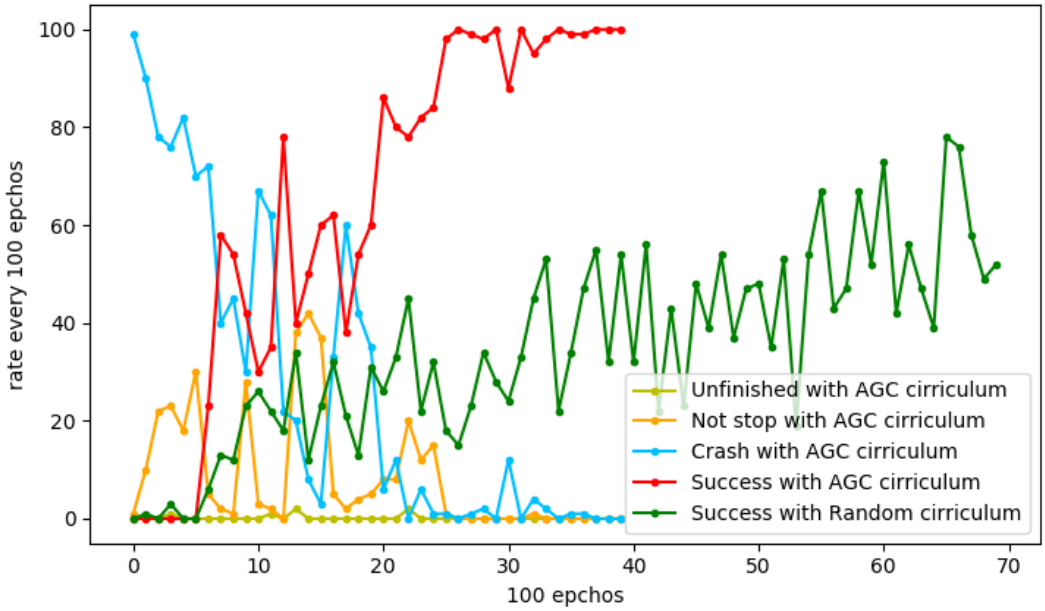

Figure 4.7: Success rate of Random Curricula and AGC-based Model for the intersection approaching case

vious observations in order to deal with scenarios having occlusion of field of view at the intersection.

4.3.1 Scenarios

We tested our algorithm in the SUMO simulator [47] (Figure 4.8) with two scenarios which are shown in Figure 4.9. It is an urban intersection with a stop sign in the lane of the ego-car (blue). The cross-traffic vehicles (yellow) have the right of way, and the ego-car's task is to enter traffic safely without strongly disturbing the behavior of the other vehicles.

In the two scenarios, the initial positions of the ego vehicle, the position of the stop-line, the position of the mid-point of the intersection and the position of the destination are all the same. However, we change the position of the lower bound of the intersection in Scenario 2, which may result in less information from the ego vehicle's sensor ray traces, especially at the beginning of each training epoch. This kind of situation is closer to real-world scenarios, in which the average

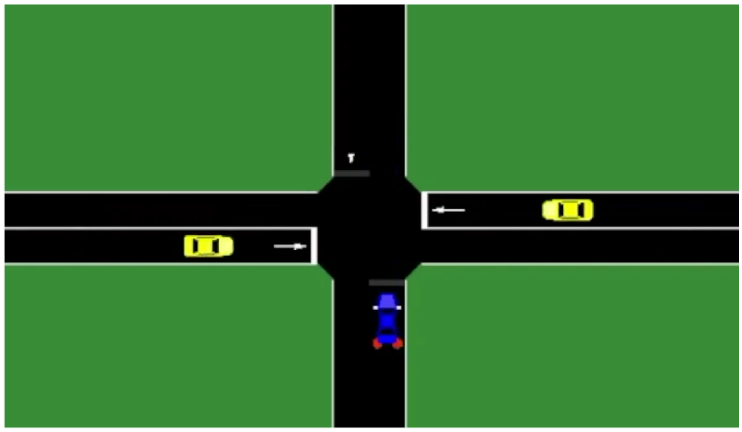

Figure 4.8: Intersection traverse scenarios in SUMO

distance between the lower bound of the intersection and the stop-line is four to six meters. As a result, most vehicles don't roll all the way up to the edge of the intersection. The farther the stop-line is away from the lower bound of the intersection, the more difficult it is to make a proper decision on when to traverse, because the geometry of the intersection blocks visibility.

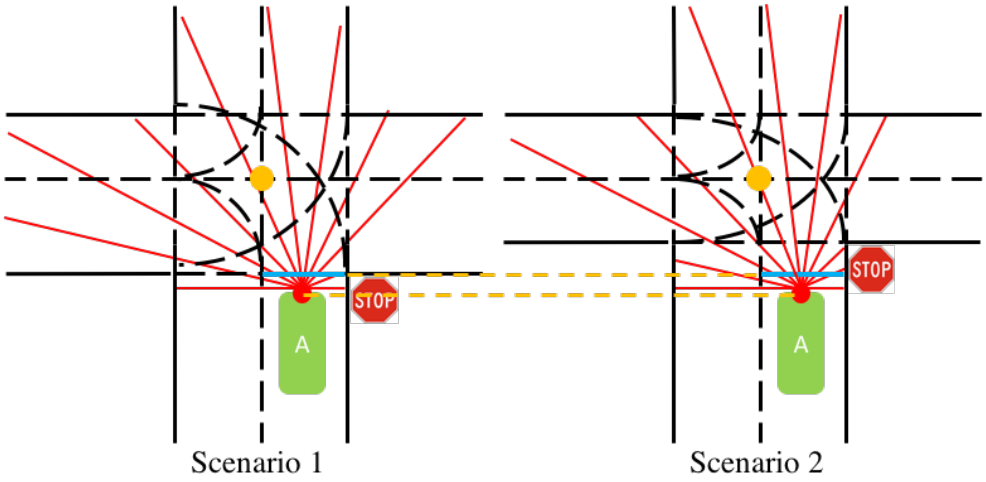

Figure 4.9: Two initial scenarios for experiments

4.3.2 Experiment Setup

In the representation of the state space, we assume that the map information is known by the agent. We define the state space as containing the ego vehicle's velocity v and road geometry

information, including the distance between the ego vehicle and the lower intersection boundary d_{lb}, the mid-point d_{mp} and the destination d_{goal} of the intersection. Figure 4.10 shows the ray-trace geometric information related to the intersection and the generated/simulated rays. The accompanying video[1] shows the simulated ray trace representations during Go Straight, Turn Right and Turn Left at the intersection in SUMO [47].

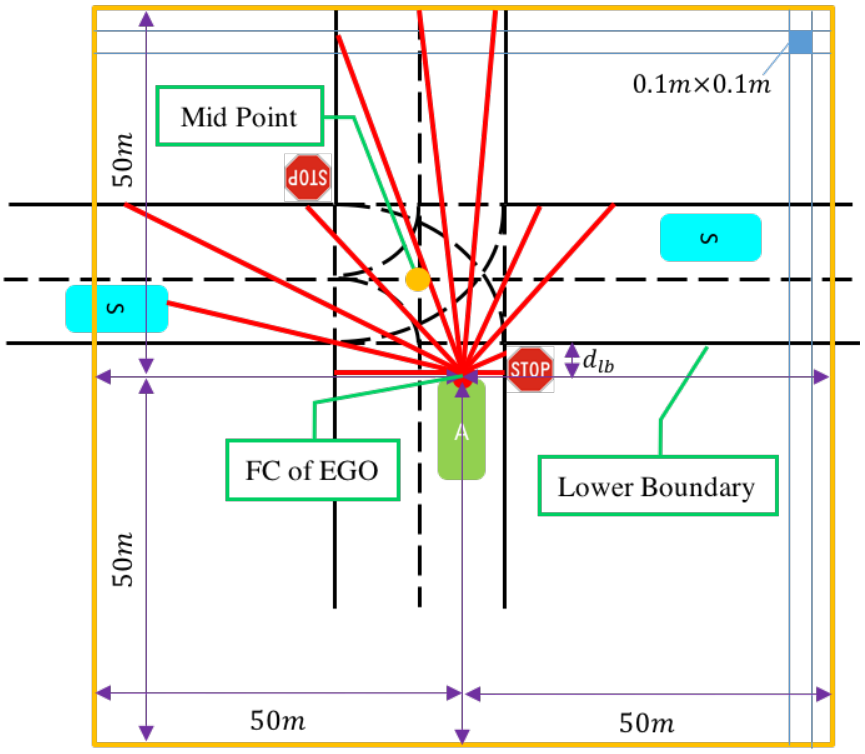

Figure 4.10: Explanation of state space and method of constructing ray trace system. FC means the front center of the ego car.

During the traverse process, the vehicle has to either Go Straight, Turn Right or Turn Left successfully while limiting the required interactions between the ego vehicle and other simulated vehicles. To evaluate the effectiveness of the proposed algorithms, we tested the following four methods:

1. time-to-collision (TTC) [6]

[1] https://youtu.be/5HMB8SWanW4

2. DDPG [17]
3. our proposed POMDP with LSTM algorithm which is described in Section 3.5
4. our proposed Hierarchical Option RL algorithm which is described in Section 3.6.1

For the evaluation we applied the resulting policies for 1000 iterations on the Go Straight, Turn Right and Turn Left task separately. We compared the different algorithms using five metrics:

- **Percentage of Success**: the percentage of epochs in which the ego vehicle successfully reaches the destination.
- **Percentage of Collision**: the percentage of epochs in which the ego vehicle collides with any other simulated vehicle before reaching the destination.
- **Percentage of Unfinished cases**: the percentage of epochs in which the ego vehicle does not achieve the goal within 1000 simulation steps, which equals 100 seconds in the real world.
- **Total steps** $step_{total}$: the average number of total steps the ego vehicle takes to finish each test iteration. Fewer steps means less time or more aggressive actions the ego car is taken to finish the whole process.
- **Percentage of Interaction** $\frac{step_{inter}}{step_{total}}$: the average percentage of steps in each test iteration in which the ego vehicle has an interaction with one of the other traffic participants. An interaction is defined as an unscheduled behavior change of a vehicle in response to the ego vehicle's behavior. Other traffic participants can *react* to the ego vehicle if the ego vehicle has passed the lower bound of the intersection and is within the visibility range of the ego vehicle. More interactions means that the ego vehicle relies more on other vehicles' decelerations to avoid collision.
- **Total Reward** $\sum_{t=0}^{T} r_t$: the reward the agent gets for each test iteration.

4.3.3 Results

First, we compare the results for our two baselines (i.e., TTC and the vanilla DDPG) for both scenarios (Figure 4.9). The results are shown in Table 4.2 . In Scenario One, DDPG outperforms TTC in terms of the success rate, total steps to finish the traversing, and the interaction rate, which is in alignment with other reported results [42] that use bird's-eye-view data instead of ray trace data. However, for Scenario Two, where there are more unobservable states initially and during the training process, DDPG and TTC vary as to which has better performance. Although DDPG has a higher success rate for Go Straight and Left Turn scenarios, it also has a higher interaction rate, which means the lower collision rate is due at least in part to the other vehicles' adjustments instead of the ego vehicle's skill.

Next, we compared RL with LSTM module and HOMDP to get the results for Scenario Two alone, which are presented in Table 4.3. In general, TTC can cause a higher collision rate and may result in more interactions, which means the method relies on the other simulated vehicles to yield to the ego vehicle in order to avoid collision, which is not similar to the situation in the real world. The result from RL with LSTM module is more conservative, which causes more unfinished failure cases, meaning the epochs in which the ego vehicle cannot finish the traversing task within 1000 simulation steps. The HOMDP method, which is our main contribution, can get the highest total reward and success rate for all the Go Straight, Turn Right and Turn Left tasks. For the Go Straight and Turn Left tasks, the interaction rate is lowest with HOMDP, which means the results rely least on the other vehicles' yielding. HOMDP takes more total steps than other methods (TTC for Go Straight and Turn Left task, RL with LSTM module for Turn Right Task), which means HOMDP is a more conservative strategy which values the success rate most.

In Figure 4.11, we visualize typical interactions between the ego car and cross-traffic which has the right of way using the time-to-collision method. We can see that, in these cases, the ego car blocks traffic, causing the other vehicles to stop in order to traverse the intersection successfully. Meanwhile, we also visualize the results for RL with LSTM module and HOMDP

Table 4.2: Results for Scenarios One and Two with TTC and DDPG methods. Bold text indicates the best result.

Task	Metrics	Scenario One		Scenario Two	
		TTC	DDPG	TTC	DDPG
Go Straight	% Success	95.6	**97.8**	95.3	**96.2**
	% Collision	2.6	**2.2**	4.7	**3.1**
	% Unfinished	1.8	**0.0**	**0.0**	0.7
	Total Steps	385	**136**	**70**	195
	% Interaction	44.82	**24.49**	**18.87**	30.02
	Reward	673	**735**	552	**587**
Turn Right	% Success	98.6	**100.0**	98.2	**97.6**
	% Collision	1.4	**0.0**	**1.8**	2.4
	% Unfinished	0.0	0.0	0.0	0.0
	Total Steps	117	**106**	66	**63**
	% Interaction	17.05	**10.14**	13.97	**11.4**
	Reward	743	**792**	665	**721**
Turn Left	% Success	86.6	**97.6**	91.4	**92.4**
	% Collision	**1.0**	2.0	8.6	**5.4**
	% Unfinished	12.4	**0.4**	**0.0**	2.2
	Total Steps	514	**196**	**108**	289
	% Interaction	35.47	**33.26**	39.57	**37.31**
	Reward	-821	**217**	-437	**-223**

in Figure 4.12 and Figure 4.13 respectively. The Hierarchical Option RL performs significantly better when encountering the cross-traffic: the ego agent can usually find a good slot to traverse so that it doesn't interact with coming vehicles frequently and doesn't block the traffic. For more details, the performance of all the policies for TTC, RL with LSTM module and HOMDP can be seen in the video[2].

[2]https://youtu.be/Rj5bYtKshh8

Table 4.3: Results for Scenario Two with Four Methods Together. Bold text indicates the best result.

Task	Metrics	Scenario Two			
		TTC	DDPG	POMDP	HOMDP
Go Straight	% Success	95.3	96.2	97.1	**98.3**
	% Collision	4.7	3.1	**1.7**	**1.7**
	% Unfinished	**0.0**	0.7	1.2	**0.0**
	Total Steps	**70**	195	261	102
	% Interaction	18.87	30.02	27.86	**26.41**
	Reward	552	587	621	**873**
Turn Right	% Success	98.2	97.6	99.5	**99.8**
	% Collision	1.8	2.4	0.5	**0.2**
	% Unfinished	0.0	0.0	0.0	0.0
	Total Steps	66	63	**57**	62
	% Interaction	13.97	11.4	**6.26**	9.28
	Reward	665	721	892	**903**
Turn Left	% Success	91.4	92.4	95.6	**97.3**
	% Collision	8.6	5.4	**2.4**	2.6
	% Unfinished	**0.0**	2.2	2.0	0.1
	Total Steps	**108**	289	276	132
	% Interaction	39.57	37.31	33.21	**28.90**
	Reward	-437	-223	213	**632**

4.4 Experiments for Hierarchical Reinforcement Learning

In this section, we apply the Hybrid Hierarchical RL which was introduced in Section 3.6 to the behavior planning of a self-driving car and make comparisons with competing methods. The main idea of the algorithm is to replace the action space of the original RL with the heuristic-based rules-enumeration strategy. In the tested scenarios, we add leading vehicles in front of the ego car to increase the difficulties with which the agent must deal. The results show that the rule-based algorithm begins to perform worse than in the easier scenarios shown in the previous

Figure 4.11: Interaction between ego car and other simulated vehicles when traversing 2-way stop-sign intersection: Time-to-collision-based method.

Table 4.4: Results comparisons among different behavior policies

	Rewards		Step	Step Penalty		Performance Rate			
	Option Reward r^o	Action Reward r^a		Unsmoothness	Unsafe	Collision	Not Stop	Timeout	Success
Rule 1	-36.82	-9.11	112	0.38	8.05	18%	82%	0%	0%
Rule 2	-28.69	0.33	53	0.32	6.41	89%	0%	0%	11%
Rule 3	26.42	13.62	128	0.54	13.39	31%	0%	0%	69%
Rule 4	40.02	17.20	149	0.58	16.50	14%	0%	0%	86%
Hybrid HRL	43.52	28.87	178	5.32	1.23	0%	7%	0%	93%

tests, whereas our approaches maintain a good performance in these scenarios.

4.4.1 Scenario

We tested our algorithm in MSC's VIRES VTD, which is a complete simulation tool-chain for driving applications [48]. Referring to Figure 4.14, we designed a task in which an autonomous vehicle (green box with A) intends to stop at the stop-line behind a random number of front vehicles (blue boxes with T) which have random initial positions and behavior profiles. The two sub-goals of the task in this scenario are designed as *STOP AT STOP-LINE* (*SSL*) and *FOLLOW FRONT VEHICLE* (*FFV*).

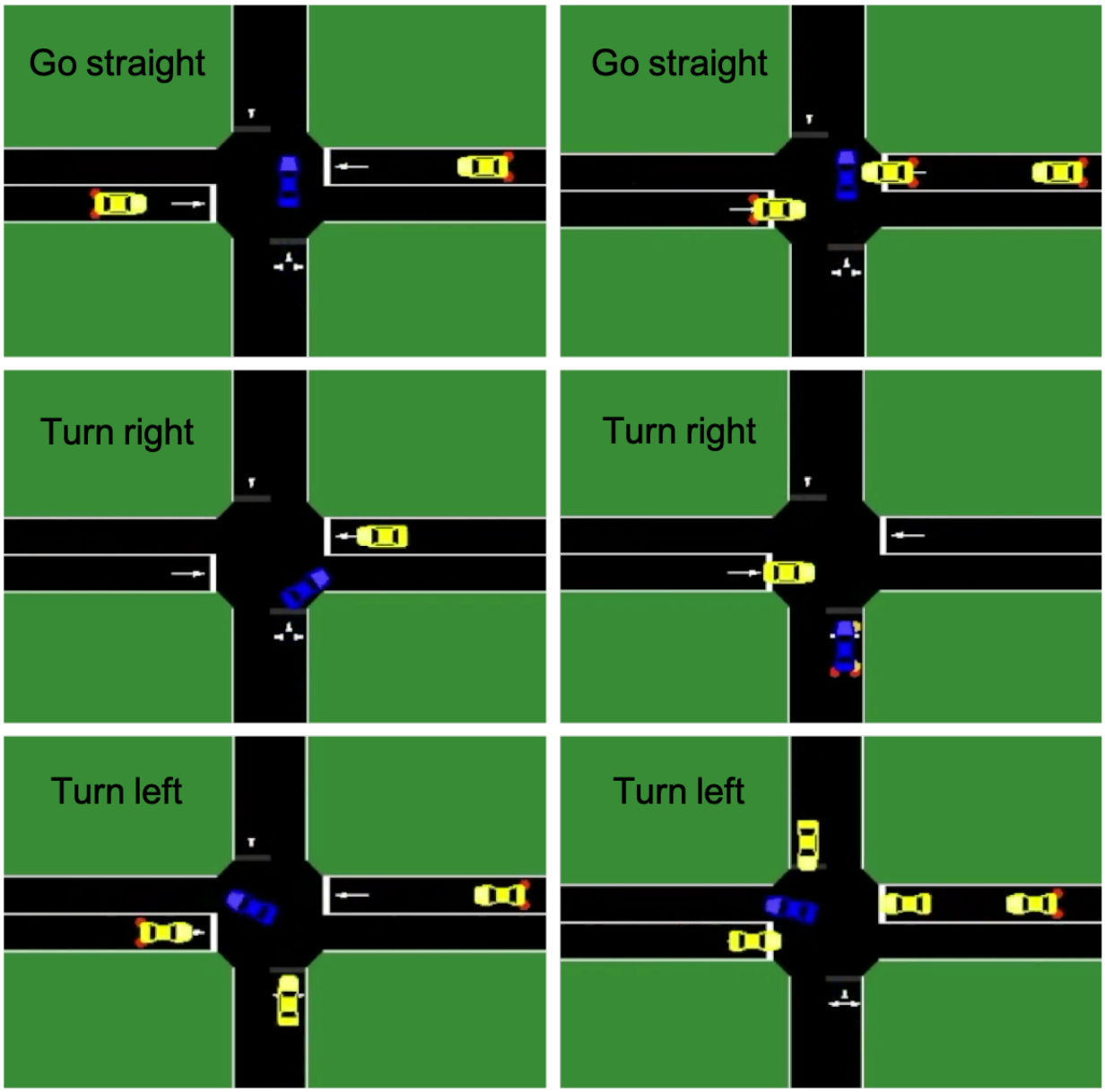

Figure 4.12: Interaction between ego car and other simulated vehicles when traversing 2-way stop-sign intersection: RL with LSTM module.

4.4.2 Transitions

State

The state which is used to formulate the hierarchical deep reinforcement learning includes the information of the ego car, which is useful for both sub-goals, and the related information that is

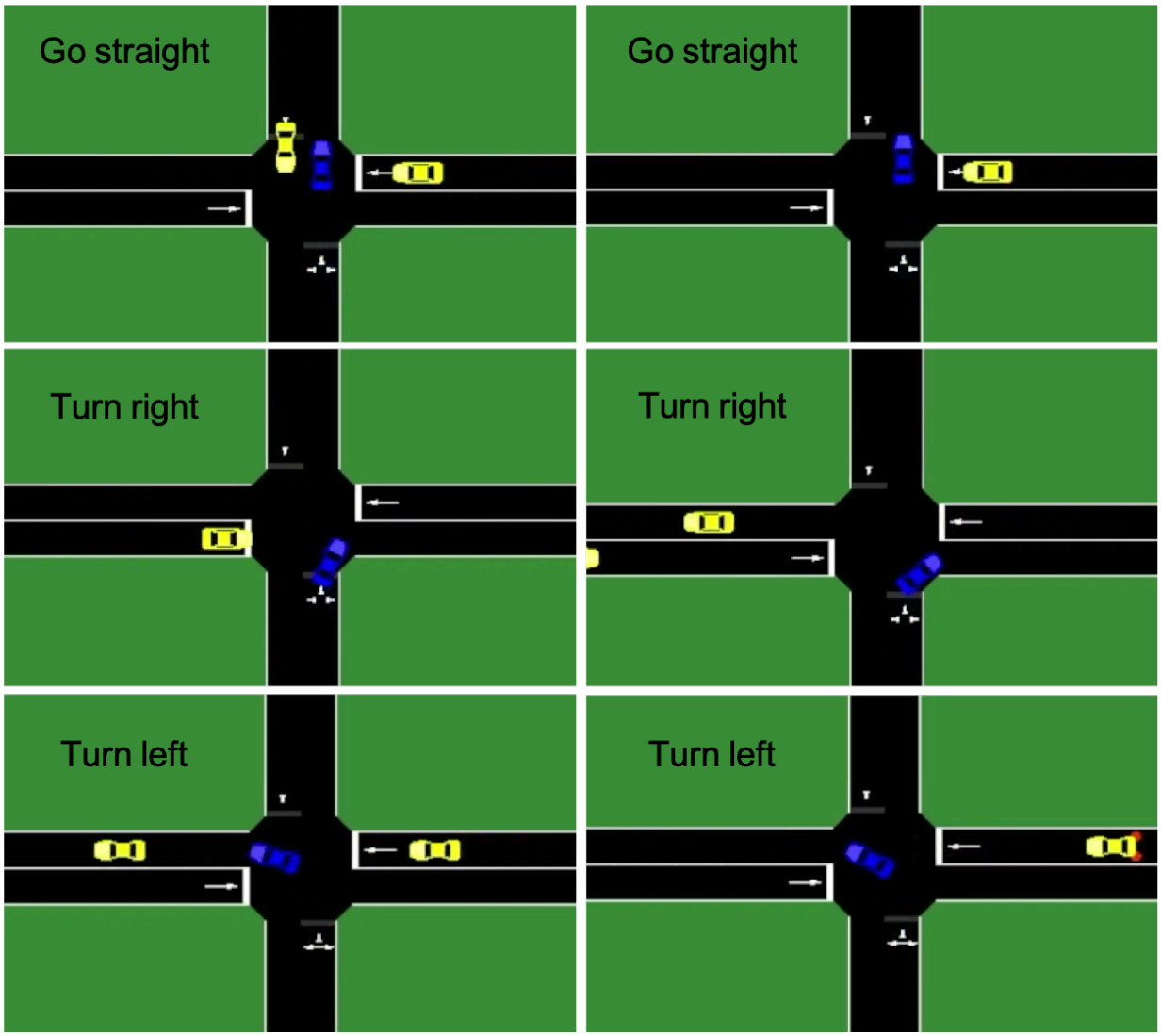

Figure 4.13: Interaction between ego car and other simulated vehicles when traversing 2-way stop-sign intersection: Hierarchical Option RL.

needed for each sub-goal.

$$s = \left[v_e, a_e, j_e, d_f, v_f, a_f, d_{fc}, \frac{d_{fc}}{d_{fs}}, d_d, d_{dc}, \frac{d_{dc}}{d_{ds}} \right] \tag{4.1}$$

Equation 4.1 describes our state space, where v_e, a_e and j_e are respectively the velocity, acceleration and jerk of the ego car, while d_f and d_d denote the distance from the ego car to the nearest front vehicle and the stop-line, respectively. A safety distance parameter is introduced as a nominal distance behind the target object which can improve safety due to different sub-goals.

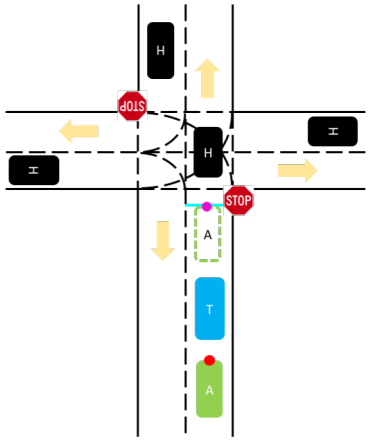

Figure 4.14: Autonomous vehicle (green box with A) approaching stop-sign intersection

$$
\begin{aligned}
d_{fs} &= \max\left(\frac{v_e^2 - v_f^2}{2a_{max}}, d_0\right), \quad d_{fc} = d_f - d_{fs} \\
d_{ds} &= \frac{v_e^2}{2a_{max}}, \quad d_{dc} = d_d - d_{ds}
\end{aligned}
\tag{4.2}
$$

Here a_{max} and d_0 denote the ego car's maximum deceleration and minimum allowable distance to the front vehicle, respectively, and d_{fc} and d_{dc} are the distance to the front car and the safety margin distances between ego car and front vehicle. The initial positions of all the simulated vehicles in front of the ego car are spaced arbitrarily. The distance from the nearest front vehicle and the ego car is randomly set between the safety margin and the max range of sensor.

Option and Action

The option network in the scenario outputs the selected sub-goal: *SSL* or *FFV*. Then, according to the option result, the action network generates the throttle or brake choices.

Reward Functions

Assume that for one step, the selected option is denoted as o, $o \in \{d, f\}$. The reward function is given by:

For each step:

- Time penalty: $-\sigma_1$.
- Unsmoothness penalty if jerk is too large: $-\mathbb{I}_{j_e>1.}\sigma_2$.
- Unsafe penalty: $-\mathbb{I}_{d_{dc}<0}\exp(-\frac{d_{dc}}{d_{ds}}) - \mathbb{I}_{d_{fc}<0}\exp(-\frac{d_{fc}}{d_{fs}})$.

For the termination conditions:

- Collision penalty: $-\mathbb{I}_{d_f=0.}\sigma_3$.
- Not stop at stop-line penalty: $-\mathbb{I}_{d_d=0.}v_e^2$.
- Timeout: $-\mathbb{I}_{\text{timeout}}d_d^2$.
- Success reward: $\mathbb{I}_{d_d=0.,v_e=0}\sigma_4$

where σ_k are constants. $\mathbb{I}_c$ are indicator functions. $\mathbb{I}_c = 1$ if and only if c is satisfied, otherwise $\mathbb{I}_c = 0$.

Assume that for one step, the selected option is denoted as o, $o \in \{d, f\}$ and the unselected option is o^-, $o^- \in \{f, d\}$:

$$\begin{aligned} sr &= -\sigma_1 - \mathbb{I}_{\text{timeout}}d_d^2 + \mathbb{I}_{d_d=0.,v_e=0}\sigma_4 \\ r^{option} &= sr - \mathbb{I}_{d_{o^-c}<0}\exp(-\frac{d_{o^-c}}{d_{o^-s}}) - \mathbb{I}_{d_{o^-}=0.}v_e^2 \\ r^{action} &= sr - \mathbb{I}_{j_e>1.}\sigma_2 - \mathbb{I}_{d_{oc}<0}\exp(-\frac{d_{oc}}{d_{os}}) - \mathbb{I}_{d_o=0.}\sigma_3 \\ r^{task} &= sr - \mathbb{I}_{d_{dc}<0}\exp(-\frac{d_{dc}}{d_{ds}}) - \mathbb{I}_{d_{fc}<0}\exp(-\frac{d_{fc}}{d_{fs}}) \\ &\quad - \mathbb{I}_{j_e>1.}\sigma_2 - \mathbb{I}_{d_f=0.}\sigma_3 - \mathbb{I}_{d_d=0.}v_e^2 \end{aligned} \tag{4.3}$$

where sr represents the portion of the reward common to r^{option}, r^{action} and r^{task}.

For comparison, we also formulate the problem without considering a hierarchical model via Double DQN. Then r^{task} denotes the reward for achieving the task in this flattened action space.

4.4.3 Results

We compare the proposed algorithm with four rule-based algorithms and some traditional RL algorithms mentioned before. Table 4.4 shows the quantitative results for testing the average

Table 4.5: Different HRL-based policies

	Hybrid Reward	Hierarchical PER	Attention Model
HRL^0	$\times$	$\times$	$\times$
HRL^1	$\surd$	$\times$	$\times$
HRL^2	$\surd$	$\surd$	$\times$
HRL^3	$\surd$	$\times$	$\surd$
Hybrid HRL	$\surd$	$\surd$	$\surd$

performance of each algorithm over 100 cases.

The competing methods include:

- Rule 1: the high-level option always chooses to *Follow Front Vehicle* (*FFV*).
- Rule 2: the high-level option always chooses to *Stop at Stop-line* (*SSL*).
- Rule 3: if $d_d > (d_f + car\_length)$, select *FFV*, otherwise choose *SSL*.
- Rule 4: if $d_f > d_{fc}$, select *FFV*, otherwise choose *SSL*.
- Table 4.5 shows the techniques used in several different HRL-based algorithms, whose results are shown in Figure 4.15.

Figure 4.15 compares the Hybrid HRL method with different setups of HRL algorithms. The results show that the hybrid reward mechanism can perform better with the help of the hierarchical PER approach.

Figure 4.16 depicts a typical case of the relative speed and position of the ego vehicle with respect to the nearest front vehicle as they both approach the stop-line. In the bottom graph we see the ego vehicle will tend to close the distance to the front vehicle until a certain threshold (about 5 meters) before lowering its speed relative to the front vehicle to allow a certain buffer between them. In the top graph we see that during this time the front vehicle begins to slow rapidly for the stop-line at around 25 meters out before taxiing to a stop. Simultaneously, the ego

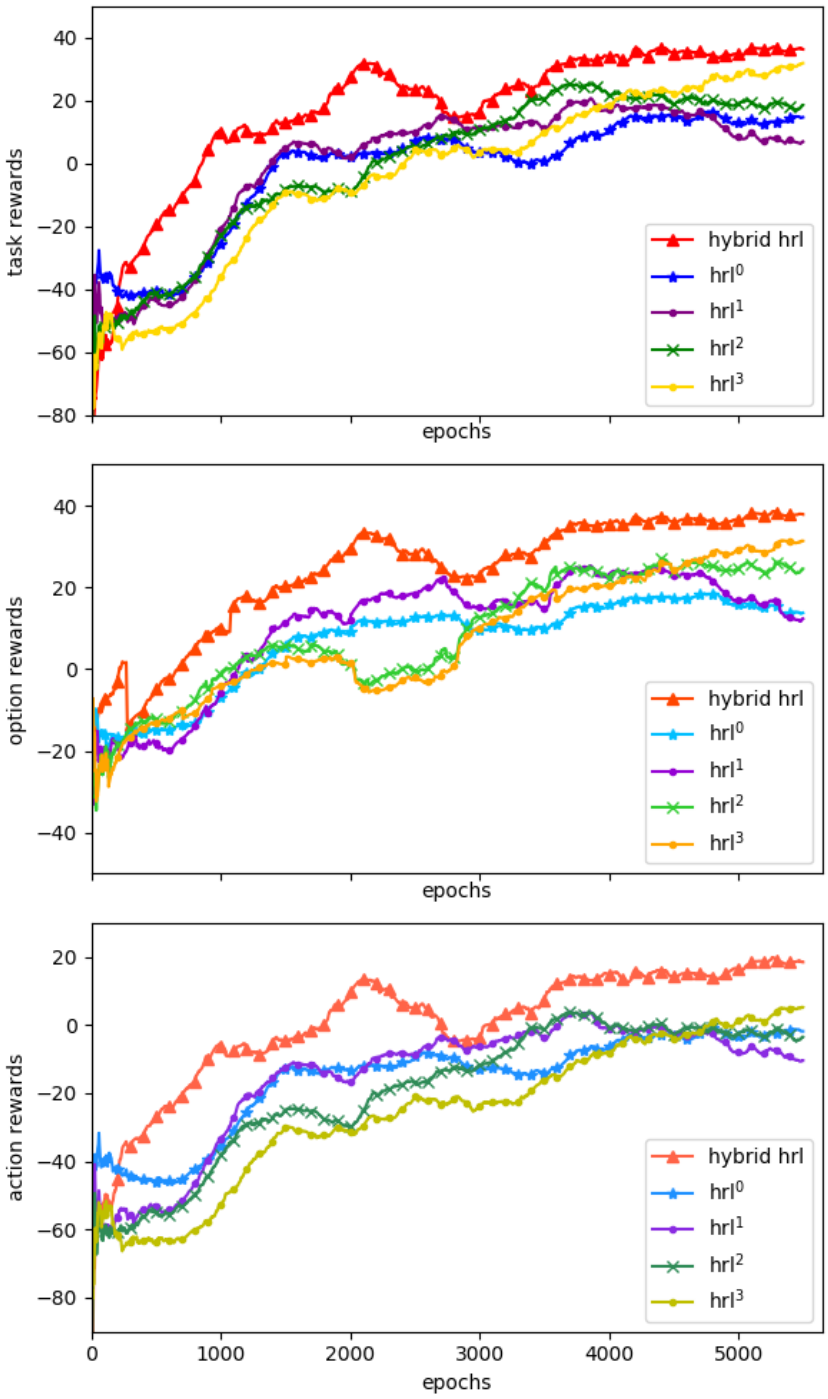

Figure 4.15: Training results for HRL with various combinations of technologies we proposed (Table 4.5). For each test result in the figure, it is an average performance from multiple training.

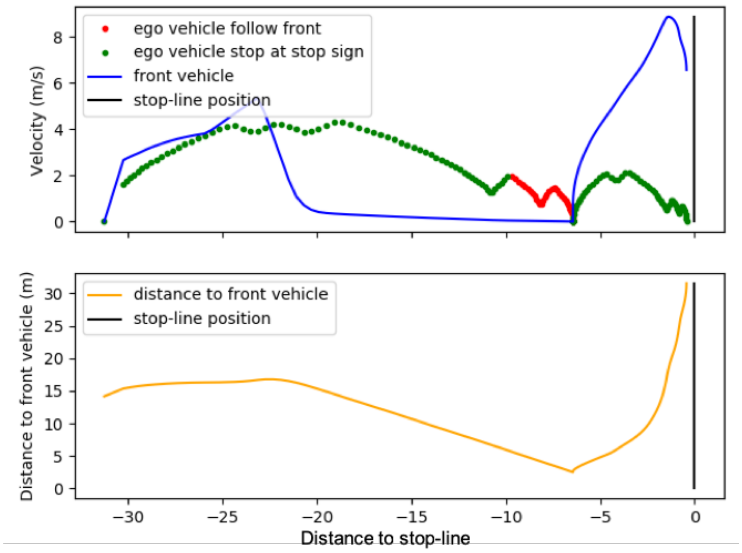

Figure 4.16: Velocities of ego car and front vehicles

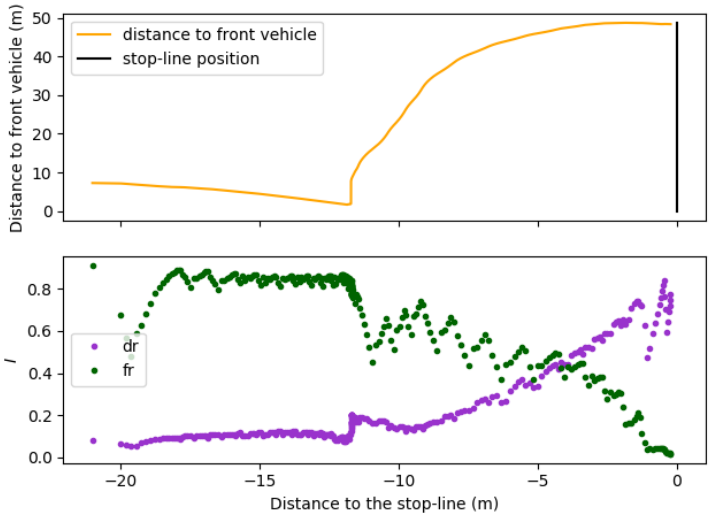

Figure 4.17: Attention value extracted from the attention layer in the model. dr and fr are $\frac{d_{dc}}{d_{ds}}$ and $\frac{d_{fc}}{d_{fs}}$ in the introduced state, respectively.

vehicle opts to focus on stopping for the stop-line until it's within a certain threshold of the front vehicle, at which point it will attend to the front vehicle instead. Finally, after a pause the front vehicle accelerates through the stop-line and at this point the ego vehicle immediately begins focusing on the stop sign once again as desired.

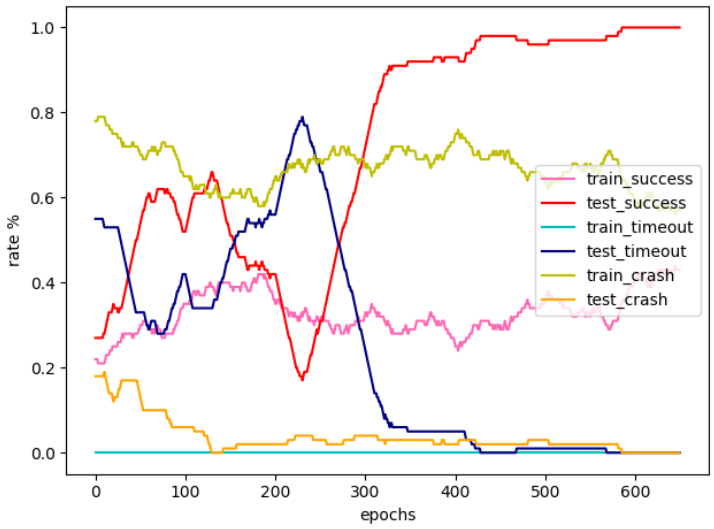

Figure 4.18: Performance rate of only training to *Follow Front Vehicles* during the training process. Results from training include random actions taken according to explorations. Results from testing show average performance by testing 200 cases based on the trained network after that training epoch.

Figure 4.17 shows the results extracted from the attention *softmax* layer. Only the two state elements with the highest attentions have been visualized. The upper sub-figure shows the relationship between the distance to the nearest front vehicle (y-axis) and the distance to the stop-line (x-axis). The lower sub-figure is the attention value. When the ego car is approaching the front vehicle, the attention is mainly focused on $\frac{d_{fc}}{d_{fs}}$. When the front vehicle leaves without stopping at the stop-line, the ego car transfers more and more attention to $\frac{d_{dc}}{d_{ds}}$ during the process of approaching the stop-line.

For the scenario of approaching the intersection with front vehicles, one of the methods is to manually design all the rules. Another possibility is to design a rule-based policy of stopping at the stop-line, which is relatively easy to model. Then we train a DDQN model (see Figure 4.18 for the training process) to be the policy of following front vehicles. Based on these two action-level models, we train another DDQN model (see Figure 4.19 for the training process) to be the policy governing which option is needed for approaching the stop-line with front vehicles. Dur-

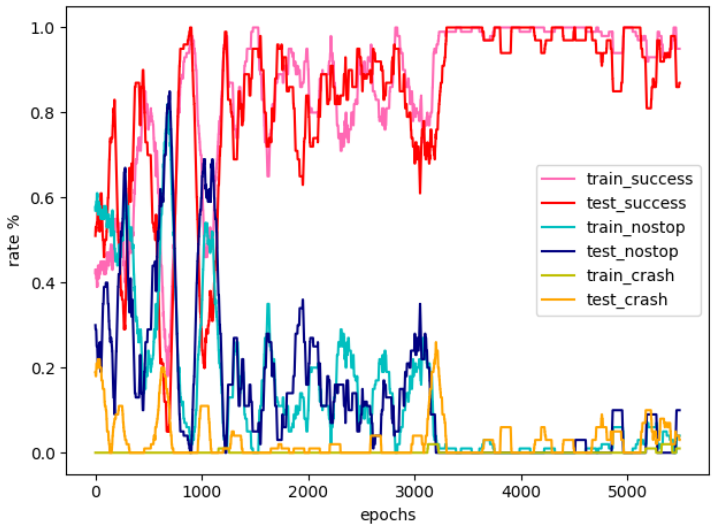

Figure 4.19: Performance rate of only training to choose the options between *FFV* or *SSL* based on the designed rule-based or trained action-level policies. Results from Test show average performance by testing 100 cases based on the trained network after that training epoch.

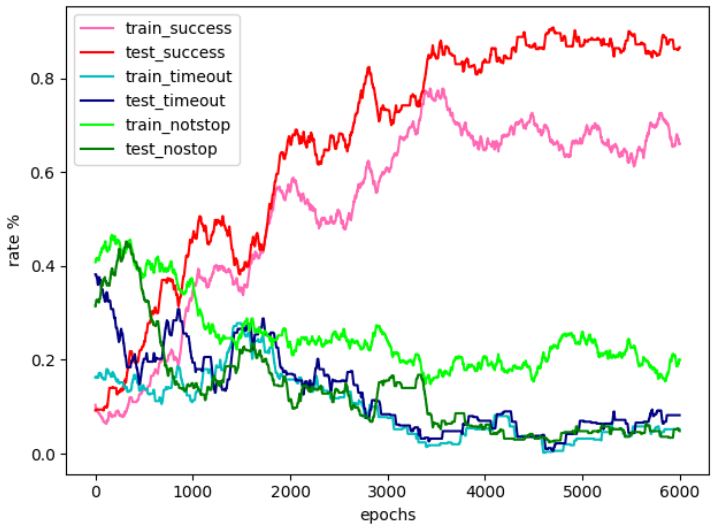

Figure 4.20: Performance rate of Hybrid HRL training process. Results from testing show average performance by testing 500 cases based on the trained network after that training epoch.

ing the training process, after every training epoch, the simulation will test 500 epochs without action exploration based on the trained-out network. By applying the proposed hybrid HRL, all

the option-level and action-level policies can be trained together (see Figure 4.20 for the training process) and the trained-out policy can be separated if the target task only needs to achieve one of the sub-goals. For example, the action-value network of *Following Front Vehicle* can be used alone with the corresponding option input to the network. Then, the ego car can follow the front vehicle without stopping at the stop-line.

4.5 Summary

In this chapter, we designed multiple experiments for autonomous vehicles to deal with different scenarios at urban intersections. The results show that ACG-based RL (Section 4.2) can significantly reduce the training time while achieving similar or better performance. An LSTM-based module (Section 4.3) embedded in the actor-critic structure can help to improve the performance of the agents for POMDP scenarios. The experiments show that the proposed extensions to hierarchical reinforcement learning achieve improved convergence speed, sample efficiency and scalability over traditional RL approaches. Results to date suggest our algorithm is a promising candidate for future research, as it is able to outperform a suite of hand-engineered rules on a simulated autonomous driving task in which the agent must pursue multiple sub-goals in order to succeed.

Chapter 5

Real-World Experiments

5.1 Overview

In this chapter, we apply the algorithms described in the previous chapters to some real-world urban intersection scenarios. We first introduce UrbanFlow in Section 5.2, a data collection and processing system we developed for urban intersection scenarios. Then based on some particular cases, we compare the results obtained from heuristic-based rule-enumeration with those obtained from our reinforcement-learning-based algorithms in Sections 5.3 and 5.4.

5.2 UrbanFlow

The current state of the art in acquiring and using real-world human driving data faces several problems. First, some published work relies on privately collected datasets, the inaccessibility of which makes them impossible to use as benchmarks for comparisons between various algorithms. Second, some datasets are collected by autonomous vehicles from the perspective of the ego vehicle. Although this perspective is ultimately the one available to an autonomous vehicle, it is difficult for it to provide full sequences showing the social behavior of surrounding vehicles. To derive models for such behavior, bird's-eye view datasets are useful. In response to these

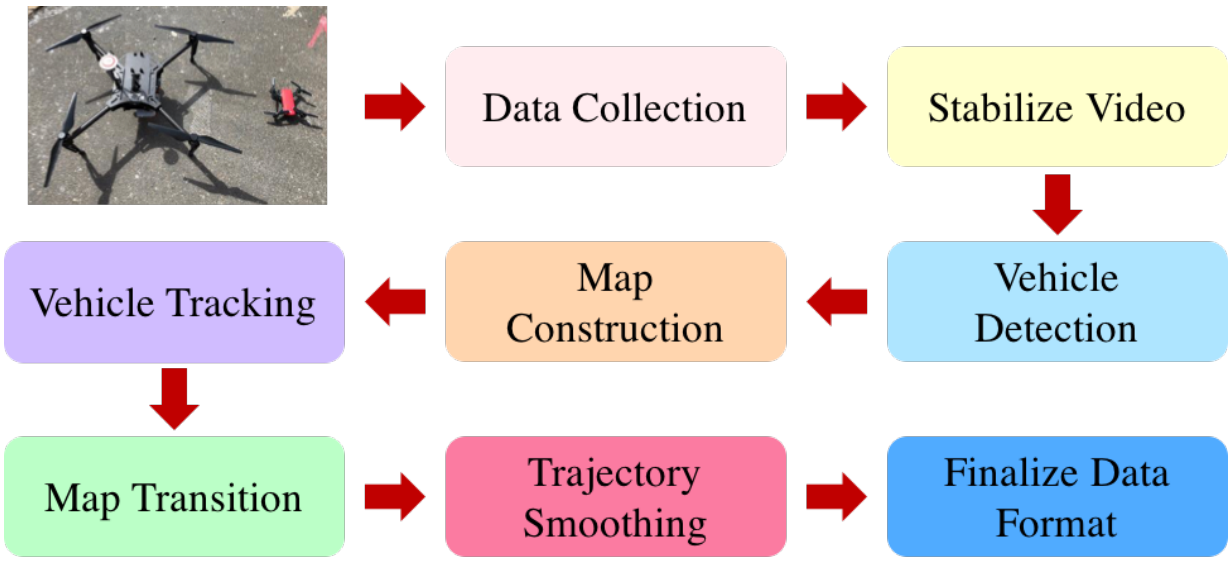

Figure 5.1: The UrbanFlow dataset processing pipeline. The pipeline includes the drone data collection and process flow from raw video data to the final trajectory data.

problems, we developed a method for benchmarking human driver behavior based on a bird's-eye-view data collection system via drone. Figure 5.1 shows the pipeline of the data processing procedure.

5.2.1 Video Stabilization

The two main challenges for video stabilization are the robustness and the speed of the alignment [49][50]. In this paper, we propose several steps for the video stabilization in order to deal with the displacement of the drone during the data collection process. Figure 5.2 visualizes the flow of the stabilization method. For each frame f_n at time step n, the algorithm chooses a reference frame f_{ref} according to the alignment evaluation score gotten from the result of the last time step and corresponding homography matrix in order to get the stabilized frame. Firstly, a re-alignment is performed when the result of the ECC alignment score is lower than a threshold. ECC takes a lot of time to converge and is not adaptive to align the current frame with a reference frame when their similarity is lower than a threshold. Secondly, since the alignment is time-consuming,

it is only performed when a reference frame needs to be re-chosen. The homography matrix is re-used for the following frames until a new reference frame is chosen when the evaluation score drops to the threshold. Then, the homography matrix calculated from ECC alignment during the previous step is used for initializing the guess for ECC in the next step to speed up the convergence. Lastly, images are down-sampled [51] so that ECC uses fewer pixels during the calculation.

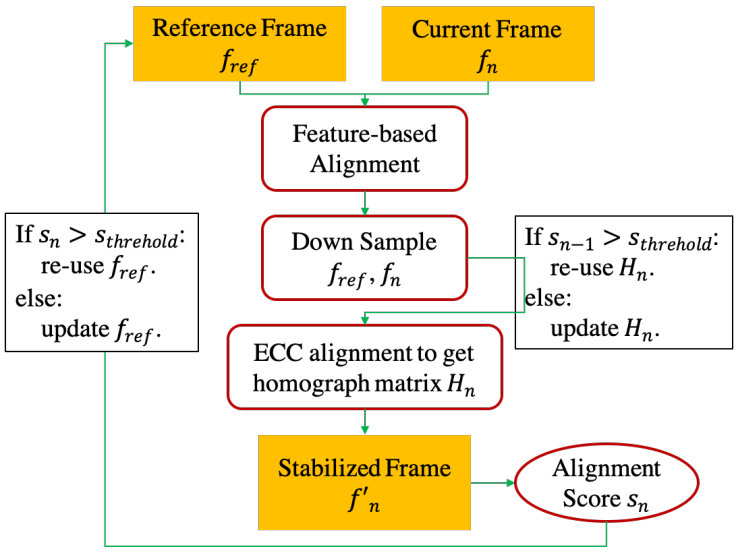

Figure 5.2: Optimized stabilization method flow.

5.2.2 Object Detection

In the proposed pipeline, RetinaNet [52] is used for detecting vehicles in the images. The training dataset contains all the bounding boxes and their corresponding labels for each image. The input images are re-sized to ensure that the size of detection objects is greater than 32-by-32 pixels as well as not too large for the GPU computational capability. Images are masked to crop out the roads in order to make detection easier. RetinaNet was fine-tuned using pre-trained weights from the COCO dataset [53].

5.2.3 Map Construction and Coordinate Transition

The first step in the creation of the map is to crop the area of interest, which in this case is the roads. To attack this problem, we took advantage of the image segmentation network "U-net", described in Ronneberger et al. [54], with just a few adjustments based on the work of Iglovikov et al. [55]. We preserve the decoder section of the network because by adding a large number of feature channels, it allows the network to propagate context information to the higher-resolution layers. The important change was in the encoder section, where it was replaced by the down-sampling elements of the VGG16 architecture in order to take advantage of the pre-trained weights in ImageNet [56], due to the limited quantity of the collected data.

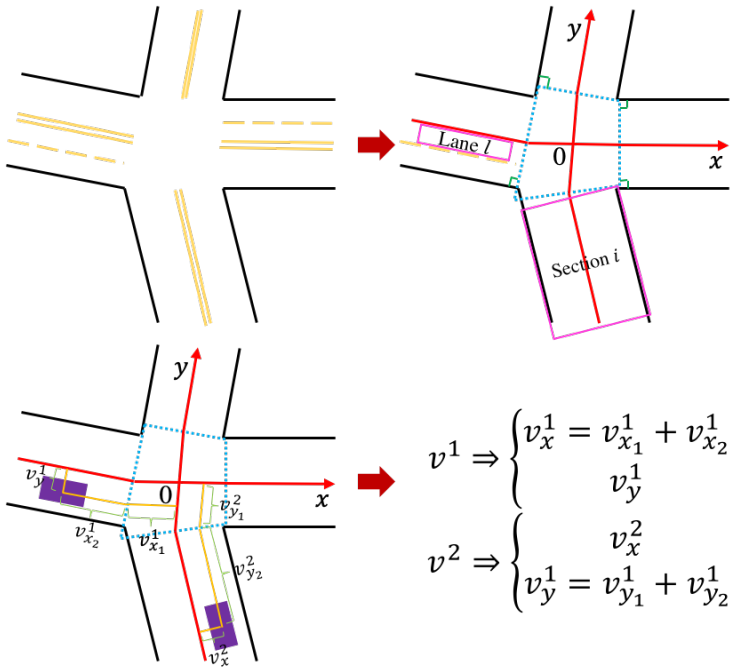

Figure 5.3: Transition from original image-based coordinate to road-based Coordinate

After detecting the road and applying a color filter to detect the lane markings on the road, we transform all the detected vehicle positions from the original image-based coordinates to the road-based coordinates. Figure 5.3 shows the method for generating the road-based coordinates based on a random road geometry which may occur in the real world. The method first chooses an

origin and then proceeds to obtain the x-axis and y-axis along the lane markings which separate the opposite directions of moving vehicles. For the given vehicles v^1 and v^2, the figure shows two examples of how to extract the road-based positions. Finally, it is able to represent the vehicles' information, which contains the following items:

- Local x and y based on the road-based coordinates
- Vehicle length and width
- Section ID i
- Lane ID l

5.2.4 Vehicle Tracking and Trajectory Smoothing

A Kalman filter [57] is used for tracking and trajectory smoothing. Based on the car's dynamic model, characteristics of the system noise and measurement noise, the measurement variables are used as the input signal, and the estimation variables that we need to know are the output of the filter. After the positions of vehicles have been transformed into the local (road) coordinates, we apply the tracking algorithm to track each car. Meanwhile, we smooth each vehicle's trajectory. In the system, we use vehicle position as the state variable. F is the state transition matrix and H is the measurement matrix. $V_q(n)$ and $V_p(n)$ represent the system noise and measurement noise, respectively.

5.3 Urban Scenarios

In this section, we will show the results of applying HybridHRL to both two-level and three-level planning systems in some urban scenarios. Figure 5.4 shows the hierarchical structure for the behavior planning system with two or three decision making layers. The two-layer setup includes a high-level behavior planner and a low-level controller. For a traditional planning system, the trajectory planner can increase the car's control stability. As a result, in the three-layer setup, we

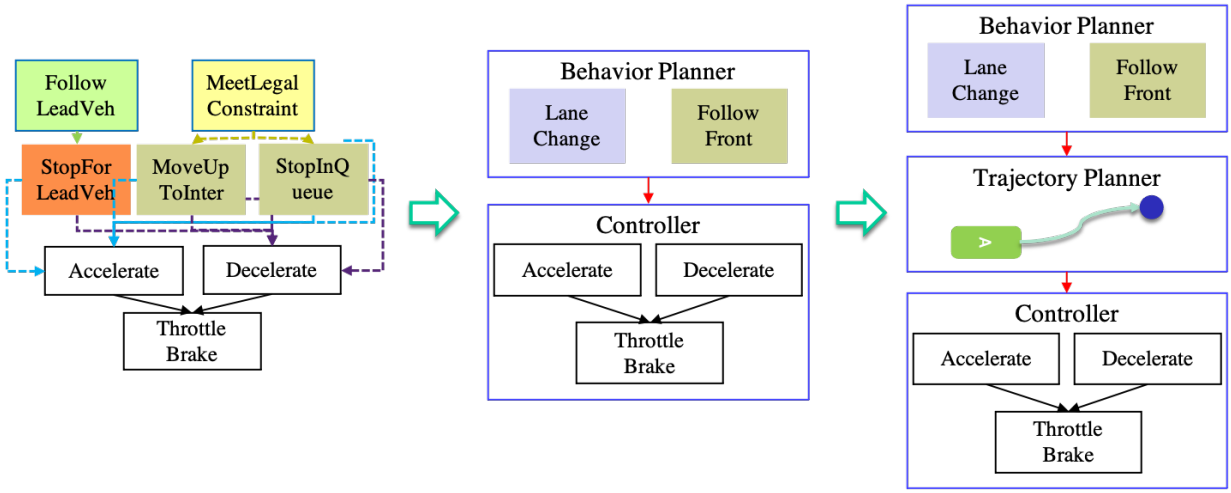

Figure 5.4: Hierarchical structure of planning system with hierarchical reinforcement learning

add an intermediate layer in order to output the target trajectory points. The system would first output the high-level behavior decision. Then, depending on the decision, the trajectory layer generates a corresponding target waypoint that the ego car intends to follow. Finally, the action layer outputs the throttle, brake and steering angle.

Meanwhile, according to the scenarios we extracted from the UrbanFlow dataset, we compare our algorithms with heuristic-based rules enumeration policies, as well as previous RL approaches. We tested our algorithm in MSC's VIRES VTD (Virtual Test Drive) simulator.

5.3.1 Scenarios

Based on traffic data collected at an urban intersection in Pittsburgh, PA using UrbanFlow [58], we identified two important cases where the human needs to make decisions that strongly depend on human maneuvering skills. The accompanying video[1] gives a clear idea of the behaviors performred by Car 1 and Car 2. Figure 5.5 shows a snapshot of the two cases we will consider here:

1. For Car 1: The white car is trying to turn left at a green light. However, in Pittsburgh,

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

Figure 5.5: Human driver scenarios requiring skilled maneuvering to get through an urban intersection

most green traffic lights control both vehicles going straight and vehicles turning left. As a result, the white car is blocked by vehicles going straight approaching from the opposite direction. When training the policy in simulation, the initial positions of the ego agent and other vehicles in the scene are all randomly assigned. There is no vehicle in front of the ego agent initially.

2. For Car 2: The black car is trying to go straight at a green light. But because the white car described above is blocked, the black car is blocked as well. As a result, the driver needs to make a lane change in order to traverse the intersection during the current green light. When training the policy in the simulation, the initial positions of the ego agent and other vehicles in the scene are all randomly assigned. Either the left lane or right lane is blocked randomly. Sometimes, there is no lane blocked and the ego vehicle is not required to perform a lane-change behavior.

5.3.2 MDP Transitions

In this part, we will introduce how to design the state, action and reward factors in the Markov Decision Process for RL training.

State

For each of the two scenarios, we choose three nearest target vehicles to be included in the state space. Figure 5.6 shows the ego agent and target vehicles under the two scenarios. The features included in the state space can be categorized into two parts:

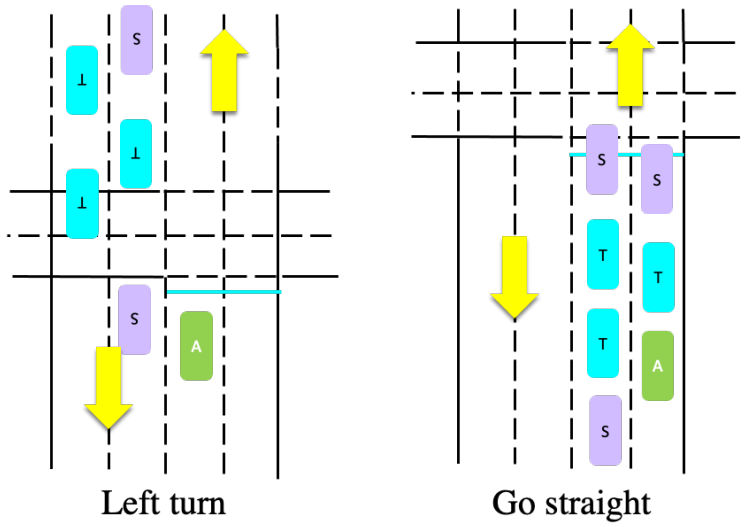

Figure 5.6: Green objects with letter A are the ego agent for two different scenarios. Cyan objects with letter T are the selected target vehicles whose features are included in the state space.

- Ego agent features:

  1) Vehicle dynamic features: velocity v_{ego}, acceleration a_{ego}, heading angle h_{ego};

  2) Localization features: lateral d_{interx} and longitudinal d_{intery} distance related to the center of the intersection, lateral distance to ego lane's center d_{lc} and target lane's center d_{tlc};

  3) For **Left-Turn** scenario: Subjective intention of waiting time feature: time spent waiting at the intersection t;

- Target vehicle features:

  1) Vehicle dynamic features: velocity v_t; acceleration a_t; heading angle h_t;

  2) Position relationship with ego car: relative distance d_t to the ego agent; We use d_f to specify the distance from ego to the front vehicle.

  3) For **Left-Turn** scenario: time to collision ttc corresponding to the ego agent.

  4) For **Go-Straight** scenario: safe margin distance from front vehicle to ego car:

$$d_s = \frac{v_{ego}^2/a_{ego} - v_{front}^2/a_{front}}{2} + \sigma_1,$$

  distance to safe margin corresponding to front vehicle: $d_c = d_t - d_s$ and safety coefficient: $e = \frac{d_c}{d_s}$.

Action

The hierarchical structure of the actions is designed according to the heuristic-based rules-enumerations decision-making system that was introduced in Chapter 3, Figure 3.2. In Figure 5.4, we also show an example of how the actions are designed based on the *FollowLeadVehicles* scenario. Based on the two-lane intersection traverse problem, we list our actions as follows:

- High-level behavior planner:
  - Left-turn scenario: *LaneChange* or *FollowFrontVehicle*.
  - Go-straight scenario: *LeftTurn* or *Wait*.
- Intermediate-level trajectory planner: the waypoint the agent intends to follow.
- Low-level controller: throttle or brake, steering angle

Reward function

The reward function design is based mainly on factors that appear to be important based on human driving experience, which is subjective. But for this scenario, one can see that if one

drives conservatively, Car 1 (that intends to turn left) may be stuck at the intersection until the light turns red (which is what actually happens in the video). Car 2, which intends to go straight, needs to wait until the next green-light period, and it is possible that another car in front of Car 2 also wants to make a left turn, so that Car 2 will be stuck for quite a long time. As a result, we design the reward with aggressive intention to traverse the intersection as quickly as possible. The reward function can be categorized into two parts:

- For each step:
  1. Time penalty $-\sigma_1$;
  2. Unsmoothness penalty if jerk is too large $-\mathbb{I}_{jerk>1}\sigma_2$;
  3. Unsafe penalty $-\mathbb{I}_{d_c<0}\exp -d_c/d_s\sigma_3$;
  4. Penalty for deviating from the lane center when doing lane keeping $-\mathbb{I}_{d_{lc}>1}\sigma_4 d_{lc}$.
- For the termination conditions:
  1. Collision penalty $-\mathbb{I}_{d_f=0}\sigma_5$;
  2. Timeout penalty. For example, the agent is stuck or waits too long to move forward $-\mathbb{I}_{timeout}d^2_{destination}$;
  3. Out of road penalty $-\mathbb{I}_{d_{lc}>lane_{width}}d^2_{lc}$;
  4. Success reward σ_6.

All the sigma parameters are tuned due to various road structure and scenarios.

5.3.3 Results

Left-turn scenario

In Table 5.1, we compare HybridHRL to the heuristic-based rules-enumeration policy which is based on predicting the time difference of arrival at the potential collision point between the target vehicles and the ego agent. Not only is the success rate improved by 47%, but each of

Table 5.1: Results comparisons between heuristic-based rules-enumeration and HybridHRL for left-turn scenario while applying two-layer planning system

	Rewards			Steps	Performance Rate			
	Behavior Reward	Throttle Reward	Steering Reward		Collision	Out of Road	Timeout	Success
Rule	3.64	-16.50	0.48	249	19.35%	11.83%	17.20%	51.61%
HybridHRL	82.98	71.97	87.37	52	0.8%	0.4%	0.0%	98.8%

the failure metrics is reduced to less than 1%. For reward evaluated from each behavior level, we found that both the behavior and controller decisions have higher rewards than the rule-based method.

In Figure 5.7, we show the training and test reward against the number of epochs. During the learning process, we train the agent for 10 epochs and then test for 10 epochs. For the training process, the adjusted heuristic exploration approach 8 will be used and for the test process, the actions are generated without exploration, which means it shows the real capability of the policy network. During the initial training stage, we can see that with the help of heuristic-based exploration, the training policy works much better than the test policy. However, after the test policy begins to converge, the performance of each of the three networks surpasses that of the training policy and begins to achieve better results.

In Figure 5.8, we compared the successful rate between the test policy results and the rule-based method. As the traffic shown in the video is quite dense, Car 1 has difficulty figuring out a good slot through which to make a left turn, even for the human driver in the video. The learning-based methods can surpass the rule-based method after about 2000 epochs of training.

For the same environment setup, we compare the behavior planner and velocity profiles for the rule-based method and HybridHRL. We set a constant time-to-collision threshold based on the velocity and the relative distance of the target vehicle for the rule-based methods and tune the threshold in order to get a relatively high average reward for a set of test cases. But due to the high density of the traffic in the scene, the time-to-collision threshold is not robust to various

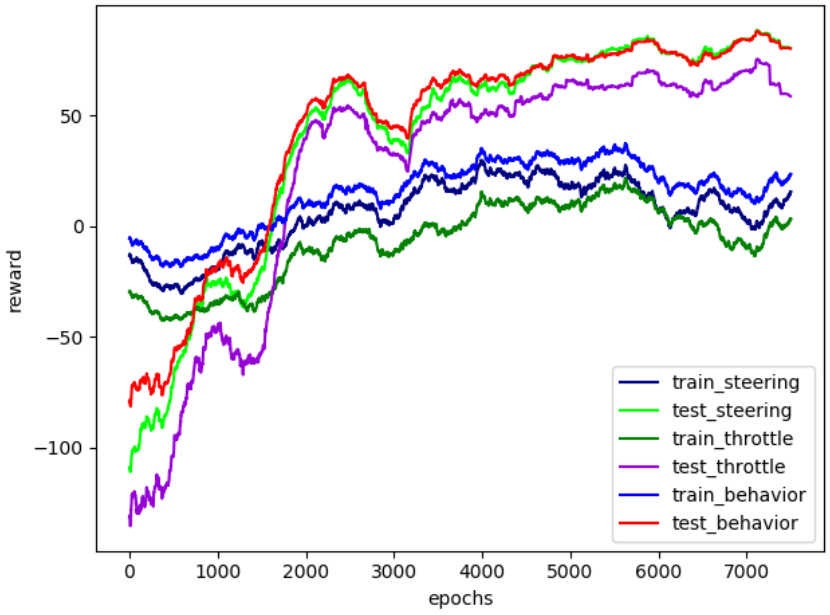

Figure 5.7: Reward for left-turn scenario during training process. The reward is an average performance for 500 cases. The train_X results mean the actions are selected with adjusted heuristic exploration. The test_X results mean the actions are selected directly through the network.

situations. An obvious drawback is that the behavior decision of the rule-based method waits a much longer time than HybridHRL, which can quickly traverse the intersection with fewer steps. Figures 5.9 and 5.10 compare the behavior-level decisions as well as the velocity profiles for the rule-based method and HybridHRL. The positive x-axis direction is the axis of forward advance in the original lane for the ego vehicle, and the y-axis is the direction of the lane into which the car is turning. In the 3D plots, the rule-based method is more likely to choose *WAIT* during the left turn, which makes the ego agent slow down frequently. In the sub-figures on the right, the rule-based ego agent faces more target approaching vehicles than HybridHRL due to its waiting behavior.

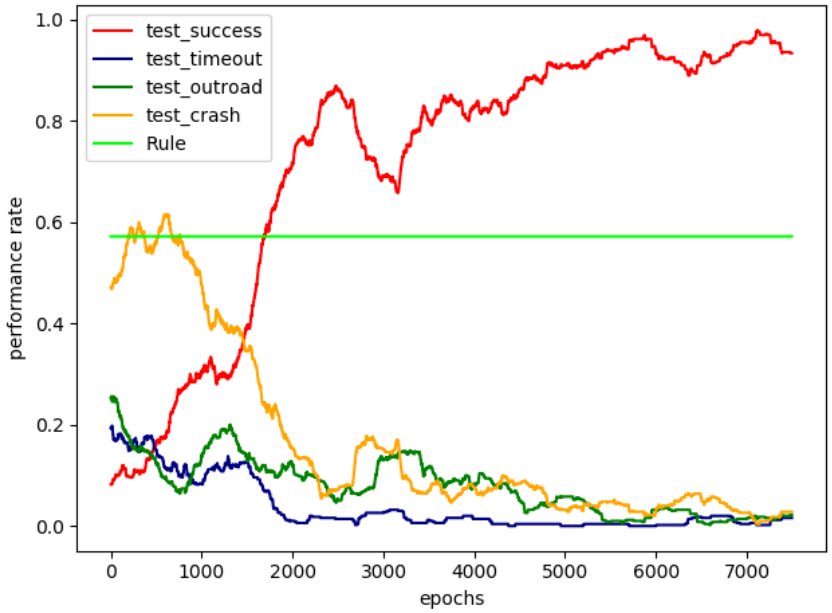

Figure 5.8: Performance rate for left-turn scenario during training process. The rate is an average performance for 500 test cases without explorations of the action space.

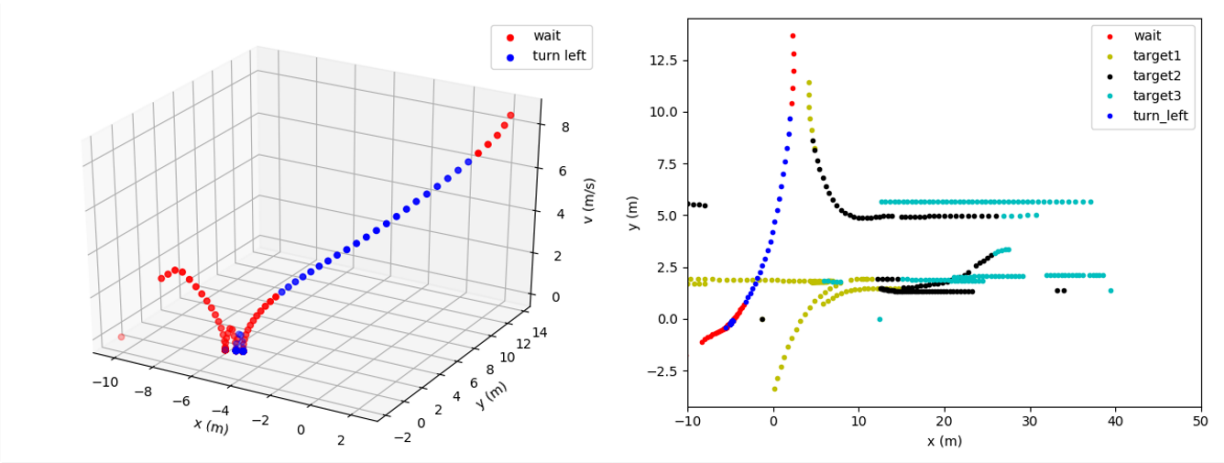

Figure 5.9: Heuristic-based rules-enumeration policy for left turn while encountering approaching vehicles from the opposite direction. The left plot shows only the ego vehicle; the right plot shows all vehicles.

Go-straight scenario

For the two-layer planning system, we visualize the behavior planner in Figure 5.11. For the rule-based algorithm, we can see that the ego car may have the intention to make the lane-change, but

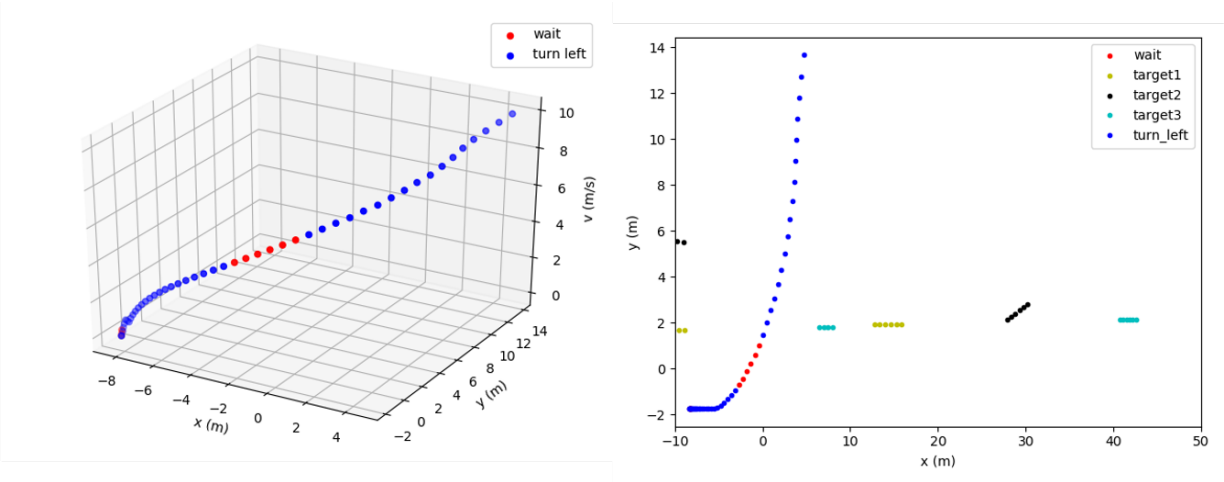

Figure 5.10: HybridHRL for left turn while encountering approaching vehicles from the opposite direction. The left plot shows only the ego vehicle; the right plot shows all vehicles.

not succeed in completing it, as shown in the upper-left subfigure by the initial horizontal blue segment followed by a horizontal red segment. Here we design the intention separately from the action, which means the ego car will also have a lane-change intention if the front vehicle begins to block it. After the lane-change intention is activated, the ego car needs to evaluate the environment to see if the action can be taken safely. However, from the HybridHRL result, we find that the ego car may choose to follow the front car during the major time of the trip. The lane change intention is quickly generated and goes away according to the environment changes which make the higher level decision to be stable.

The steering controller did not always work well in the HybridHRL algorithm, especially for some cases that did not need lane change (see the lower-right subfigure in Figure 5.11). When the ego agent moved forward, it could not stay centered in the lane. As a result, we introduced the three-layer planning system so that the trajectory planner can help to stabilize the controller.

In Table 5.2, we compare the HybridHRL and heuristic-based rules-enumeration policy. For HybridHRL, we tested on both the two-layer and three-layer planning systems. When evaluating the reward for the three-layer planning system, it shares the same reward mechanism with the two-layer planning system.

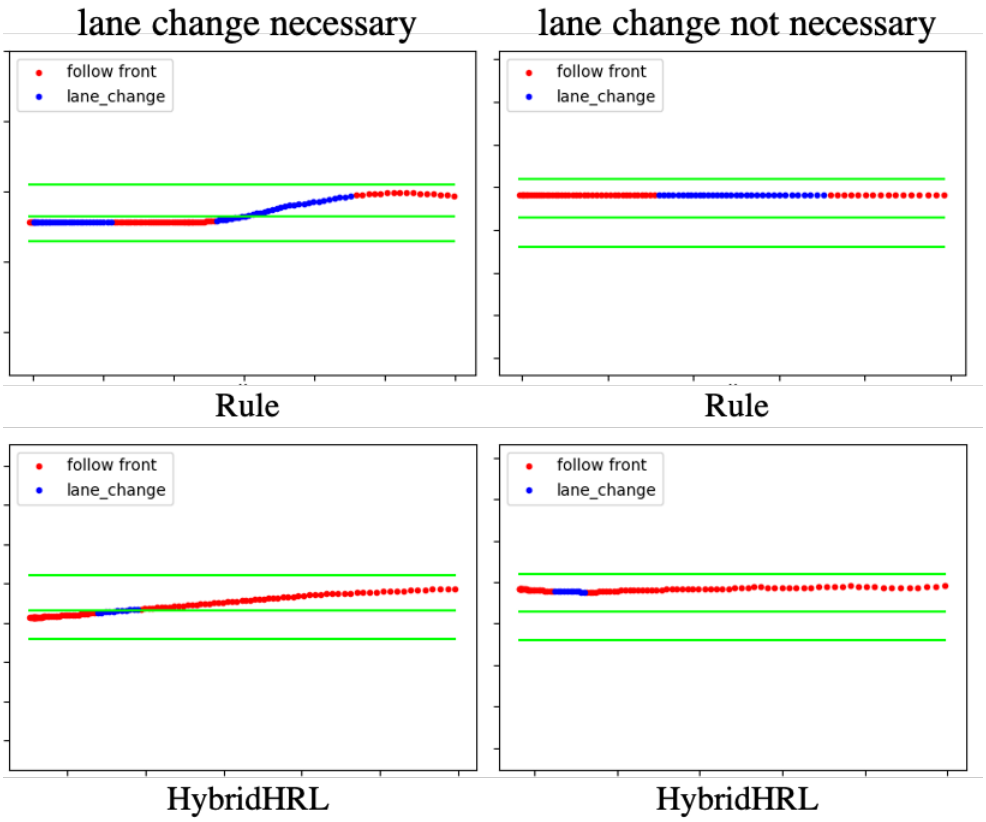

Figure 5.11: Behavior planner visualization for go-straight cases. For some situations when the lane is not blocked, the lane-change behavior is unnecessary. Different-colored dots show the corresponding behavior decisions that are selected.

Table 5.2: Results comparisons between heuristic-based rules-enumeration and HybridHRL for go-straight scenario

	Rewards			Step	Performance Metrics			
	Behavior Reward	Throttle Reward	Steering Reward		Collision	Out of Road	Timeout	Success
Rule	34.33	32.43	40.84	108	19.60%	0.0%	1.2%	79.8%
HybridHRL 2-layer	52.37	47.12	50.26	157	11.6%	3.7%	0.0%	84.7%
HybridHRL 3-layer	82.31	76.32	77.53	157	4.2%	0.2%	0.0%	95.6%

Moreover, we tested the scenario for other reinforcement learning algorithms and visualized the training process in Figure 5.12. For the Double DQN (DDQN) method, we used ϵ-greedy

exploration and no hybrid reward mechanism was applied. For HybridHRL without heuristic exploration, we applied the hybrid reward mechanism technique and the HybridHRL is for a two-layer planning system. The HybridHRL trajectory point is for the three-layer planning system, which also performs the best compared to the other methods. With the help of hybrid reward mechanism and the heuristic-based exploration, the policy network can converge faster than the original DDQN method. Meanwhile, with the help of the PID-controller instead of learning steering angle and throttle directly, the agent can both achieve a higher performance and do it much more quickly than Hybrid-HRL with the two-layer decision structure.

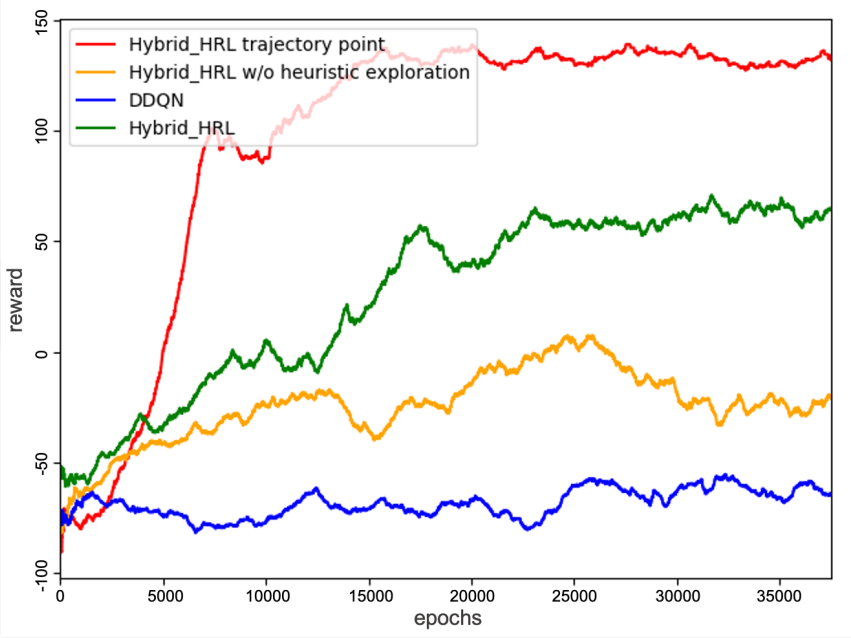

Figure 5.12: Training results of different RL algorithms for go-straight scenario.

In the following Figure 5.13 compare the trajectories generated from both 2-layer and 3-layer HybridHRL structure. With the 3-layer method, we can see that the trajectories yield less to the center of the lane (at the center of the two green lane markers). For 2-layer's structure, sometimes the lane change cannot be performed well when it makes the higher level decision of lane change, but at the lower level, the trajectory are still close to the original lane.

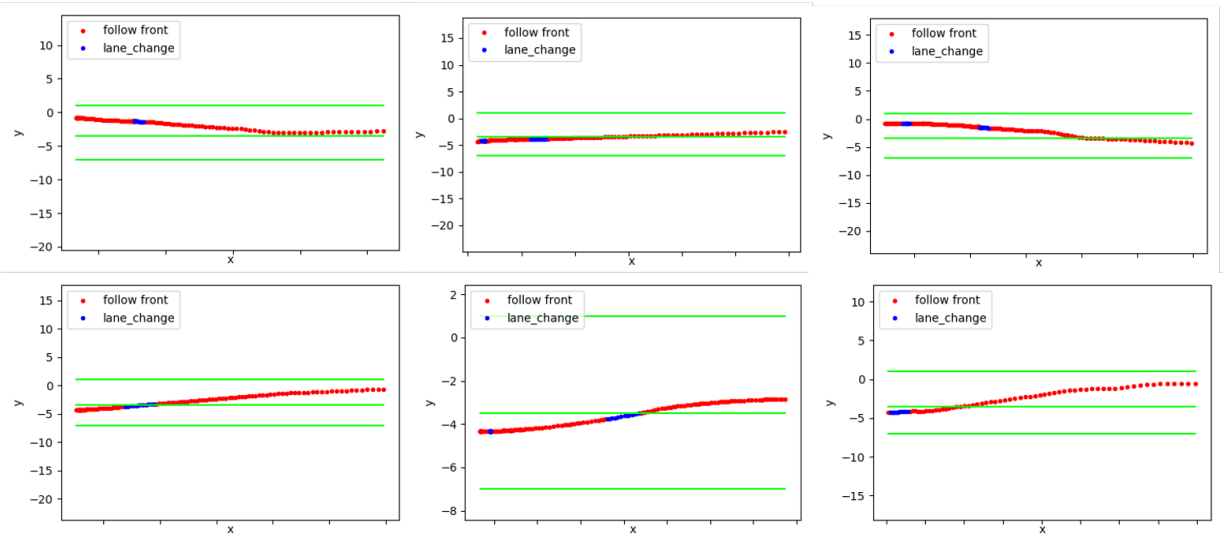

Figure 5.13: Comparison between 2-layer and 3-layer HybridHRL structure. The upper row is the trajectory generated from 2-layer HrybridHRL structure and the lower row is generated from 3-layer system. The green lines are the lane markers.

For both scenarios we include the dynamic results in the video[2]. The video illustrates the planner results for both HybridHRL and rule-based decisions.

5.4 Failure cases

In this section, we compare the HybridHRL and rule-based methods by comparing several failure cases. For various intersection scenarios, we visualize the results for both algorithms and discuss the reasons for failure in each case.

5.4.1 Left-turn Scenario

In Figure 5.14 and Figure 5.15, we visualize the left-turn procedure with different decision making strategies. In this case, the rules-enumeration decision strategy is based exclusively on a constant time-to-collision value. As a result, when the traffic is dense as shown in the figures,

[2] https://youtu.be/Wn3o0PwuVes

it is hard to find a slot for which the time-to-collision for all target vehicles is greater than the threshold, which results in the ego car waiting a long time before turning left. From Figure 5.14 (a) to (b), the ego car moves forward a bit, but many target cars have already passed, and the ego car is still stuck at the waiting to make a left-turn. Even after the ego car makes a decision to turn, a target vehicle coming into the visibility range may cause the ego car to be unable to brake or accelerate in time, leading to a crash. For HybridHRL, due to the flexibility of the decision, which takes more features into consideration, the strategy of when to turn is quick and reliable so that even when new vehicles come into the scene, the algorithm sticks with turning instead of braking to avoid a possible crash under a similar situation. We can see from the figures that the ego car traverses the intersection at the same time as other vehicles waiting at the intersection.

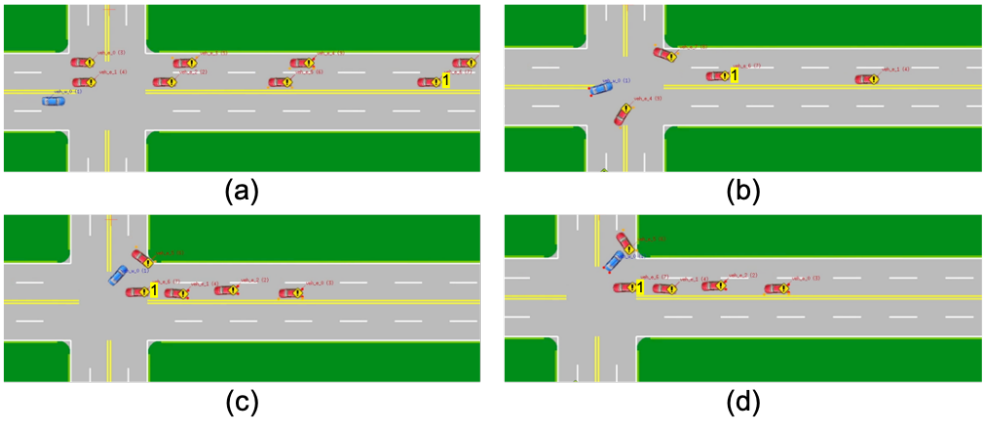

Figure 5.14: Turning left with rules-enumeration: Fail. The blue ego car begins to turn left in the second sub-figure. But the ego car doesn't move forward as quickly as needed because it detects approaching vehicles and the approaching vehicles don't slow down enough to avoid collision.

5.4.2 Go-straight Scenario

For the go-straight scenario, the following figures show a failure case for HybridHRL with the two-layer method and a corresponding successful case for the heuristic-based method. In Figure 5.17, we can see that after the high-level behavior decision to make a lane-change, the ego car

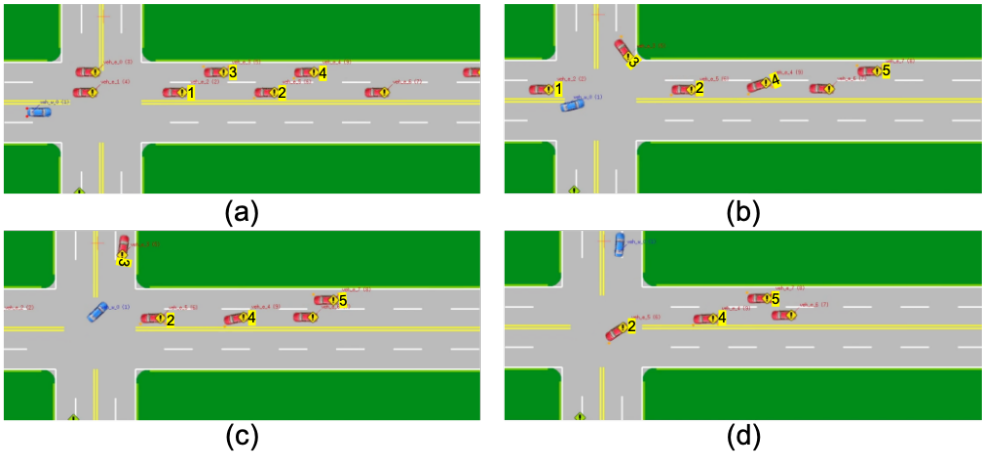

Figure 5.15: Turning left with HybridHRL: Success. Under the same conditions, the blue ego car turns left decisively to pass in front of the approaching vehicles.

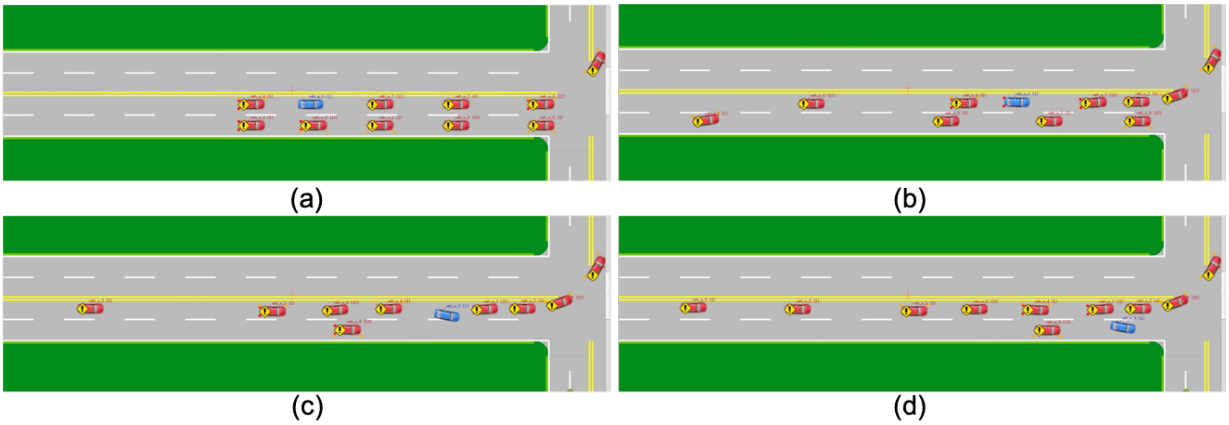

Figure 5.16: Lane change to move forward with Rules-enumeration: Success. In the third subfigure, the blue ego car begins to execute a lane-change.

makes a left lane change successfully. However, while the ego car begins to adjust itself to follow the center of the new lane, it fails to adjust the steering angle and crashes into a car to its left. When the HybridHRL method uses a two-layer hierarchical structure, we use RL to pick a low-level control command, which is more erratic. As a result, we use the three-level hierarchical structure to improve the trajectory planning of the decision making system.

In the following figures 5.16 and 5.19, we can see that the rule-based method is bad at high-

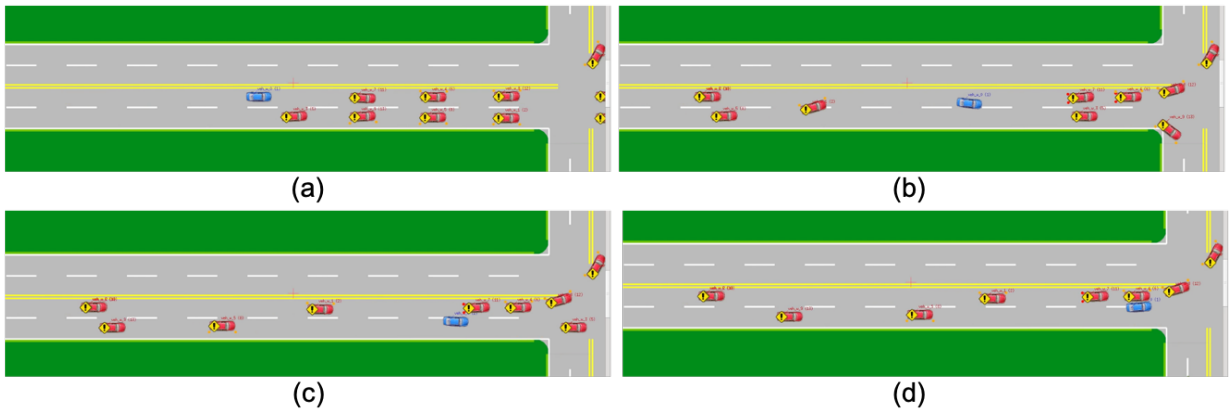

Figure 5.17: Lane change to move forward with HybridHRL: Fail. In the second sub-figure, the blue ego car successfully executes a lane-change into the target lane to get around blocked front vehicles. However, after lane-changing, due to the unstable steering behavior, the blue car deviates from the center of the lane sufficiently that it crashes into a vehicle to its left.

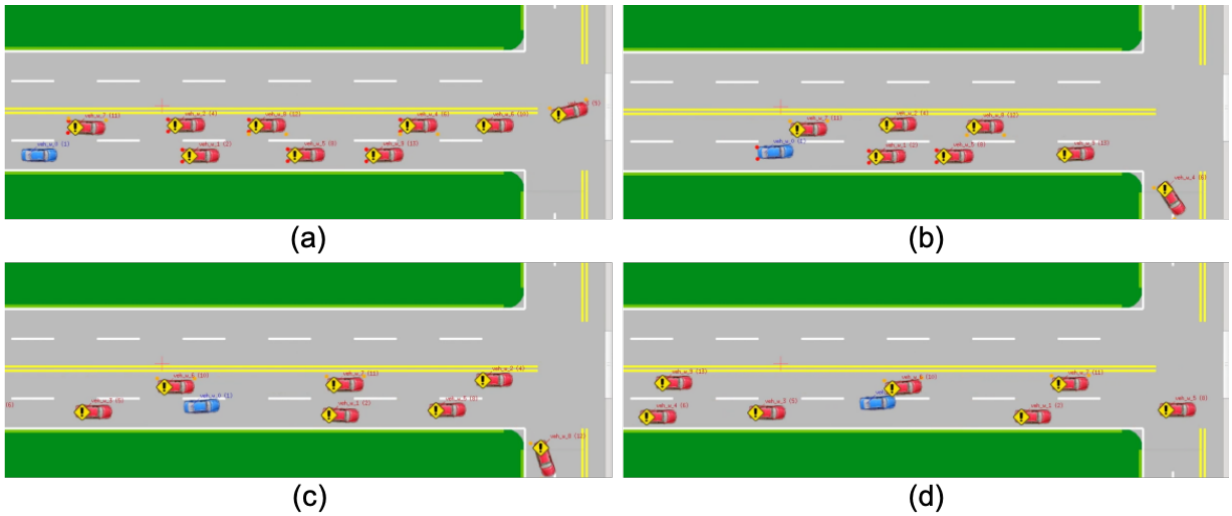

Figure 5.18: Lane change to move forward with Rules-enumeration: Fail.

level decision making and it crashes into the vehicle on the left while making the lane change. In a similar situation, the three-layer HybridHRL method succeeds in making the lane-change as well as maintaining the lane-following decisions at both levels with the help of the PID-controller.

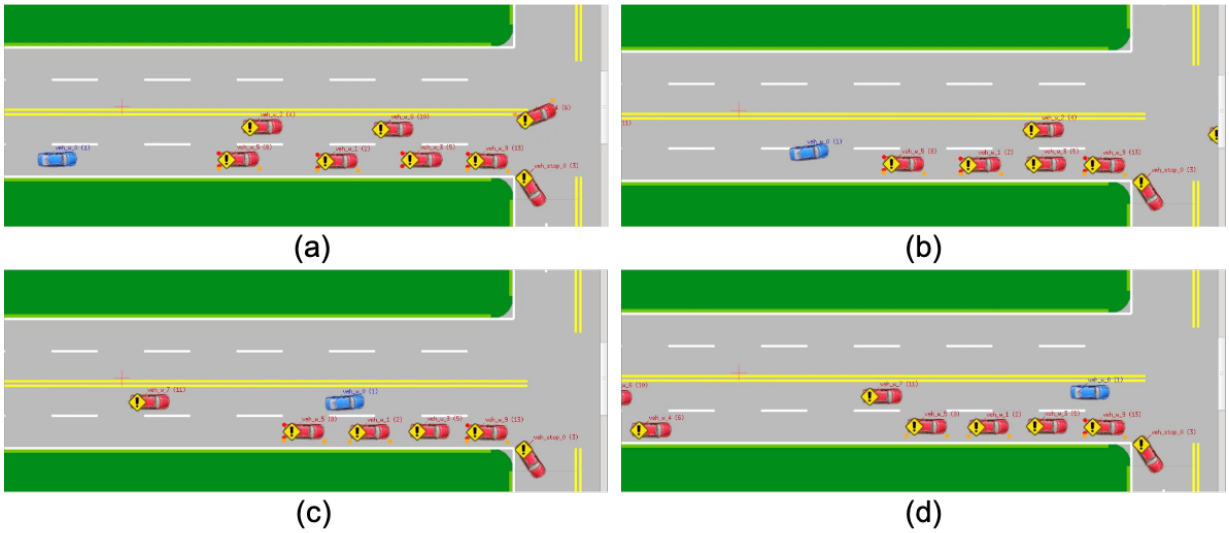

Figure 5.19: Lane change to move forward with HybridHRL: Success.

5.5 Summary

In this chapter, we first introduced our real-world traffic data collection system via drone and then chose two scenarios that require skillful decisions and maneuvering by human drivers. We showed that the rules-enumeration-based method has limited ability to deal with such decision-making problems and that the Hybrid-HRL algorithm can improve the decisions with the help of the approaches we discussed in Section 3.6.

Chapter 6

Conclusions and future work

6.1 Conclusions

In general, heuristic-based rules enumeration methods seem to be a plausible approach to describe the human decision process. Prior distance-based and time-to-collision-based (TTC-based) algorithms include some parameter-tuning to deal with different scenarios. However, tuning these parameters is laborious, since the algorithms are not easily adapted to various environmental situations. They also require the design of a large number of distance-based rules to handle different situations. Recently, as Deep reinforcement learning (DRL)-based approaches have become very popular, the idea of applying DRL to autonomous driving scenarios has gained attention, since it is able to learn successful policies that are comparable to or can even outperform rule-based systems in terms of successfully reaching the goal. Unlike rule-based algorithms, RL can learn to deal with a continuously changing environment by trial and error. Unlike supervised learning, RL does not need a large amount of labeled data to train a data-based model. Rather than learning a mapping from input to label, RL enables an autonomous agent to learn a mapping from environment states to agent actions from its experience, which is similar to how humans learn to drive.

This dissertation applies various RL-based algorithms to improve autonomous driving per-

formance compared to both rule-based methods and existing RL algorithms. We proposed an automatic curriculum generation algorithm for the RL training process that can significantly reduce training time while achieving similar or better performance. We also created an LSTM-based and hierarchical-structure-based policy network which is able to improve the performance of the agent when the ego-car faces limited-field-of-view situations. Based on the hierarchical structure, we finally introduced a decision making system for autonomous vehicles that enables the ego car to cooperate with human-driven cars in urban intersection scenarios, including lane-change, intersection-traverse, and yielding scenarios.

For each algorithm, we conducted experiments by designing various scenarios: single-lane stop-sign intersection approaching, multiple-lane traffic-light intersection approaching, intersections with traffic light control of both left-turn and go-straight vehicles. According to the results gained in simulation from synthetic data as well as from real human-driver data, our decision-making structure enables improved performance compared to both traditional RL methods (DQN/DDQN/DDPG) and heuristic-based methods.

6.2 Future work

Currently, none of the scenarios in our work considers road geometry. We only consider straight roads with a relatively good forward visibility range. Figure 6.1 shows two scenarios in which the road geometry will play a key role when the ego vehicle is making a decision.

For scenario (a), the ego car wants to go straight to traverse the intersection. However, its field of view is blocked due to the geometry of the road. If the ego is blocked by traffic ahead, it may not have a good view to analyze the front and back target vehicles in order to make a higher-level decision on whether a lane-change is safe and needed. Similarly for scenario (b), when the ego vehicle intends to make a left turn at the intersection, it also doesn't has a clear field of view of approaching traffic. When the target vehicles are unobserved in the scenario, the POMDP model can be applied to our hierarchical structure so that the system will be able to

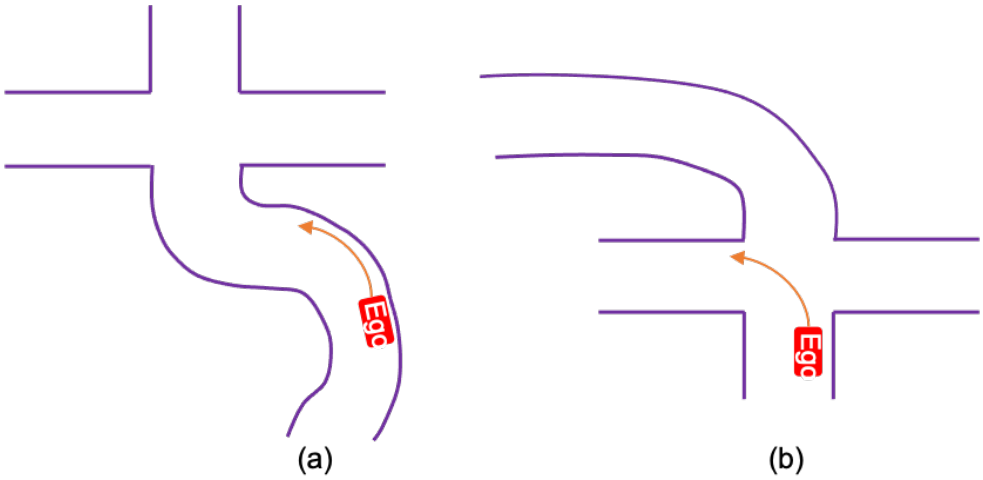

Figure 6.1: Scenarios requiring consideration of road geometry.

handle more complicated scenarios and also predict the surrounding environment. First, we need to transfer the lateral and longitudinal position into a Frenet coordinate system to be compatible with the road geometry. Second, a LSTM structure of the policy network is essential for action output. As discussed in Section 3.5, this kind of policy network will be able to memorize multiple historical observations to produce single- or multi-step actions. Finally, with this kind of road geometry, the hierarchical system needs to have one more layer to determine whether the current field of view is enough to make a high-level decision directly, or, due to the currently perceived conditions, the ego should go for a conservative behavior in order to ensure safety.

Bibliography

[1] Amos Tversky and Daniel Kahneman. Judgment under uncertainty: Heuristics and biases. *Judgment and decision making: An interdisciplinary reader*, pages 38–55, 1986. 1.1, 2.1

[2] Herbert A Simon. Rational decision-making in business organizations. *Economic Sciences (1968-1980). The Sveriges Riksbank (Bank of Sweden) Prize in Economic Sciences in Memory of Alfred Nobel*, 1:343–371, 1992. 2.1

[3] Herbert A Simon. Rational decision making in business organizations. *The American economic review*, 69(4):493–513, 1979. 2.1

[4] Herbert A Simon. Bounded rationality. In *Utility and probability*, pages 15–18. Springer, 1990. 2.1

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

[6] David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976. 2.2.1, 1

[7] Vicente Milanés, Joshué Pérez, Enrique Onieva, and Carlos González. Controller for urban intersections based on wireless communications and fuzzy logic. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):243–248, 2009. 2.2.1

[8] Joshué Pérez Rastelli and Matilde Santos Peñas. Fuzzy logic steering control of au-

tonomous vehicles inside roundabouts. *Applied Soft Computing*, 35:662–669, 2015. 2.2.1

[9] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 2.2.2

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 2.2.2

[11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 2.2.2

[12] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016. 2.2.2

[13] Liting Sun, Cheng Peng, Wei Zhan, and Masayoshi Tomizuka. A fast integrated planning and control framework for autonomous driving via imitation learning. In *Dynamic Systems and Control Conference*, volume 51913, page V003T37A012. American Society of Mechanical Engineers, 2018. 2.2.2

[14] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000. 2.2.2

[15] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA, 2016. 2.2.2

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2.2.3, 2.1, 3.4, 3.4, 4.2.1

[17] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yu-

val Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 2.2.3, 2.1, 3.4, 3.4, 3.5, 4.2.1, 2

[18] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 2.2.3

[19] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015. 2.2.3

[20] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017. 2.2.3

[21] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2.2.3

[22] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019. 2.2.3

[23] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982. 2.2.3, 3.5

[24] Maxim Egorov. Deep reinforcement learning with pomdps, 2015. 2.2.3

[25] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015. 2.2.3

[26] Jakob N Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672*, 2016. 2.2.3

[27] Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. On improving deep reinforcement learning for pomdps. *arXiv preprint arXiv:1804.06309*, 2018. 2.2.3, 3.5

[28] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 392–399. IEEE, 2014. 2.2.3

[29] David Isele, Akansel Cosgun, and Kikuo Fujimura. Analyzing knowledge transfer in deep q-networks for autonomously handling multiple intersections. *arXiv preprint arXiv:1705.01197*, 2017. 2.2.3

[30] David Isele, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating intersections with autonomous vehicles using deep reinforcement learning. *arXiv preprint arXiv:1705.01196*, 2017. 2.2.3

[31] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039. IEEE, 2018. 2.2.3

[32] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4):323–339, 1992. 2.2.3

[33] Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, volume 98, pages 118–126. Citeseer, 1998. 2.2.3

[34] Nicholas K Jong and Peter Stone. Hierarchical model-based reinforcement learning: R-max+ maxq. In *Proceedings of the 25th international conference on Machine learning*, pages 432–439. ACM, 2008. 2.2.3

[35] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–

231, 2002. 2.2.3

[36] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015. 2.2.3, 3.6.2

[37] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *AAAI*, pages 1934–1940, 2016. 2.2.3

[38] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016. 2.2.3, 3.3.4, 3.6.2

[39] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016. 2.1

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 3.3.2

[41] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 3.3.2, 3.6.2

[42] David Isele and Akansel Cosgun. Transferring autonomous driving knowledge on simulated and real intersections. *arXiv preprint arXiv:1712.01106*, 2017. 3.4, 4.3.3

[43] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 3.4, 3.6

[44] Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Maharlevan. Learning hierarchical observable markov decision process models for robot navigation. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 1, pages 511–516. IEEE, 2001. 3.6.2

[45] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018. 3.7.1

[46] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018. 3.7.1

[47] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 5(3&4), 2012. 4.3.1, 4.3.2

[48] VTD homepage., 2019. 4.4.1

[49] Georgios D Evangelidis and Emmanouil Z Psarakis. Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1858–1865, 2008. 5.2.1

[50] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier, 1987. 5.2.1

[51] Stack Overflow. cv2 motion euclidean for the warpmode in ecc image alignment method. 5.2.1

[52] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 5.2.2

[53] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *Computer vision and pattern recognition (CVPR), 2018 IEEE conference on*. IEEE, 2018. 5.2.2

[54] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for

biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 5.2.3

[55] Vladimir Iglovikov and Alexey Shvets. Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation. *arXiv preprint arXiv:1801.05746*, 2018. 5.2.3

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 5.2.3

[57] YT Chan, AGC Hu, and JB Plant. A kalman filter based tracking scheme with input estimation. *IEEE transactions on Aerospace and Electronic Systems*, (2):237–244, 1979. 5.2.4

[58] Zhiqian Qiao, Jing Zhao, Jin Zhu, Zachariah Tyree, Priyantha Mudalige, Jeff Schneider, and John M Dolan. Human driver behavior prediction based on urbanflow. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10570–10576. IEEE, 2020. 5.3.1