Highway Lane change under uncertainty with Deep Reinforcement Learning based motion planner

by

Nazmus Sakib

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Motion Planning is a fundamental component of a mobile robot to reach its goal safely avoiding collision. For a self-driving car on a highway, the presence of non-communicating vehicles, specially those whose intent is unknown, creates a lot of uncertainty for the motion planner in generating a safe trajectory. State-of-the-art planning methods do not work well in case of adversary driving scenarios, where the other vehicles may make mistakes or have a competing or malicious intent. We use reinforcement learning framework to improve safety under those scenarios. In most recent deep reinforcement learning applications, there is a neural network that maps an input state to an optimal policy over actions. However, learning a policy over such original or primitive actions is very slow and inefficient and is therefore not suitable for many robotics tasks. On the other hand, the knowledge already learned in classical planning methods should be inherited and reused. In this thesis, in order to take advantage of reinforcement learning good at exploring the action space for optimal solution and classical planning skill models good at handling most driving scenarios, we propose to learn a policy over an action space of primitive actions augmented with classical planning methods. By doing so, we show that our agent outperforms the primitive-action reinforcement learning agent and the classical planning methods in terms of collision rate.

Preface

The work presented in this thesis is an elaboration of a research article advised by Professor Hong Zhang (Nazmus, Hengshuai Yao, Hong Zhang, 2019) [37]. Dr. Hengshuai Yao participated in the discussion. The author was responsible for implementing the method and performing the experiments.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Hong Zhang for his immense support and guidance. I am really grateful for his valuable advice and supervision throughout the whole journey. He taught me how to define a problem and approach it from different research perspectives. During this period, I made many mistakes but he was very patient, pointed out the right things for me and always encouraged me to improve my skills. He also allowed me to do an internship which was a great opportunity for me to get in touch with industry grade research and valuable experience for my career.

I am also indebted to Dr. Hengshuai Yao for his discussion, support and valuable feedback throughout the internship period. He taught me how to keep trying on a problem until it is solved. I would also like to thank Prof. Dr. Nilanjan Ray, Dr. Ron Kube and Dr. Moein Shakeri for their discussion and suggestions. Many thanks to Ali, Sean, Shing Yan and my other friends and colleagues from Edmonton.

Finally, I am thankful to my family members for their tremendous support and a lot of sacrifices.

Contents

1 Introduction **1**
- 1.1 Self-driving Software Architecture . . . 2
  - 1.1.1 Route Planner . . . 3
  - 1.1.2 Behavioral Planner . . . 3
  - 1.1.3 Motion Planner . . . 4
  - 1.1.4 Control Planner . . . 5
- 1.2 Thesis statement . . . 5
- 1.3 Thesis Contribution . . . 7
- 1.4 Organization of the thesis . . . 9

2 Background Study **10**
- 2.1 Reinforcement Learning . . . 10
- 2.2 Deep Reinforcement Learning (DRL) . . . 12
  - 2.2.1 Discount Factor γ . . . 13
  - 2.2.2 Experience Replay . . . 13
  - 2.2.3 Target Network . . . 14
  - 2.2.4 ϵ greedy . . . 14
  - 2.2.5 Model-based and Model-free . . . 14
  - 2.2.6 On-policy and Off-policy . . . 16
- 2.3 Classical Planning and Reinforcement Learning based planning 16

3 Related Work **18**
- 3.1 Related work . . . 18
- 3.2 Rule-based . . . 18
  - 3.2.1 Sampling . . . 18
  - 3.2.2 Control logic . . . 21
  - 3.2.3 Mathematical Function . . . 23
- 3.3 Learning based . . . 25
  - 3.3.1 Supervised Learning . . . 25
  - 3.3.2 Reinforcement Learning . . . 28

4 Lane changing with DRL motion planner **31**
- 4.1 Problem overview . . . 31
- 4.2 Our methodology . . . 31
  - 4.2.1 2D simulator . . . 31
  - 4.2.2 Experimental Setup . . . 32
  - 4.2.3 Algorithm Setup . . . 36
- 4.3 Experiment and Results . . . 37
- 4.4 Analysis of the learned Q values . . . 41

5 Conclusion **43**

References **45**

List of Tables

4.1 *The adversary lane changing task: Performance of our method, primitive action reinforcement learning, human and three planning methods. "Ours-P" is the our method with P being the additional classical planner besides primitive actions.* 37

List of Figures

1.1 Prominent self-driving platforms, from left to right and top to bottom: the task of lane change, Stanley robot for DARPA 2005 challenge, Google's Waymo self-driving car ride sharing, Uber's self driving car fleet. 2
1.2 Hierarchy of decision-making process, figure from [29] 4
1.3 Successful moments of driving with our method: merging (row 1), passing (row 2) and finding gaps (row 3). 8

2.1 Reinforcement Learning framework 11
2.2 Difference between Q-learning and Deep Q Learning 13
2.3 Deep Q Learning pseudo code from [23] 15

3.1 Different approaches to solve CA, the leaves of the tree contain few examples of the subdivision. 19
3.2 Trajectory generation for different speed values, image from [17]. 20
3.3 Motion plans are propagated using the vehicle's dynamical model. Propagated paths are then evaluated for feasibility, image from [15]. 21
3.4 The control architecture of the autonomous vehicle system [14]. 22
3.5 The membership function definition for the input fuzzy variables. (a) Straight-path-tracking error. (b) Straight-path-tracking angular error. (c) Lane-change lateral error. (d) Lane-change angular error, image from [26]. 23
3.6 (a) Occupancy grid obtained from Gaussian function H, (b) discretizing road spaces, (c) using Dijkstra to calculate the shortest path, image from [32]. 26
3.7 (a) E2E learning architecture by Nvidia, image from [7]. . . . 27
3.8 Network architecture, simulator and occupancy grid of [25]. . 28
3.9 Network architecture of [21]. 29

4.1 Hierarchy of the action selection by the learned policy 33
4.2 Learning curve (Collision): our method (an RL agent with primitive actions and augmented skill actions) vs. primitive actions RL agent. 37
4.3 Learning curve(Reward): our method (an RL agent with primitive actions and augmented skill actions) vs. primitive actions RL agent. 38
4.4 The system used for collecting driving performance data from human testers: A logitech driving wheel, acceleration and braking paddles, and a chair. 39
4.5 Sampled moments: Q values for different actions. 41
4.6 $Q(s, a = switch\_right)$ at a number of successive moments . . 42

Chapter 1

Introduction

A self-driving car is nothing but a mobile robot equipped with sensors to perceive its surroundings and proper planning algorithm to reach a destination. Although humans are good drivers but they are prone to tiredness, fatigue and distraction which lead to unwanted accidents. As a result, safety has been the utmost concern in developing an autonomous driving system for both academia and commercial companies. It is believed that sensors will be able to see beyond human capabilities, a good perception system will be able to understand/predict the environment and a good planner will result in collision-free motion. Autonomous vehicles are assumed to be operable both in urban and highways. Changing lane is one of the most important tasks of a self-driving car navigating on the highway. This feature adds to the comfort driving for long-distance travel. Although the ultimate goal is to improve safety and make a consumer-grade system to be used by mass people.

Autonomous driving has drawn much attention in the last decade, but addressing the problem is not new. One of the earliest successful demonstrations of the self-driving car goes back to 1989. At that time, the project ALVINN [33] used camera image and laser data as input to a neural network trained with simulated road images to navigate a vehicle autonomously. 2005 DARPA challenge inspired a lot of researchers/engineers to reuse the robotics knowledge learned so far into a sensor fused vehicle to solve the task of 212 km driverless drive in a desert terrain. The champion team Stanley [42] is a great example of the use of machine learning and probabilistic reasoning. 2007 DARPA chal-

Figure 1.1: Prominent self-driving platforms, from left to right and top to bottom: the task of lane change, Stanley robot for DARPA 2005 challenge, Google's Waymo self-driving car ride sharing, Uber's self driving car fleet.

lenge required autonomous vehicle in city traffic conditions in the presence of dynamic vehicles. Followed by these events many car companies and university labs started developing fully autonomous vehicle systems. Google-owned self-driving car company Waymo has the highest driverless drive of more than one million miles with no major accidents. Now almost all the motor companies have self-driving division. GM's cruise, Ford's Argo, Uber, Tesla are the most prominent ones. Of them, Uber and Tesla have encountered accidents and caused the death of people which lead us to focus on more tested safety features before mass production of these vehicles.

1.1 Self-driving Software Architecture

The planners for the self-driving car can be divided into four hierarchical parts [29] as shown in Figure 1.2. At the upper level, there is the **Route Planner** responsible for navigating the car from point A to point B. The middle-level one is the **Behavioral Planner** which acts locally, decides driving behavior obeying the traffic rules, maintaining speed limits, etc. The next

planner is called, the **Motion Planner** responsible for collision-free trajectory in order to follow the previous two planners towards reaching a goal. The **Control Planner** is the last planner which is responsible for the actual driving of the car with low-level controls such as steering control and speed control.

1.1.1 Route Planner

Usually, there is a map with the route planner. One such example is the use of google map in our day to day driving and deciding which roads to take. It is the highest level of decision-making system and also known as the **Mission planner**. Given the destination from the source, it generates the path among the road network. One basic intuition is to get to the goal destination in the shortest path. Dijkstra [10] and A* [28] are two classical planners for global route planning. State-of-the-art planners take many facts during generating paths, such as distance, the average time to reach, traffic congestion, traffic lights, road speed limit, road availability, etc.

1.1.2 Behavioral Planner

The behavioral planner deals with the rules of the traffic. The ego car's behavior will certainly be different based on whether it is on the highway or city or parking lot. The driving norms such as stopping at a red light, giving right of way to other vehicles, turning on signals before switching lane, etc. are defined in this planner. A large part of the planner depends on extracting behaviors of other vehicles, pedestrian and their future intention. Many teams in DARPA Urban Challenge [20] used finite state machines along with different heuristics rules to design this type of planner. Some planners use Markov Decision Process (MDP) for modeling uncertainties in the behavior of other traffics [36] [43].

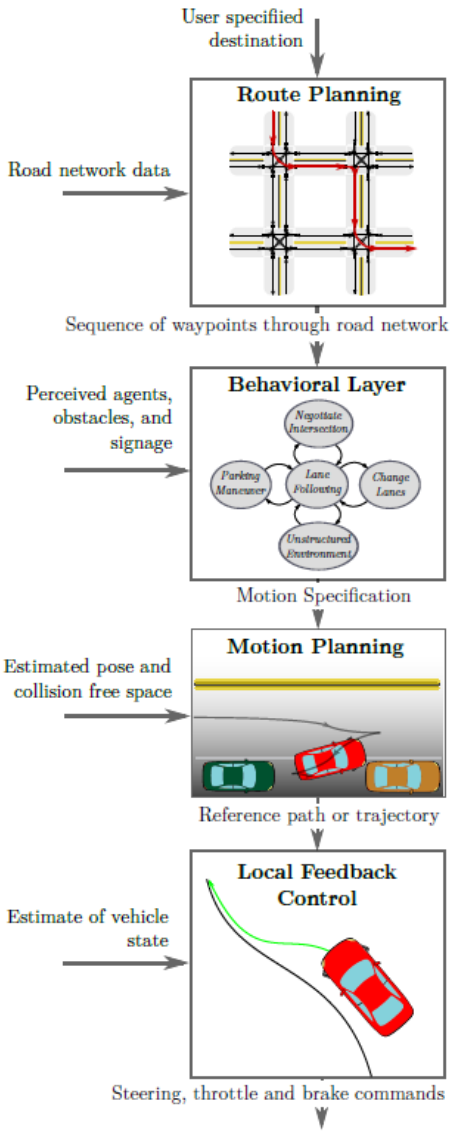


Figure 1.2: Hierarchy of decision-making process, figure from [29]

1.1.3 Motion Planner

Motion planner mostly deals with **collision avoidance** and generating a smooth trajectory for comfortable driving. It is always consistent with the previous two planners. Many motion planners output a trajectory that is passed to the low-level controller to follow. The motion planner takes input from the map or sensor values to locate obstacles and define free space. There are many approaches to design this type of planner. One approach is a graph-based where the dividable spaces are defined as nodes in a connected graph.

The trajectory is obtained by solving the graph for the shortest path. It often requires smoothing to get a good trajectory. Dijkstra[10], A*[28], potential field [11] are a few examples of graph-based solutions. Another approach is sample based [19], where derivable spaces are discretized as points, sampling is done on those points until a feasible trajectory is found. Planners are also designed with learning-based approaches. In the supervised type of learning [35], the behavior of an expert driver is learned. In the reinforcement type of learning [21], the planner learns the behavior by interacting with the operating environment. Other approaches to design planners include breaking the planner's task into a set of rules and following them based on some expert knowledge. For more elaborate discussion see 3.1.

1.1.4 Control Planner

The control planner is responsible for the actual movement of the robot. For an autonomous vehicle, speed control, steering control, braking, etc., are low-level actuation. If the motion planner generates a trajectory, the control planner calculates the error in following the trajectory and tries to adjust it in a closed-loop feedback manner. PID[9] and MPC[14] are two control algorithms popularly used in a self-driving car. Tasks like adaptive cruise control, lane follow and emergency brake are done with these types of controllers.

Our solution is a deep reinforcement learning-based motion planner for lane changing in the highway. We assume that we have a good low-level controller for speed control and steer towards the next lane. Here we are basically taking high-level decisions to decide when to stay in lane or switch to the next one. As a result, it is compatible with a global planner and behavioral planner.

1.2 Thesis statement

In this thesis, we design a reinforcement learning(RL) based motion planner for the task of highway lane changing. Our planner also uses a classical planner in the RL framework. The planner operates in adversarial driving scenarios. By adversarial we mean, the other vehicles in the environment can make mis-

takes or have a competing or malicious intent, show inappropriate lane change or sudden stopping behavior which might lead to accidents. Although human drivers drive well in normal traffic but they are not good at handling accidents because a human driver rarely experiences accidents in one's life regardless of a large amount of accident-free driving time. On the other hand, most state-of-the-art planning algorithms for autonomous driving do not consider these adversaries, usually assuming all the agents in the environment are cooperative. For example in [32] the other vehicles are assumed to be "self-preserving", i.e, they are giving brake and following the norm of highway driving. Another example is Optimal Reciprocal Collision Avoidance (ORCA)[5], a popular navigation framework in crowd simulation and multi-agent system for avoiding collision with other moving agents and obstacles. It works with guaranteed collision-free motion when all the agents follow the same protocol. There is always room for improving these planners by incorporating the rules associating with the cases they fail to address. Under these circumstances, the question for us to explore in the long term is, can we improve safety in terms of predicting collision and avoiding it?

Another approach is supervised learning where the system will try to mimic good human driving behavior given a lot of positive and negative samples but it suffers from bounded performance due to imbalanced samples. We believe that using programs to simulate billions of accidents in various driving scenarios will provide necessary training samples. With such large scale accident simulation, we can take advantage of reinforcement learning RL framework to design a controller that would learn new rules automatically and encapsulate in a function systematically in contrast to an ad hoc way of incorporating case-specific rules into the classical controllers.

Driving can be formulated under a reinforcement learning framework. It has a clear temporal nature e.g., *the current action has an effect on choosing the actions in the future*. An action to take at every time step influences the resulting state which the agent observes next, which is the key feature of many problems where reinforcement learning has been successfully applied. The reasoning which action to apply by considering its long-term effects is usually

called temporal credit assignment, which is usually modeled as a reinforcement learning problem.

In most recent reinforcement learning applications, there is a deep neural network that maps an input state to an optimal policy over primitive actions(original). However, learning a policy over primitive actions is very difficult and inefficient. For example, hundreds of millions of frames of interacting with the environment are required in order to learn a good policy even for a simple 2D game in Atari 2600. On the other hand, the autonomous driving field has already practiced a rich set of classical planning methods. It is worth pointing out that the problem of state-of-the-art planning is not that their intended performance is bad. In fact, both research and industrial applications have shown that classical planning works great in the scenarios they are developed for. As a result, the knowledge already learned in state-of-the-art planning methods should be inherited and reused. To take advantage of both methods, we propose to learn a policy over an augmented action space from both primitive actions and classical planning methods. Classical planning methods are treated as skills and reused. They can be called with an input state and give an action suggestion.

Our work opens the door to an effective architecture solution for autonomous lane changing: building a decision hierarchy of skills using classical planning or learning-based methods, and calling them as augmented actions by reinforcement learning.

1.3 Thesis Contribution

Our proposed reinforcement learning planner will be able to select over the action suggestions by classical planning methods as well as the primitive actions. It is able to call classical planning methods to apply the skills in normal conditions for which they are developed, but is also able to pick the best primitive action to avoid the collision in scenarios where classical planning cannot ensure safety. In this way, we do not have to re-learn for the majority of scenarios in driving where classical planning methods already can deal with, saving

lots of time for training the deep networks, and focus on the rare but most challenging scenarios they are not designed for. The advantage of our method is that we do not have to manually detect whether classical planning fails or not instead, failures of their actions are propagated by reinforcement learning to earlier time steps and remembered through neural networks in training to avoid selecting classical planning on similar failure cases in the future.

We compare our method with some rule-based planners and also primitive action RL planner. Our method is able to achieve the lowest collision rate. It also learns behaviors like merging, passing or fitting in a gap, see figure 1.3 .

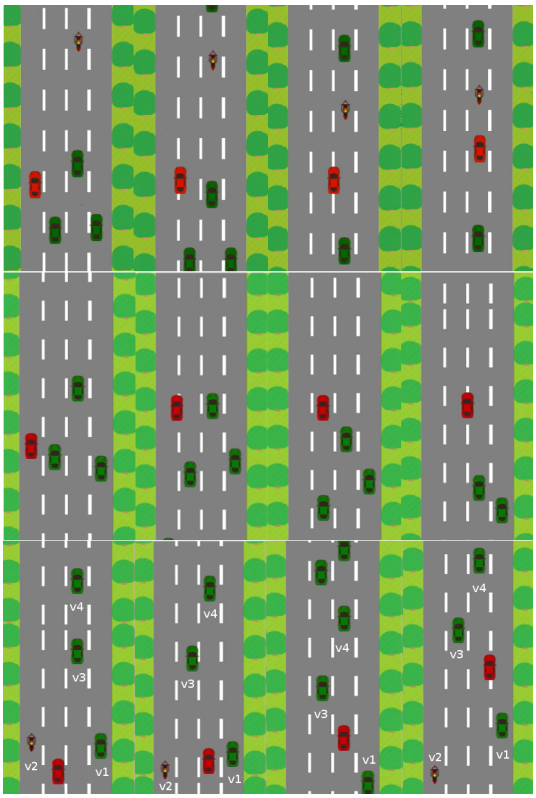


Figure 1.3: Successful moments of driving with our method: merging (row 1), passing (row 2) and finding gaps (row 3).

1.4 Organization of the thesis

Chapter 2 discusses the basics of reinforcement learning, deep reinforcement learning and how classical planners are related to DRL based planners. Chapter 3 describes different approaches to collision avoidance in the highway and designing different motion planners. Our proposed method is described in Chapter 4. Here we talk about the simulator we used and the whole experimental setup and discuss the results. Chapter 5 summarizes the proposed solution and points the future work.

Chapter 2

Background Study

Our solution consists of a combination of classic motion planner and a reinforcement learning (rl) based motion planner. In this chapter, we review the necessary reinforcement learning background, formulation deep reinforcement learning and how it can be used in the design of a motion planner. We also define classical planning and how both rl planner and classical planner are related.

2.1 Reinforcement Learning

Reinforcement Learning is a kind of machine learning method that learns by interacting with the environment. Usually, the learning agent takes an action in the environment it operates, observes the environment as state and receives a reward. The agent updates its knowledge based on the state-reward pair and takes a new action and the cycle goes on until the environment sends a terminal signal to end the episode. The rewards are either positive, negative or neutral based on the goal the agent is trying to achieve. A general intuition is to give high positive reward on success and penalize(high negative reward) on failure. Sometimes the rewards are also given to encourage some behavior like solving in the shortest time or avoid some obvious facts.

One example of an environment can be a Pong Atari game. The actions of the agents are up and down to slide the bar and hit the ball such that the opponent fails to hit back. We can call these two original actions as primitive actions. The reward can be 0 at each step of playing and +1 for a miss from

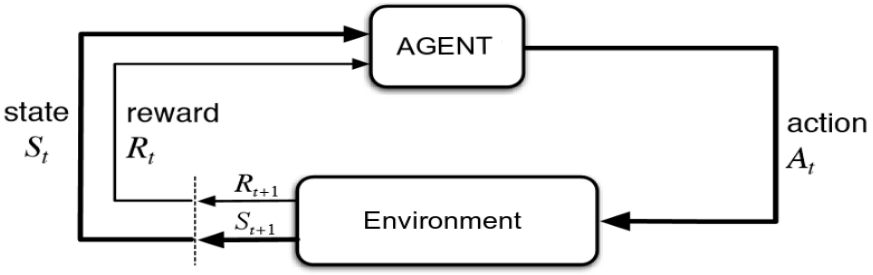


Figure 2.1: Reinforcement Learning framework

the opponent or -1 if the learning agent misses. The state can be the image of the game state or a tuple of the positions of the opponent's bar, agent's bar and the position of the ball. The goal of the agent is the maximize the reward and eventually win.

The RL framework is formulated as a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward "function" $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a transition kernel $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and a discount ratio $\gamma \in [0, 1)$. The reward "function" R is generally a random variable or constant variable. The bandit setting is a special case of the general RL setting, where we usually have only one state.

We use $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ to denote a stochastic policy. We use $Z^\pi(s, a)$ to denote the random variable of the sum of the discounted rewards in the future, following the policy π and starting from the state s and the action a. We have $Z^\pi(s, a) \doteq \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)$, where $S_0 = s, A_0 = a$ and $S_{t+1} \sim p(\cdot|S_t, A_t), A_t \sim \pi(\cdot|S_t)$. The expectation of the random variable $Z^\pi(s, a)$ is

$$Q^\pi(s, a) \doteq \mathbb{E}_{\pi,p,R}[Z^\pi(s, a)]$$

which is usually called the state-action value function. In general RL setting, we are usually interested in finding an optimal policy π^*, such that $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ holds for any (π, s, a). Thus RL is to solve for each state an optimal policy which achieves maximum rewards. All the possible optimal policies share the same optimal state-action value function Q^*, which is the unique fixed point of the Bellman optimality operator [4] as follows,

$$Q(s, a) = \mathcal{T}Q(s, a) \doteq \mathbb{E}[R(s, a)] + \gamma \mathbb{E}_{s' \sim p}[\max_{a'} Q(s', a')]$$

Based on the Bellman optimality operator, [45] proposed Q-learning to learn the optimal state-action value function Q^* for control. At each time step, we update $Q(s, a)$ as

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2.1}$$

where α is a step size and (s, a, r, s') is a transition. This is usually referred to *tabular* Q-learning. Tabular Q-learning uses no function approximation and does not have generalization and usually is used in small problems.

2.2 Deep Reinforcement Learning (DRL)

There have been many works extending Q-learning to linear function approximation [39] [40] where the Q values are stored in a table for all possible state and action pair but in reality, this does not scale to complex problems. Because the amount of memory required to save and update the table increases as the number of states increases. Secondly, the amount of time required to explore each state to create the required Q-table would be unrealistic. The figure 2.2 shows difference between them. [23] combined Q-learning with deep neural network function approximators, resulting in the Deep-Q-Network (DQN). Here a neural network is used to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output.

DQN relies on the use of experience replay buffer. At each time step, it samples a minibatch from a memory buffer of past experience. Stochastic gradient descent is applied to the mini batch for training. Because RL has no training targets provided like the typical supervised learning setting, a trick of target networks is used to generate training target from a historical snapshot of the networks. Assume the Q function is represented by a network θ, at each time step, DQN performs a stochastic gradient descent to update θ minimizing the loss

$$(r_{t+1} + \gamma \max_{a} Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t))^2$$

where θ^- is target network [23], which is a copy of θ and is synchronized with θ periodically, and $(s_t, a_t, r_{t+1}, s_{t+1})$ is a transition sampled from a experience replay buffer [23], which is a first-in-first-out queue storing previously experienced transitions. See figure 2.3 for the pseudo code.

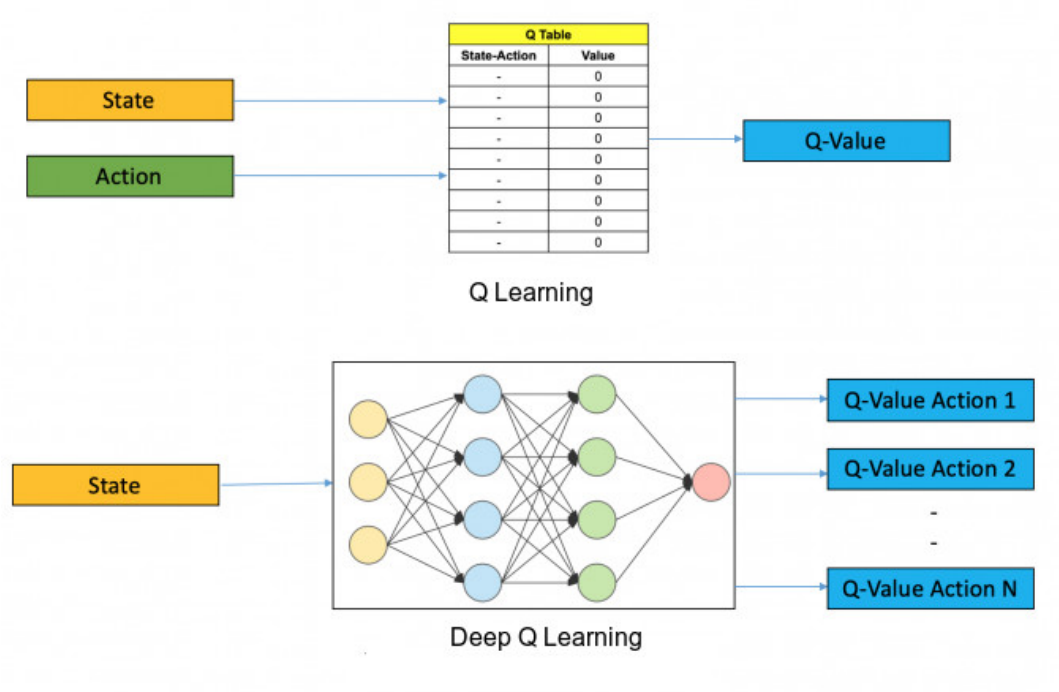


Figure 2.2: Difference between Q-learning and Deep Q Learning

2.2.1 Discount Factor γ

The discount factor *gamma* determines how much importance the rl agent gives to the future rewards in the value function. A discount factor $\gamma = 0$ helps the agent learn about immediate reward. On the other hand, for $\gamma = 1$, the agent cares about the sum of all the future rewards.

2.2.2 Experience Replay

Each experience contains current state, current action, reward, next state and a boolean variable about whether the transition is terminal or not. They are stored in a buffer of fixed size. Then a sample of batches is taken randomly during the update in a learning step. Since the data is highly correlated the

experience replay buffer helps to learn in a reasonable way. The experience replay helps to prevent the network from learning from what it is doing immediately. Usually, each new experience is pushed at the end and if the buffer is full, the oldest one is removed.

2.2.3 Target Network

In order to generate target Q values, the copy of the learned network is saved as a second network after a fixed number of learning steps. This network is used to compute the loss for every action during training. If the same network is used every time, the network values shift and it is hard to stabilize. $r_{t+1} + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)$ part of the above mentioned loss function is called the target part.

2.2.4 ϵ greedy

In the DQN algorithm, at each time step of training, the agent picks a random action with probability ϵ or action from the current estimate of the Q-values with probability 1 - ϵ. The random search for good action is called the exploration phase while using the current estimates of Q-values is called exploitation. The value of ϵ is usually decreased over time as the agent gradually learns the task.

2.2.5 Model-based and Model-free

The goal of the RL agent is to maximize the reward. In order to do so, it needs to find a strategy or a policy that collects more rewards in the long run. The policy can be treated as a function that outputs an optimal action given the state as input. The environment is **deterministic** if the same action for the same state always leads to the same next state, e.g., in the game of chess, moving a dice always leads to a fixed next game state. On the other hand, the environment is **stochastic** or **non-deterministic** if the next state is different for the same set of state and action i.e, there are some probabilities in the transition. For instance, a mobile robot taking a movement action will not

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Figure 2.3: Deep Q Learning pseudo code from [23]

always lead to the same next position due to uncertainties in the motion.

The MDP mentioned earlier can be treated as a representation of the "dynamics" of the environment. Given any state, it defines how the environment will react to the possible actions the agent is allowed to take. The MPD is coupled with the transition function such that given current state and action, it outputs the probability of moving to the next states. The transition function and reward function together are called the model of the environment.

Model-based algorithms use a model to estimate the optimal policy. That means the agent has the capability to predict the dynamics of the environment since it is using the transition and reward functions.

Model-free algorithms estimate the optimal policy without using the dynamics of the environment. This type of algorithm estimates a "value function" or "policy" directly from the agent's interaction with the environment without using transition function or reward function. A value function evaluates a

state or an action taken in a state.

The highway road environment we are considering in this thesis is **stochastic** and the RL algorithm we use is **model-free** which tries to learn the uncertainties of the other agent's driving behaviors.

2.2.6 On-policy and Off-policy

An on-policy RL agent learns the value function based on its current action which is derived from the current policy. The algorithm SARSA[39] is an example of an on-policy algorithm. The update function of SARSA is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

Here the action a' is taken according to policy π. In case of off-policy, the action a' is derived from another policy. For instance, the Q learning update rule uses the action which will yield the highest Q-Value (see equation 2.1) which is different from the current policy π.

On-policy versus off-policy can be simply understood whether the action reuses the same action that is optimized in the last state. Experience replay is a typical off-policy learning problem we iteratively solve the policy at state s' and we never follow the same action at s' (because this is an off-line experience the action at s' was already determined when we collect data). Epsilon-greedy is also off-policy because the action to follow in the next step may be a random action not from the optimized action at the previous state.

2.3 Classical Planning and Reinforcement Learning based planning

By classical planning, we refer to the traditional way of designing a planner for robotics. The term has been used to differentiate between the rule-based and learning-based methods which are comparatively newer approaches. There are a few common fundamental principles in the core ideas of classical planning and reinforcement learning for designing a planner.

First, the temporal relationship between the actions selected at successive time steps is considered in both the fields. Optimizing the cost over future time steps is the key idea commonly shared between classical planning and reinforcement learning algorithms. For example, in MPC classical planner, there is a cost function defined over a time horizon for the next few actions. The cost function can be assumed as one special case of the (negative) reward function in reinforcement learning. MPC relies on a system model and an optimization procedure to plan the next few optimal actions. The collision avoidance algorithm using risk level sets [32] maps the cost of congestion to a weighed graph along a planning horizon and apply Djikstra's Algorithm to find the fastest route through traffic [32]. Many collision avoidance planning algorithms evaluate the safety of the future trajectories of the vehicle by predicting the future motion of all traffic participants, e.g., see [17]. However, MPC, Djikstra's Algorithm and collision avoidance planning are not sampled based, while reinforcement learning algorithms are sample-based.

Second, both fields tend to rely on decision hierarchies for handling complex decision making. Arranging the software in terms of high-level planning, including route planning and behavior planning, and low-level control, including motion planning and closed-loop feedback control, became a standard for the autonomous driving field [44] [24] [38]. In reinforcement learning, low-level options and a high-level policy over options are separately learned [3]. In robotics, locomotion skills are learned at a fast time scale while a meta policy of selecting skills is learned at a slow time scale [31].

Third, sampling-based tree search methods exist in both fields. For example, RRT is a motion planning algorithm for finding a safe trajectory by unrolling a simulation of the underlying system [15]. In reinforcement learning, Monte-Carlo Tree Search (MCTS) runs multiple simulation paths from a node to evaluate the goodness of the node until the end of each game.

Chapter 3

Related Work

3.1 Related work

The key point of a motion planner is to avoid collision locally. The problem of Collision Avoidance (CA) has seen a wide range of solution approaches. Hence in this chapter, we talk about different collision avoidance solutions from the perspective of mobile robot applications such as lane change, indoor or outdoor navigation and crowd navigation. We can divide the solutions into two major categories based on the use of data and expert knowledge. The expert knowledge in the domain can be defined as **rule-based** solutions. On the other hand, a **learning based** system can be trained with data coming from a robot's locomotion or sensor. The rule-based approaches can be subdivided into sampling, control logic and mathematical function-based methods. The learning-based methods can be again subdivided into reinforcement and supervised learning. See the tree structure in Figure 3.1.

3.2 Rule-based

3.2.1 Sampling

In the sampling-based methods usually, multiple trajectories are generated and the best one is chosen based on some predefined safety measure. The interactive scene prediction [17] method generates future trajectories by predicting the future motion of all the mobile vehicles. Then they compute the collision probability of each of the trajectories. The intention is predicted though

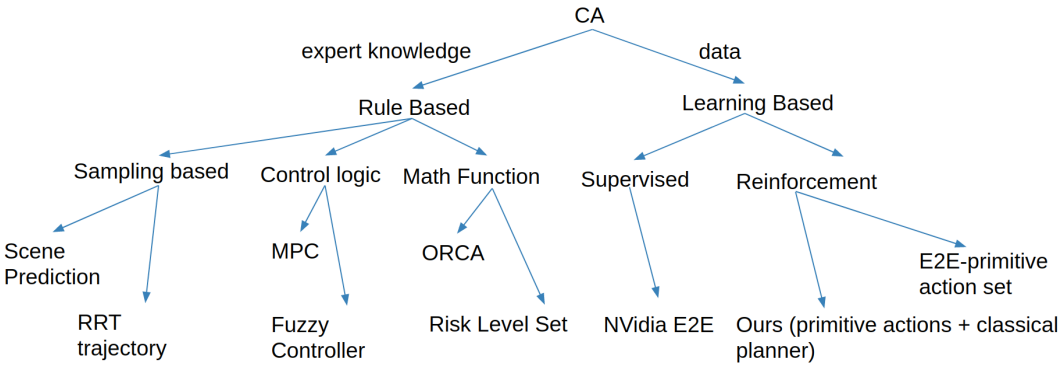


Figure 3.1: Different approaches to solve CA, the leaves of the tree contain few examples of the subdivision.

the trajectories. Since the number of such trajectories can be infinite. They discretize the continuous movement space with the probable future position of the agents by considering their motion dynamics, e.g., a car can not slide to right or left or jump to the far lanes. They denote this space as "circle of force" see Figure 3.2. Based on acceleration, braking and constant speed different levels of trajectories can be generated and corresponding intention is estimated. The collision probabilities are used for threat assessment. They propose that they can estimate the interaction aware maneuver probabilities from intention estimation and collision probabilities for the driver assistance system.

MIT's team [15] in the DARPA Urban Challenge, used a Rapidly Exploring Random Trees (RRT) [16] based planner to drive in the city. The RRT algorithm is used for planning in the high dimensional space. It builds tree incrementally from a sample drawn randomly from the search space and grows towards the unsearched portion. The constraints such as obstacle avoidance, vehicle dynamics can easily be integrated in generating the space-filling tree. In the proposed method, they assume a low-level controller is able to follow a trajectory, see Figure 3.3. Therefore they select a node (a point on the road), expand, check constraints up to some fixed time stamps. The best trajectory is selected based on the constraints and sent to the low-level controller. If no feasible trajectory is found the car starts braking. In order to address the un-

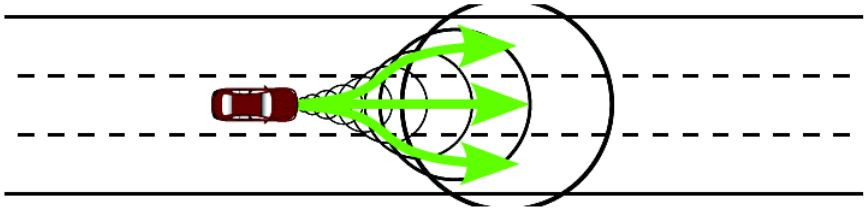


Fig. 2: The possible future positions of this vehicle on a highway are only restricted by the dynamic constraints. The black circles, called "Circles of Forces", define the area which a vehicle is able to reach in a particular time. The green arrows show possible future trajectories in this area.

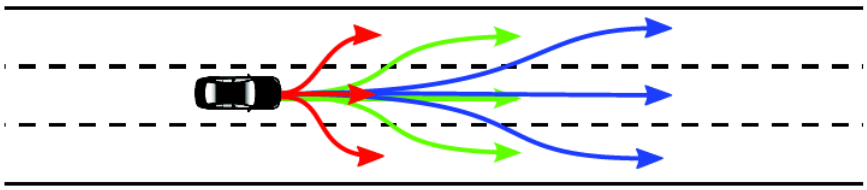


Fig. 3: The arrows show example trajectories for the allowed maneuvers. Red arrows label braking maneuvers. Green arrows show trajectories which keep their speed and the blue ones indicate an acceleration of the vehicle.

Figure 3.2: Trajectory generation for different speed values, image from [17].

certainty properly, they add some extension to the RRT such as bias sampling, which starts exploring the node from the front of the car generating practical trajectories. Vehicle turning radius is also taken into account in order to expand the tree for practical movement of the car between two points. They incorporate the rules of the road to deal with static or dynamic obstacles. The planner always ensures that the ego car can stop at the future uncertain events and then start moving within that predicted environment. In order to evaluate the risk, they differentiate drivable and non-drivable regions, lane marks, non-stopping zone, etc.

Sang-Hyun and Seung-Woo [18] proposed candidate trajectory generation and selecting the optimal one using a trained data set. The sample trajectories are obtained from a Gaussian Process's posterior distribution. Then the set of the samples is evaluated with maximum entropy inverse reinforcement learning [46] with a learned cost function. They learn the optimal set of trajectory from the expert driver in a supervised fashion.

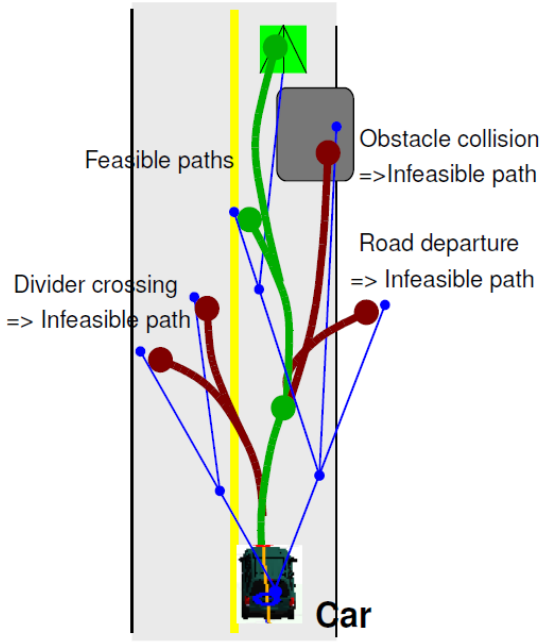


Figure 3.3: Motion plans are propagated using the vehicle's dynamical model. Propagated paths are then evaluated for feasibility, image from [15].

3.2.2 Control logic

Control logic-based planners take ideas of the control theories. They try to correct the error towards the goal in a closed feedback loop. The goal can be following a lane/trajectory, passing a target vehicle etc.

Model Predictive Control(MPC) [34] is one of most popular control theories. In order to design the controller it takes the model of the process into account and has the predictive ability in the future time steps and optimizes the cost function from the current step to the future step based on a set of constraints. It is an iterative, finite-horizon optimization-based controller proved to be used in many applications including different aspects of self-driving car. The method Scenario MPC (SCMPC) [8] takes the future traffic scenario and directly controls the ego car for lane change assistance(LCA) and automated highway driving(AHD). LCA is initiated by a human driver as a part of Advanced Driver Assistance Systems(ADAS). They take into ego vehicle's bicycle kinematic motion model, maximum speed, maximum acceleration and road boundaries. They also define necessary constraints for lane keeping, switching

and following a vehicle in the lane (adaptive cruise control). They predict the traffic intention over the finite horizon to handle the uncertainties.

In [14], the authors describe the MPC based controller to overtake a vehicle in a two-way road. They generate a feasible trajectory with a convex optimization method minimizing yaw acceleration for practical steering values and considering the vehicle dynamics. The MPC constraints are designed such that it follows the trajectory and avoid collision with other vehicles by driving all the wheels within the allowed boundary, Figure 3.4.

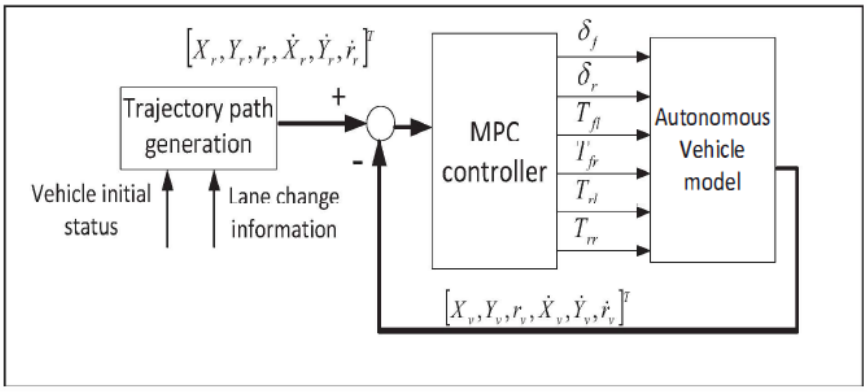


Figure 3.4: The control architecture of the autonomous vehicle system [14].

Fuzzy controller [30] is another useful controller popularly used in Robotics. It is based on fuzzy logic that evaluates a value in terms of some logical variables that takes a continuous value between 0 and 1. The term "fuzzy" refers to something that is not true or false but how much partially true/false in a continuous manner. In a fuzzy control system, there are membership functions based on expert knowledge. The input is mapped by the membership functions and a rule is defined on it to take action to achieve a certain goal. [26] used fuzzy logic to design a controller for overtaking maneuvers. They localize the vehicle with a Global Positioning System (GPS) and send the information to all other vehicles over a wireless network. To follow a lane the proposed system uses a straight path fuzzy controller that maintains a constant speed. In order to overtake a slow-moving vehicle in the right lane, they at first check if the left adjacent lane is empty using the vehicle positioning and the speed information. The vehicle switches to the immediate left lane with a lane change

fuzzy controller. It takes the lane coordinates into account so no trajectory is required. The fuzzy steering controller takes a lateral and angular error as crisp values and compares with the predefined expert values to steer either to left or right until the vehicle is centered in the lane. In order to pass, the vehicle now uses a fuzzy lateral controller using the right vehicle as a reference to pass. When the right lane is free enough they switch back the vehicle to the right lane. See Figure 3.5.

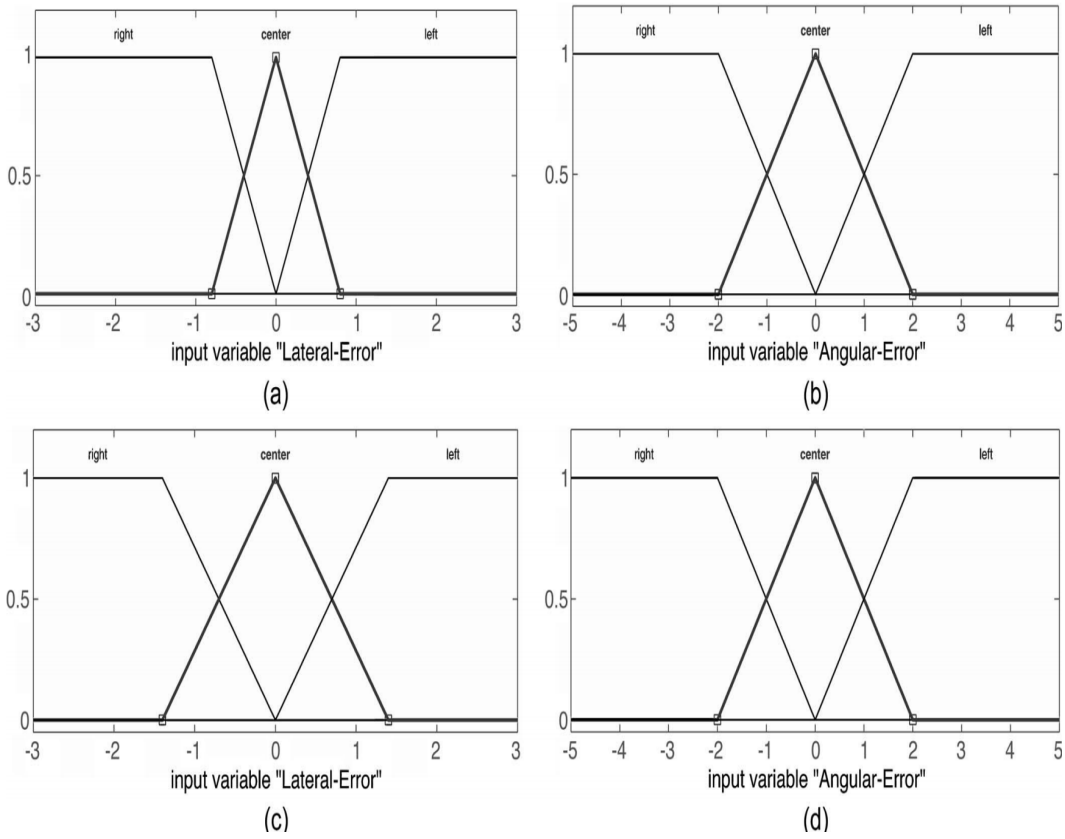


Figure 3.5: The membership function definition for the input fuzzy variables. (a) Straight-path-tracking error. (b) Straight-path-tracking angular error. (c) Lane-change lateral error. (d) Lane-change angular error, image from [26].

3.2.3 Mathematical Function

Mathematical function based planners take inspiration from mathematical constraints or cost functions. They convert the rules of the road or driving parameters into numeric values, thresholds and equations. These methods often

derive necessary and sufficient conditions to generate practical driving commands given the sensor data or state information.

One of the most popular motion planners for local collision avoidance is ORCA (Optimal Reciprocal Collision Avoidance) [5]. They derive sufficient conditions for multiple robots to guarantee collision avoidance without communicating with one another, assuming all the robots follow the ORCA strategy to decide their actions. The key idea behind their method is use of **Velocity Obstacle** [12] (VO) and **Reciprocal Velocity Obstacle** [6] (RVO).

VO refers to the set of all velocities for an ego agent that results in collision between it and the other moving agents within some short time interval in the future with the assumption that they retain constant velocity for that time period. Therefore if an agent chooses a velocity within this velocity space, it will collide. A general intuition of avoiding collision is to choose a velocity outside this set.

In RVO, the ego agent assumes the other agent takes half the responsibility to avoid the collision. This approach can also guarantee that choosing a velocity outside the RVO space induced by the other agent will avoid the collision even when they pass each other from the same side. In this aspect, the agent tries to choose a velocity close to its current velocity. The chosen velocity might avoid the collision but could be highly deviated from the goal-directed velocity and often might end up in a deadlock. ORCA solves this problem by finding velocity u which is the smallest change to the relative velocity of the two approaching agents. . It can be extended for n number of agents by finding u for all the participating agents with the help of linear programming. But the main limitation of the method is that it is assumed that all the agents in the environment follow the same protocol which is very much unpractical for autonomous vehicles driving alongside the human drivers. The method also assumes constant velocity whereas the acceleration is a huge factor in the future position of the high-speed vehicles. Besides, solving a linear program might lead to an infeasible solution.

The risk level set method proposed by Pierson et. al. [32] has a cost function which takes density and motion of the non-ego agents and derives an occu-

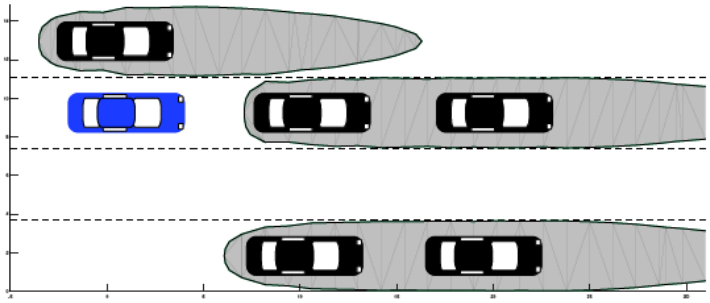
pancy grid in the road surface and a greedy Djikstra [10] planning algorithm over the unoccupied spaces to navigate in a crowded traffic, see Figure 3.6. The occupancy grid comes from a Gaussian function multiplied by a logistic function which helps the shape to be skewed towards the direction of driving which resembles the uncertainty in the state estimation of the non-holonomic motion of the vehicles. In other words, the shape has more occupancy in the direction it is driving and lower on both sides. They assume all the agents in the environment are "self-preserving" which means they are cooperative and follow the rules of the road e.g., changing lane in such a way that do not go into the braking distance of the other vehicles. This is very unpractical because all the cars in the road can not be autonomous right at this moment and human drivers are prone to do mistakes while changing lanes. The method does not address inappropriate lane change and how to avoid accidental scenarios, although it works under sudden speed change of the non-ego vehicles within the same lane.

3.3 Learning based

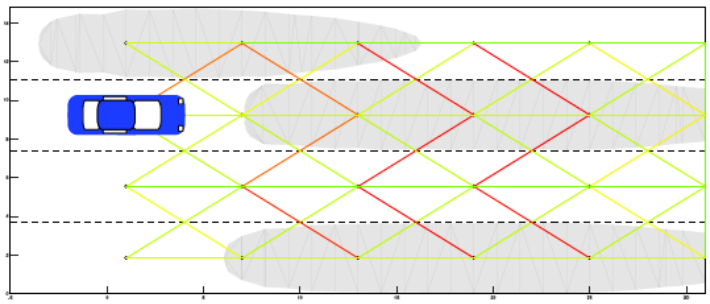
3.3.1 Supervised Learning

Supervised learning based planners try to learn and mimic pre-recorded driving behavior. The project ALVINN [33] mentioned in chapter 1 is a good example of this category. However, the supervised methods for the task of lane change are not popular due to their lack of negative samples in the training set e.g., collision data. These methods also do not generalize well. Training in one type of data set hardly performs well on a new set of data.

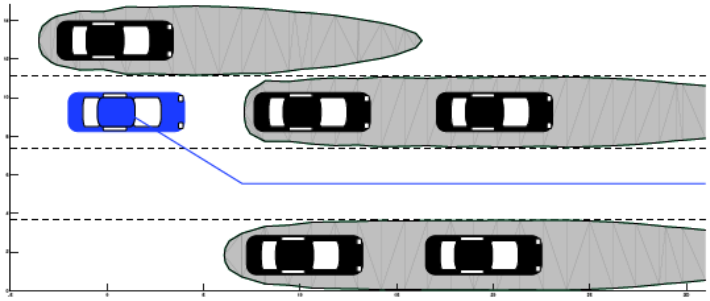
In some literature [27], lane changing task is divided into 3 types: Mandatory Lane Change(MLC), Discrepancy Lane Change(DLC) and Anticipatory Lane Change(ALC). MLC refers to the behavior that the driver must leave the current lane. DLC improves the driving conditions by adjusting speed and finding a better gap. The ALC describes how to avoid traffic congestion [41] [2]. The DLCE method [27] addresses a supervised way of executing the DLC for a lane change. They used the classical machine learning algorithm SVM to



(a) Calculation of H around ego agent (blue)



(b) Graph of planning horizon



(c) Planned route

Figure 3.6: (a) Occupancy grid obtained from Gaussian function H, (b) discretizing road spaces, (c) using Dijkstra to calculate the shortest path, image from [32].

accept or reject the adjacent gap in the target lane. The training data comes from NGSIM(next-generation simulation program), a labeled lane changing data collected from a highway in California.

Tobias et. al [35] propose high-level decision making for lane change to the left. They collect human triggered lane change data with the help of a simulator and train the parameters of a Bayesian Network (BN) which uses a Logistic Regression (LR) to model when is the right time to initiate the lane change maneuver. Their feature vector includes time to collision and time gap

in the next lane to model the traffic scenario.

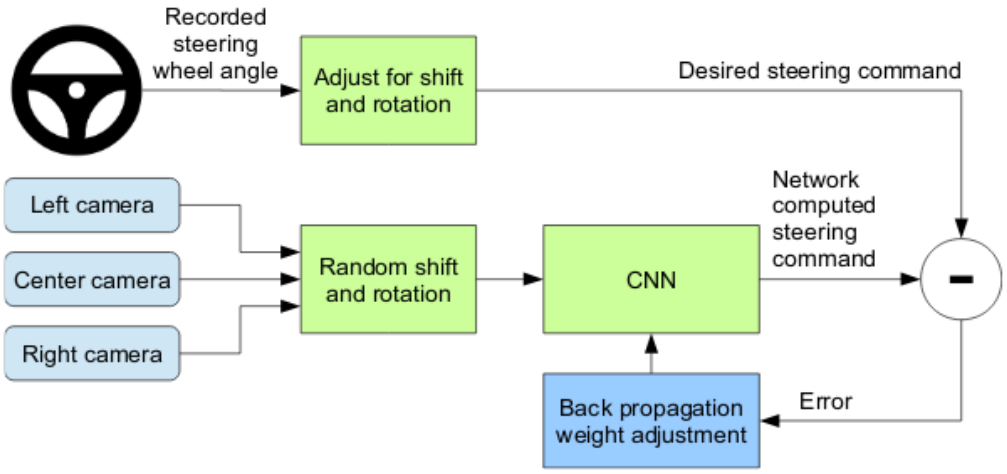


Figure 3.7: (a) E2E learning architecture by Nvidia, image from [7].

Nvidia [7] proposed a convolutional neural network(CNN) approach to autonomous driving. Here given an image of the road the network is able to generate appropriate steering command. With this method, they were able to drive on the highway, change lane and turn from one road to another. They gathered training data by recording from the human driver's steering wheel command and corresponding video of the road surface ahead of the car. Three cameras were set at left, center and right which later helped them increase the training samples by random shift and rotation and adjusting the steering wheel commands accordingly, Figure 3.7. The neural network has nine layers, a normalization layer, five convolutional layers and three fully connected layers at the end, totaling 250k parameters. Since the process is end to end, it is hard to plug with a global planner and the performance of such a system deteriorates with new driving scenarios that are absent in the training data.

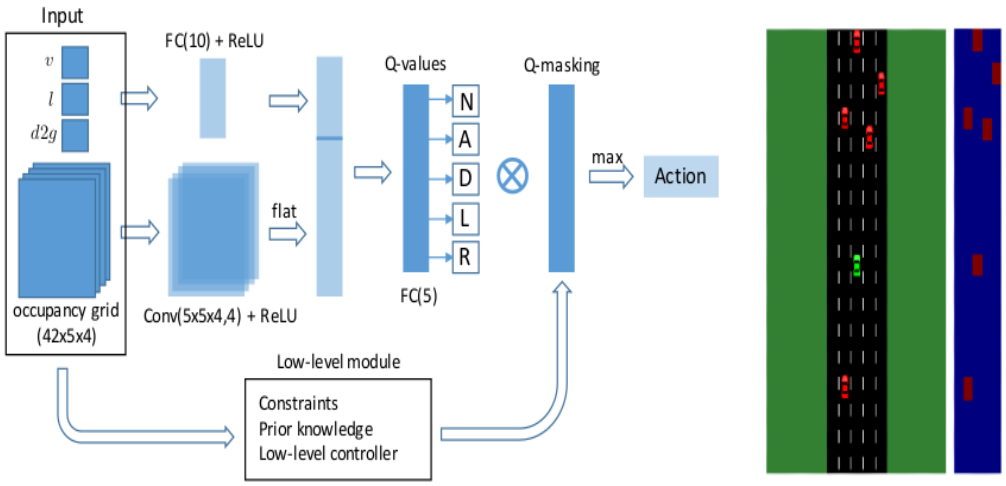


Figure 3.8: Network architecture, simulator and occupancy grid of [25].

3.3.2 Reinforcement Learning

The reinforcement learning-based planner tries to learn from both good and bad samples. This type of planner is comparatively newer and in most cases, they take advantage of the simulated environment. One of the challenges of this approach is to transfer the knowledge from the simulator to the real-world or training the planner on an actual robot which is a very expensive procedure.

The work of Mustada et. al. [25] is very similar to ours. They address the problem of exiting the highway through proper lane change. They take high-level decisions to speed up, speed down, stay in-lane or switch to right/left based on some predefined low-level controllers. They use a DQN algorithm to train on a simulator. The input to the network is velocity, current lane, distance to the goal lane and history of 4 past binary occupancy grids. The grids are passed through the convolutional layer and flattened and concatenated with the output of the fully connected layer with the rest of the scalar input layer, see Figure 3.8.

In this method, the reward is very sparse, +10 for a successful exit and -10xl (the rightmost lane is 0 and the value l increases by 1 from right to left) for failure. They introduce a Q-masking which compares the max of Q values from the network with predefined knowledge to verify if the action is valid (not choosing beyond the availability of the lane) or safe (not colliding based

on the time to collision). This method is also very similar to the safety system mentioned in Deep Traffic [13], which overrides the action taken by the output of the network whenever the network fails to avoid a future collision.

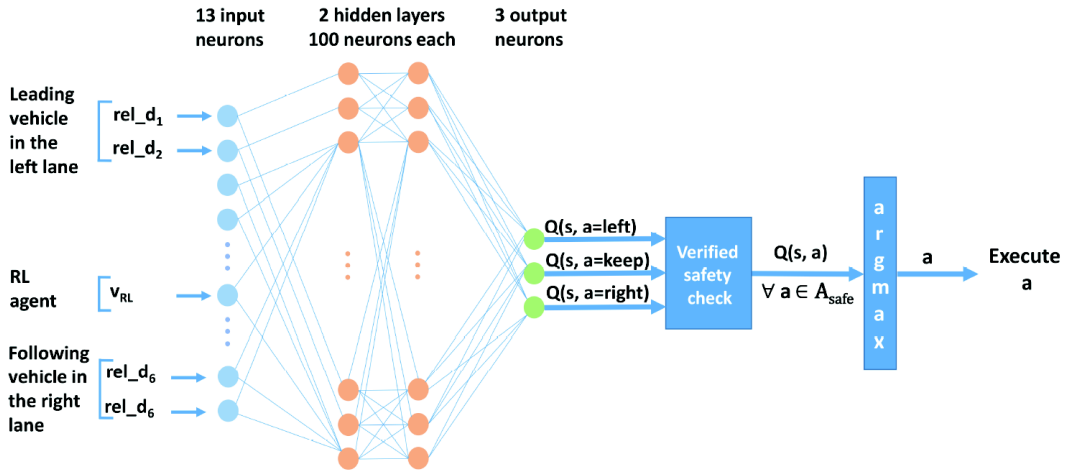


Figure 3.9: Network architecture of [21].

The work of Branka et. al. [21] is also very relevant to our work. They train a DQN agent to perform the lane change while maintaining the desired velocity. There is also a safety verification to filter the appropriate action selected from the network. They take the relative distance and velocity of the leading and following vehicle in the current and adjacent one lane on the left and right. The actions are switch to left, keep lane and switch to the right. The neural network has 2 layers each having 100 fully connected neurons, see Figure 3.9. The reward is the negative value of the absolute difference of the desired velocity and the current velocity of the ego vehicle. They show a better performance than an expert knowledge incorporated rule-based controller [1]. But this methodology is not suitable for the accident-prone environment, because it does not see other vehicles beyond the adjacent lanes. All the 3 methods we have discussed so far formulate driving as Markov Decision Process (MDP) and they learn by interacting with the simulator environment.

Difference between Ours and State-of-the-art RL planners

In this subsection, we summarize the difference between the above methods with our method.

The main difference is the use of a classical planner. In our action set, we have one additional action that is coming from a classical planner. This helps to reuse the classical planner and also to overcome the limitation of it. This is applicable where the classical planner is sub-optimal in some uncertain scenarios. Therefore our approach takes advantage of both state-of-the-art RL planners and any existing rule-based planner that we discussed in this chapter.

Another difference is the state vector. The occupancy grid has both position and velocity information of the cars. The state information also has more fine-grain data because of the subdivision of each lane into 3 corridors. We argue that this detailed information helps the neural network to capture the features related to uncertainty in the motion of the surrounding vehicles.

Our reward function is also different than the above mentioned RL methods. It encourages to safely exit the current lane, avoid the proximity of the other cars and even by slowing down if necessary.

Chapter 4

Lane changing with DRL motion planner

4.1 Problem overview

We are trying to solve lane changing on the highway. The task is to control an ego vehicle that moves autonomously to the rightmost lane without collision. This scenario happens frequently when we drive close to freeway exit in everyday life. The other agents in the environment are non-communicating i.e, their intentions are unknown. Some of the vehicles do not follow the rules of the road properly. They can change lane randomly or speed up/down suddenly.

4.2 Our methodology

State-of-the-art[23][22] implementations of deep reinforcement learning use an action space over the primitive actions and a neural network that maps an input state to a policy over primitive actions. To take advantage of classical planning methods, we treat them as action functions that can be queried with a state input and gives an action suggestion. Our method is an implementation of DQN with an augmented action space with both primitive actions and action query functions by classical planning methods.

4.2.1 2D simulator

The driving simulator used in this project consists of 4 lanes in a 2D space. Each lane is subdivided into 3 corridors. There are 19 vehicles in total within

a 200 meter range. Vehicle types include car and motorcycle. A car occupies three corridors and a motorcycle occupies one corridor. We map the pixels of the simulator into meters. A car is represented as a $(width = 2m, height = 4m)$ rectangle and a motorcycle is a $(0.6m, 1.5m)$ rectangle.

The vehicles are placed based on their descriptions (e.g. initial speed, location, behavior) specified in a configuration file. The behavior includes random or fixed speed, random lane change and self-preserving nature (giving right of way to other vehicles by slowing down or stop by braking if needed) during the run. If all the vehicles are initialized with self-preserving mode and random speed during the run, there is no collision. The vehicles do not communicate with each other. Given a target corridor, the vehicle can start changing lanes towards it with a fixed linear speed. However, it does not go to the next corridor in the next simulator step. It requires a few simulator steps to reach, based on the target speed and position of the target corridor from the current corridor. We design the simulator such that if a new target corridor comes before executing the previous corridor change command, it starts the new instruction from its current position. By introducing a delay in between the simulator step we can map the speed of the car and acceleration to approximate with the real-world values. But during training, we do not specify the delay in order to train it faster and later on we map it. The values are no exact and only the relative differences of the values with respect to one another are significant.

4.2.2 Experimental Setup

We devise a challenging environment where non-ego vehicles change lane randomly without any sort of safety measures and give rise to the adversarial situation where they can collide an existing vehicle in the next lane or step into the braking distance of the other vehicles, eventually leading to a collision in the future time stamps. Out of 19 vehicles, seven can change lane randomly with 0.01 probability. The vehicles with higher speed than ego vehicle disappear from the top and reappear at the bottom at the random location of the lanes. This way we ensure very diverse vehicle distribution. The speed is also

chosen randomly from 20 kmph to 80 kmph .

Task

During the experiment, we initialize the ego vehicle at the left-most lane whose goal is to reach towards the rightmost lane. The simulator input-output values can be used for an MDP framework. We can pass an action to the simulator, it will execute the action and return its state after a single step taken in it. The state also comes with a reward and a binary value if the task was complete or failure due to a bad action.

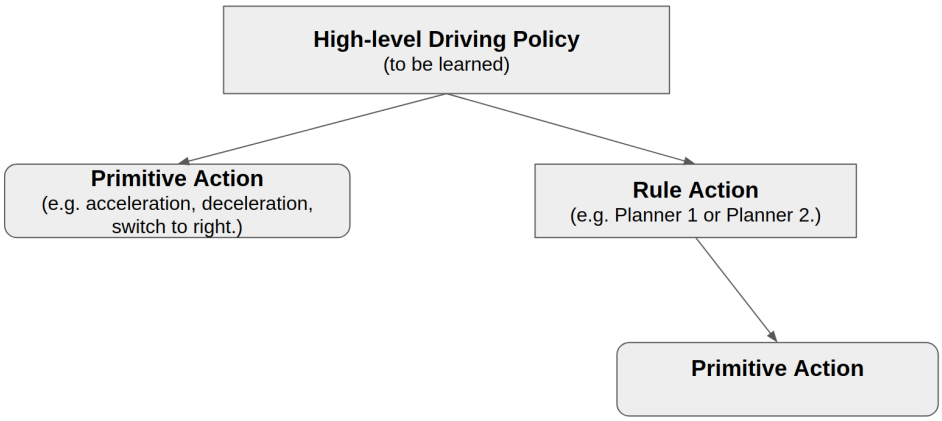


Figure 4.1: Hierarchy of the action selection by the learned policy

Classical Planning Methods

We implemented three non-learning based planning methods. The methods do not consider the lane change information of the other vehicles. The planner P1 and P2 described below are inspired by the human driver and are rule-based. The planner P3 is a mathematical function based planner.

Planner P1: Planner 1 mimics the basic lane change strategy of a new driver. If there are sufficient gaps in the front of the ego vehicle in the current lane and both the front and back in the right lane then the ego vehicle switches right; otherwise, follows the front vehicle in the current lane with a PID controller for a target speed. If there is no vehicle in the front but the right lane is occupied, a target speed of the speed limit is applied.

Planner P2: This method is more complex than planner P1. It mimics advanced human driving by checking both the gaps in the right lane and the speed of the closest cars in the right lane, to ensure that the ego vehicle will not run into the braking distance of the other vehicles. We used the mapping table of "speeds and stopping distances" by the State of Virginia to calculate braking distance for different speeds. [1]

Planner P3: This planner is an implementation of the risk level sets [32] described in the section 3.2.3, Figure 3.6. For the correctness check of our implementation, we tested it in a simplified scenario where all the other vehicles do not change lane and show "self-preserving" behavior. We noted that our implementation was able to ensure collision-free driving as claimed in their paper.

We learn the high-level driving policy by selecting appropriate actions such as primitive action or a rule-based planner. The planner outputs in terms of primitive actions see figure 4.1. In our method, planner P1 and P3 are used as an action function to augment the action space of the DRL algorithm. Without loss of generality, our method can also work with any classical planning methods added into the action space.

States

We follow a similar occupancy grid-like state representation similar to Deep Traffic [13]. The columns in the grid represent lateral state information around the car. We take into account the current lane and 2 more lanes on right and left totaling 5. At each time step (16 ms), the simulator returns the observations of the positions, speeds, distances of the other vehicles in the ego-centric view. It also returns collision and safety breaking events information. We set the safety distance threshold to two meters from the front and back of the ego vehicle.

Along the longitudinal direction, we take 50 meters ahead and 50 meters back of the ego car and discretize it with 1 meter per cell resulting in a grid of shape 100x5. The cell values contain the speed of the vehicles occupying

[1] `https://law.lis.virginia.gov/vacode/46.2-880/`

them otherwise 0. We differentiate the non-driving area with -100 and the cells containing ego vehicle with 100 (any vehicle has a maximum speed of 80 kmph). We normalize the state by diving with 100 to get the cell values within -1 to 1.

Actions

We define the vehicle command actions in terms of high-level commands by setting the target speed and target corridor. The low-level controller does the job of speeding up or changing lane over the future time steps. During training, we map each of the actions to some target speed and target corridor and send it to the simulator. For the primitive action rl agent, the actions are [accelerate, no action, decelerate, switch to next right lane]. In our proposed method the action set is executed in augmented nature, [accelerate, no action, decelerate, switch to next right lane, Planner]. The Planner can be any classical planner for changing lanes.

The "accelerate" action applies a constant acceleration of $3m/s^2$. The "decelerate" action applies a deceleration of $4m/s^2$. The "no action" applies no action and the momentum of the car is kept. It requires a few simulator steps in order to reach the next right lane. During switching to the right lane we keep a fixed longitudinal speed of 50 kmph and fixed vehicle angle of 20 degree. These are set by hyper-parameters, can be changed easily. We assume the low-level controller takes care of this part.

Reward

The simulator can return the ground truth of the positions, speed, distances between the vehicles and collision. We define 20 meters from back and front as safety distance threshold. If any vehicle breaks this proximity we consider it as safety distance broken.

Whenever the ego agent reaches right we give +10, for each collision -10, for each safety distance broken -1. If the agent fails to reach right within 8000 simulator step we give a penalty of -10, otherwise -0.001 to encourage it to reach right quickly.

Reinforcement learning agents need to interact with the simulator continuously through episodes. So at the beginning of each episode, we initialize the ego car at the leftmost lane. An episode is terminated if reaching the rightmost lane successfully or fails with a collision or safety breaking.

4.2.3 Algorithm Setup

We use a modified version of OpenAI DQN baseline[2]. We design a custom environment like openAI gym by wrapping the simulator and making it gym compatible. The primitive DQN agent's neural network's input layer has the same size as the state occupancy grid. There are three hidden layers, each of them having 128 neurons with the "tanh" activation function. The last layer has four (the number of actions) outputs, which are the Q values for the four actions for the given state. The learning rate is 10^{-4}, the buffer size for experience replay is 10^6, the discount factor is 0.99, and the target network update frequency is 100. An epsilon-greedy strategy for exploration was used for action selection. With probability ϵ, a random action is selected. With probability $1 - \epsilon$, the greedy action, $a^* = \arg\max_{a \in \mathcal{A}} Q(s, a)$ is selected at a given state s. In each episode, the value of ϵ starts from 0.9 and diminishes linearly to a constant, 0.02.

Our method is also implemented with a DQN agent, which has the same neural networks architecture as the primitive action agent, except that the output layer has 5 outputs, which include the Q values for the four same actions as the primitive agent plus the Q value estimate for Method P1 or P3. The learning rate, buffer size, discount factor, target network update frequency and exploration factor are completely the same as the primitive agent.

[2] `https://openai.com/blog/openai-baselines-dqn/`

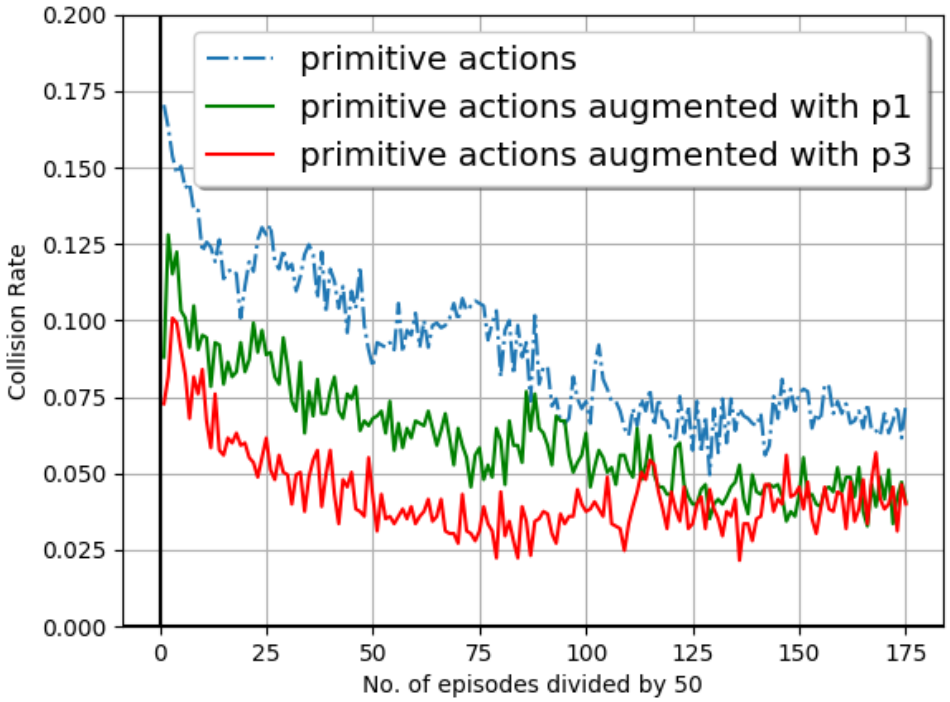


Figure 4.2: Learning curve (Collision): our method (an RL agent with primitive actions and augmented skill actions) vs. primitive actions RL agent.

4.3 Experiment and Results

Table 4.1: *The adversary lane changing task: Performance of our method, primitive action reinforcement learning, human and three planning methods. "Ours-P" is the our method with P being the additional classical planner besides primitive actions.*

	Ours-P1	Ours-P3	primitive agent	human	P1	P2	P3
collision	2.1%	2.4%	6.0%	16.0%	14.2%	11.6%	9.9%
success	85.0%	81.3%	70.1%	79.2%	69.4%	69.6%	71.7%
avr. speed	54.7	51.5	57.6	48.0	55.2	54.1	58.0

Figure 4.2 shows the learning curve for the collision rate of every 50 episodes for the two types of agents. Thus the x-axis is the number of training episodes divided by 50. The y-axis shows the collision rate in the past 50 episodes. The curves show that our method learns much faster than the primitive agent. With the augmented planning method providing action suggestions, we effec-

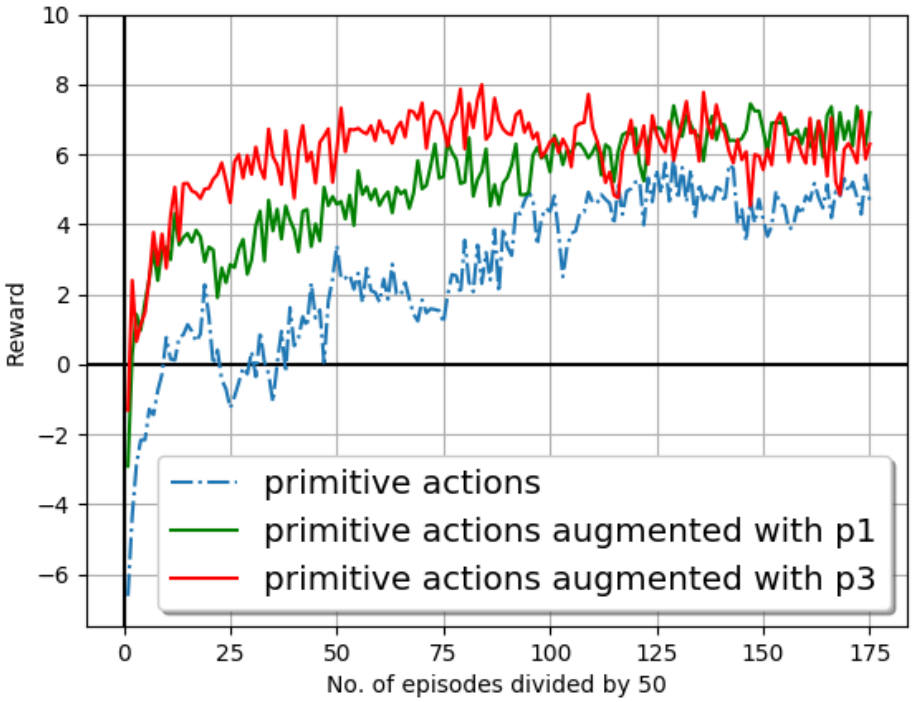


Figure 4.3: Learning curve(Reward): our method (an RL agent with primitive actions and augmented skill actions) vs. primitive actions RL agent.

tively reduce the amount of time and samples in order to learn a good collision avoidance policy. Figure 4.3 shows that our method also learns larger rewards in the same amount of training time.

We also tested the final performance after training finishes in $10,000$ episodes for both the primitive agent and our agent. In addition, we also implemented a gaming system using Logitech G29 consisting of driving wheels, acceleration and deceleration paddles, to collect human performance data. Three human testers were recruited. Each tester was trained for 30 minutes. Their best performance over 30 trials was recorded. In each trial, 25 episodes were attempted. Finally, their performances were averaged to get the human performance index.

Table 4.1 shows the performance of our method compared to the primitive agent and human. Our method performs better than both the primitive agent and human, achieving a low collision rate of 2.1%. This low rate was

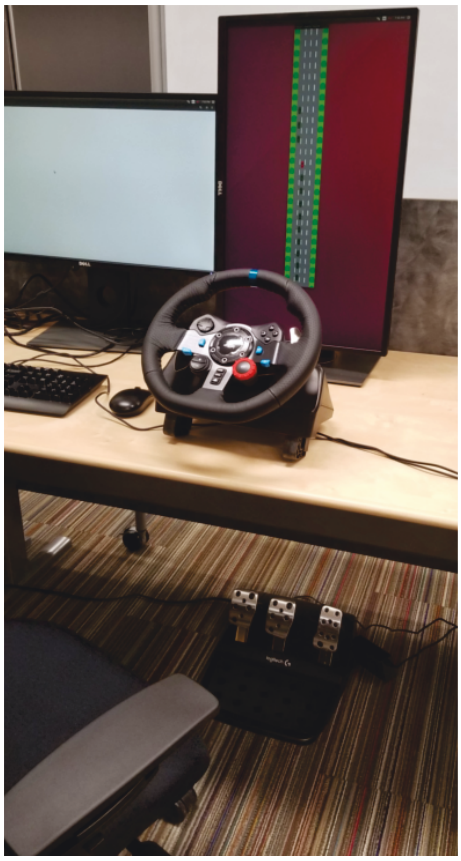


Figure 4.4: The system used for collecting driving performance data from human testers: A logitech driving wheel, acceleration and braking paddles, and a chair.

achieved with a similar average speed to primitive agent and human. In terms of the rate of successfully reaching the rightmost lane within the limited time, our algorithm achieves 85.0%, which is much higher than the primitive agent (70.1%) and human (79.2%). It seems human testers tend to drive at slow speeds to reach a good success rate. Although collision is unavoidable in this adversary setting, the performance of our method is very impressive. Note that at the end of training shown in Figure 4.2, the collision rate of our method was around 4% instead of being closing to our testing performance, 2.1%. This is due to that at the end of the training, there is still a random action selection with a probability of 0.02 used in epsilon-greedy exploration.

The table also shows the collision rate of Method P1 is 14.2% on this adversary setting. This poor performance is understandable because Method P1

was developed in a much simpler, non-adversary setting. The other planning methods P2 and P3, although perform better than P1, still cannot solve the adversary task with satisfactory performance. The method P3 works collision-free in non-adversary case. But in our setting, it performs poorly (9.9%). One of the reasons is that the method does not consider other vehicle's motion during the lane change. The interesting finding here is that by calling Method P1 in our method as augmented action, we learn to avoid collision faster as well as improve the collision rate of Method P1 or P3 significantly by using reinforcement learning for active exploration. Thus our method achieves the goal of reusing classical planning as skills to speed up learning.

Figure 1.3 shows the successful moments of driving with our agent. The first row shows a sequence of actions applied by our agent that successfully merge in between two vehicles on the right. Specifically, the first moment accelerates; the second moment cuts in front of the vehicle on the right; and the third and fourth moments merge in between two other vehicles on the right. The second row shows our agent speeds up and successfully passes other vehicles on the right. The third row helped with annotations of the surrounding vehicles. In the first moment, our vehicle is looking for a gap. The second moment, *v3* switches left, creating a gap and the ego car switches right into the gap. In the following moments, the ego car keeps switching right because there are gaps on the right.

4.4 Analysis of the learned Q values

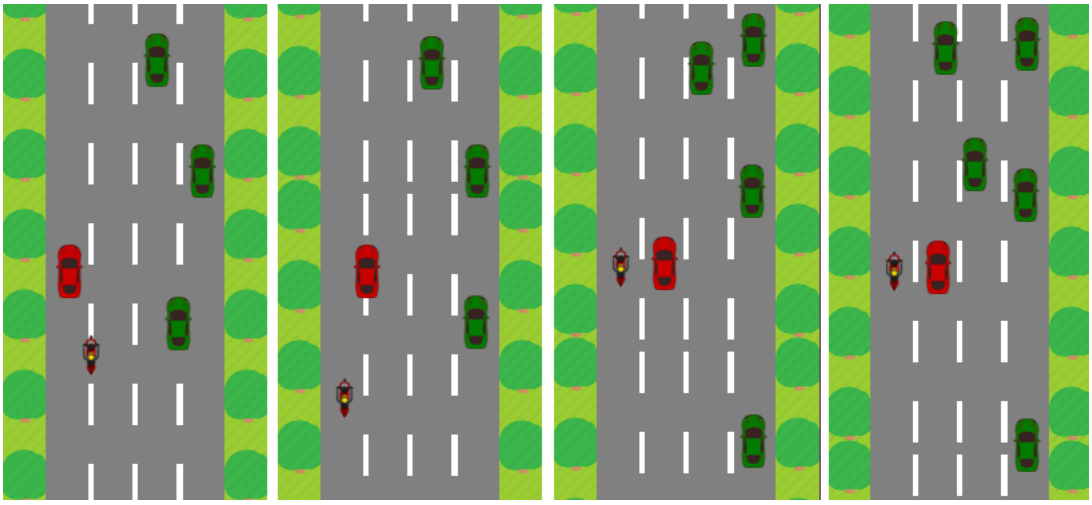


Figure 4.5: Sampled moments: Q values for different actions.

The advantage of using reinforcement learning for autonomous driving is that we can learn evaluation function for actions at any state. With classical planning, knowledge represented is not clear unless reading the code.

Figure 4.5 shows a few sampled moments of the primitive rl agent. The first moment (accelerating), the action values are, $[0.851, 0.841, 0.829, 0.844]$; the second moment (decelerating), the action values are, $[1.030, 1.042, 1.043, 1.036]$; the third moment (accelerating), the action values are, $[1.421, 1.416, 1.406, 1.418]$ and the fourth moment (decelerating), the action values are, [1.316, 1.324, 1.334, 1.319]. Let us take the first moment for example, the ego vehicle was selecting the "accelerate" action because the action value corresponding to the acceleration action is the largest (0.851). So the acceleration action was chosen (according to the argmax operation over the Q values). These values can also be used for giving a warning signal as a part of the advanced driver assistance system (ADAS).

Figure 4.6 shows the $Q(s, a = switch\_right)$ at a number of successive moments. The left color plot shows the values of switching right within the time window. The middle moments have the largest values for switching right; while at the two ends, the values are small, indicating the switching right is not favorable because collision will occur. The right color bar is the color legend.

The middle shows the trace of the car in the time window that corresponds to the left color plot (dotted line). It clearly shows that the best moment of switching right is when the ego car moves near to the middle line between the two vehicles on the right. This finding means that our method has the potential to be used to learn and illustrate fine-grained driving knowledge that is conditioned on distances and speeds of other vehicles. These values are useful for driver warning systems as a part of assistance.

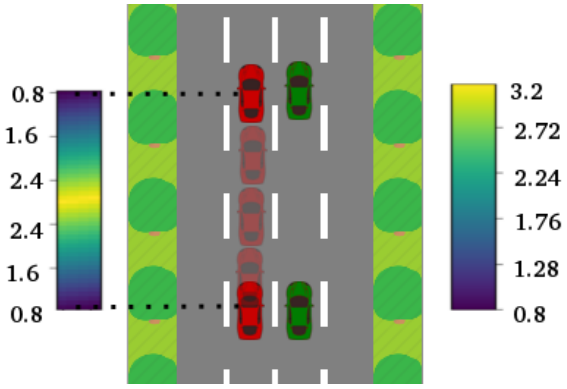


Figure 4.6: $Q(s, a = switch\_right)$ at a number of successive moments

Chapter 5

Conclusion

In this thesis, we design a deep reinforcement learning-based motion planner that reuses classical planner and works under the uncertain driving scenarios. The problem is very challenging in that the other vehicles may change lane to collide with our ego vehicle at a random time step. We proposed a novel way of combining classical planning methods with naturally defined primitive actions augmented with the planners. The key finding in this work is that this method learns faster for collision avoidance and performs better than the primitive-action reinforcement learning agent. The comparison with human testers is promising, which shows our new method performs better than the average performance of three testers.

Chapter 1 of the thesis introduced how self-driving car software is structured and the motivation of this research on this domain. The proposed solution is aimed at improving safety under uncertain environments.

Chapter 2 reviewed the reinforcement learning background and relevant connections of RL to classical motion planners. We also discussed Deep Reinforcement learning(DRL) and the different components of this learning method. The classical planners are mostly non-learning based. The chapter described how DRL can be used to design a motion planner.

In chapter 3, we define the problem of lane changing. The chapter also described state-of-the-art solution approaches. The solutions are divided into learning and non-learning methods. Non-learning methods are also known as rule-based methods. One such method is a sampling-based method which

usually generates multiple trajectories and the best trajectory is chosen after considering the constraints related to the problem. Another approach is the use of control theory to directly control the steering angles and the speed of the vehicle to do the lane change task. The function-based approaches apply the constraints on the state information and output appropriate behavior or controls. In the supervised type of learning method, the learning agent tries to mimic pre-recorded good driving behavior. On the other hand, reinforcement learning methods learn by interacting with the environment and learn from mistakes.

Chapter 4 described the proposed solution, experimental setup and results. The research is useful in the sense that it is reusing the existing knowledge of the classical planners instead of learning the knowledge from scratch. However, one major limitation of this work is to train on a real car. It would require a lot of driving hours. Another drawback is that our method is highly dependant on the low-level controller. So if the low-level controller changes, the system might require retraining. Future work is to compare with human testers in a first-person view on a 3D simulator.

References

[1] T. K. G. Aachen, "Pelops white paper," *Technical Report,Institute for Automotive Engineering Aachen and the BMW Group*, 2010. 29

[2] Ahmed and K. Iftekhar, "Modeling driver's acceleration and lane changing behavior," *Massachusetts Institute of Technology*, 1999. 25

[3] P.-L. Bacon, J. Harb, and D. Precup, *The option-critic architecture*, 2016. arXiv: 1609.05140 [cs.AI]. 17

[4] R. Bellman, *Dynamic programming*. Courier Corporation, 2013. 11

[5] V. D. Berg, G. J., S. J., M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," *Springer Robotics research*, pp. 3–19, 2011. 6, 24

[6] V. D. Berg, Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," *International Journal of Robotics Research*, 1988. 24

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, *End to end learning for self-driving cars*, 2016. arXiv: 1604.07316 [cs.CV]. 27

[8] G. Cesari, G. Schildbach, A. Carvalho, and F. Borrelli, "Scenario model predictive control for lane change assistance and autonomous driving on highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 3, pp. 23–35, 2017, ISSN: 1941-1197. DOI: 10.1109/MITS.2017.2709782. 21

[9] S. Chamraz and R. Balogh, "Two approaches to the adaptive cruise control (acc) design," in *2018 Cybernetics Informatics (K I)*, 2018, pp. 1–6. DOI: 10.1109/CYBERI.2018.8337542. 5

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," numerische mathematik," *Numerische mathematik*, pp. 269–271, 1959. 3, 5, 25

[11] J. M. Fakoor. Mahdi Kosari. Amirreza, "Revision on fuzzy artificial potential field for humanoid robot path planning in unknown environment," *International Journal of Advanced Mechatronic Systems*, 2015. 5

[12] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research*, 1988. 24

[13] L. Fridman, J. Terwilliger, and B. Jenik, *Deeptraffic: Crowdsourced hyperparameter tuning of deep reinforcement learning systems for multi-agent dense traffic navigation*, 2018. arXiv: 1801.02805 [cs.NE]. 29, 34

[14] C. Huang, F. Naghdy, and H. Du, "Model predictive control-based lane change control system for an autonomous vehicle," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 3349–3354. DOI: 10.1109/TENCON.2016.7848673. 5, 22

[15] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using rrt," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1681–1686. DOI: 10.1109/IROS.2008.4651075. 17, 19, 21

[16] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. DOI: 10.1177/02783640122067453. eprint: https://doi.org/10.1177/02783640122067453. [Online]. Available: https://doi.org/10.1177/02783640122067453. 19

[17] A. Lawitzky, D. Althoff, C. F. Passenberg, G. Tanzmeister, D. Wollherr, and M. Buss, "Interactive scene prediction for automotive applications," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 1028–1033. 17, 18, 20

[18] S. Lee and S. Seo, "A learning-based framework for handling dilemmas in urban automated driving," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1436–1442. DOI: 10.1109/ICRA.2017.7989172. 20

[19] N. Z. Liang Ma Jianru Xue, "Efficient sampling-based motion planning for on-road autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2015. 5

[20] K. I. M. Buehler and S. Singh, "The darpa urban challenge:autonomous vehicles in city traffic," *vol. 56. springer*, 2009. 3

[21] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," Sep. 2018. DOI: 10.1109/ITSC.2018.8569448. 5, 29

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. 31

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015. 12, 13, 15, 31

[24] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008. 17

[25] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," Dec. 2017. 28

[26] J. E. Naranjo, C. Gonzalez, R. Garcia, and T. de Pedro, "Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 438–450, 2008, ISSN: 1558-0016. 22, 23

[27] J. Nie, J. Zhang, X. Wan, W. Ding, and B. Ran, "Modeling driver's acceleration and lane changing behavior," *Intelligent Transportation Systems (ITSC)*, 2016. 25

[28] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," *tech. rep.,DTIC Document*, 1969. 3, 5

[29] B. Paden, M. Cap, S. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, 2016. 2, 4

[30] W. Pedrycz, *Fuzzy Control and Fuzzy Systems (2Nd, Extended Ed.)* Taunton, UK, UK: Research Studies Press Ltd., 1993, ISBN: 0-86380-131-5. 22

[31] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, 41:1–41:13, Jul. 2017, ISSN: 0730-0301. DOI: 10.1145/3072959.3073602. [Online]. Available: http://doi.acm.org/10.1145/3072959.3073602. 17

[32] A. Pierson, W. Schwarting, S. Karaman, and D. Rus, "Navigating congested environments with risk level sets," *ICRA*, pp. 1–8, 2018. 6, 17, 24, 26, 34

[33] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems 1*, pp. 305–313, 1989. 1, 25

[34] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Jan. 2009. 21

[35] T. Rehder, W. Muenst, L. Louis, and D. Schramm, "Learning lane change intentions through lane contentedness estimation from demonstrated driving," *Intelligent Transportation Systems (ITSC)*, 2016. 5, 26

[36] T. G. S. Brechtel and R. Dillmann, "Probabilistic mdp-behavior planning for cars," *14th International Conference on Intelligent Transportation Systems*, 2011. 3

[37] N. Sakib, H. Yao, H. Zhang, and S. Jui, "Single-step options for adversary driving," *Nuerips Autonomous Vehicle Workshop*, 2019. arXiv: 1903.08606 [cs.AI]. iii

[38] S. Shalev-Shwartz, N. Ben-Zrihem, A. Cohen, and A. Shashua, "Long-term planning by short-term prediction," *CoRR*, vol. abs/1602.01580, 2016. [Online]. Available: http://arxiv.org/abs/1602.01580. 17

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction (2nd Edition)*. MIT press, 2018. 12, 16

[40] C. Szepesvári, *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010. 12

[41] T. T., H. Koutsopoulos, and M. Ben-Akiva, "Modeling integrated lane changing behavior," *Journal of Transportation Research Board*, pp. 30–38, 2003. 25

[42] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, and A. A. et al, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, 2006. 1

[43] S. Ulbrich and M. Maurer, "Probabilistic online pomdp decision making for lane changes in fully automated driving," *16th International Conference on Intelligent Transportation Systems*, 2013. 3

[44] C. Urmson, J. A. Bagnell, C. R. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. E. Rybski, S. Scherer, R. Simmons, S. Singh, *et al.*, "Tartan racing: A multi-modal approach to the darpa urban challenge," 2007. 17

[45] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992. 12

[46] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, 2008, pp. 1433–1438. 20