Reinforcing Classical Planning for Adversary Driving Scenarios

Nazmus Sakib[1], Hengshuai Yao [2], and Hong Zhang[3]

***Abstract*— Adversary scenarios in driving, where the other vehicles may make mistakes or have a competing or malicious intent, have to be studied not only for our safety but also for addressing the concerns from public in order to push the technology forward. Classical planning solutions for adversary driving do not exist so far, especially when the vehicles do not communicate their intent. Given recent success in solving hard problems in artificial intelligence (AI), it is worth studying the potential of reinforcement learning for safety driving in adversary settings. In most recent reinforcement learning applications, there is a deep neural networks that maps an input state to an optimal policy over primitive actions. However, learning a policy over primitive actions is very difficult and inefficient. On the other hand, the knowledge already learned in classical planning methods should be inherited and reused. In order to take advantage of reinforcement learning good at exploring the action space for safety 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. We show two advantages by doing so. First, training this reinforcement learning agent is easier and faster than training the primitive-action agent. Second, our new agent outperforms the primitive-action reinforcement learning agent, human testers as well as the classical planning methods that our agent queries as skills.**

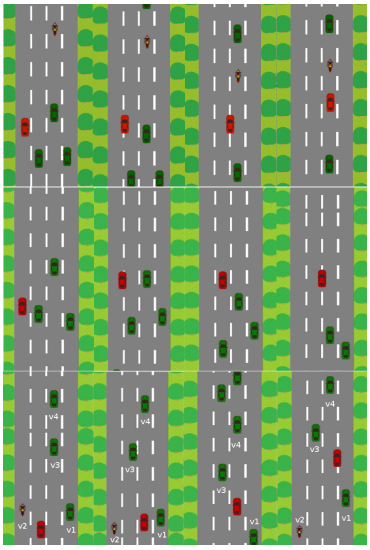

Fig. 1: Successful moments of driving with our method: merging (row 1), passing (row 2) and finding gaps (row 3).

I. Introduction

In this paper, our problem context is autonomous driving. The question for us to explore in the long term is, can computers equipped with intelligent algorithms achieve superhuman level safe driving? The task of autonomous driving is very similar to game playing in the sequential decision making nature. Although driving is not a two-player game leading to a final win or loss, accident outcomes can still be treated as loss. In driving, 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. However, unlike games, driving poses a unique challenge for reinforcement learning with the stringent safety requirement. Although the degree of freedom is relative small for vehicles, the fast moving self-motion, high-dimensional observation space and highly dynamic on-road environments pose a great challenge for artificial intelligence (AI).

Human drivers drive well in normal traffic. However, human beings are not good at handling accidents because a human driver rarely experiences accidents in one's life regardless of the large amount of accident-free driving time. In this regard, the highly imbalanced positive and negative driving samples poses a great challenge for supervised learning approach for training self-driving cars. We believe that the prospect of autonomous driving is using programs to simulate billions of accidents in various driving scenarios. With a large scale of accident simulation, reinforcement learning, already proven to be highly competitive in large and complex simulation environments, has the potential to develop the ultimately safest driving softwares for human beings.

This paper studies the problem of adversary driving where vehicles do not communicate with each other about their intent. We call a driving scenario *adversary* if the other vehicles in the environment can make mistakes or have a competing or malicious intent. Adversary driving are rare events but they can happen on the roads from time to time, posing a great challenge to existing state-of-art autonomous driving softwares. Adversary driving has to be studied, not only for our safety, but also to address the safety concerns from the public in order to push the technology forward. However, most of the state-of-art classical planning algorithms for autonomous driving do not consider adversary driving, usually assuming all the agents in the environment are cooperative. For example, the other vehicles are assumed to be "self-preserving", which actively avoid collisions with other agents whenever possible, e.g.,

[1]Nazmus Sakib is with Department of Computing Science, University of Alberta. Work done during an internship at Huawei. nazmus@ualberta.ca

[2]Hengshuai Yao is with Huawei Noah's Ark Lab, Edmonton, Canada. hengshuai@gmail.com

[1]Hong Zhang is a Professor at Department of Computing Science, University of Alberta. hzhang@ualberta.ca

see (Pierson et al., 2018). Optimal Reciprocal Collision Avoidance (ORCA) (Van Den Berg et al., 2011) is a popular navigation framework in crowd simulation and multi-agent system for avoiding collision with other moving agents and obstacles. The traffic that ORCA generates is cooperative, by planning on each vehicle's velocity to avoid collision with others. Recently, (Abeysirigoonawardena et al., 2019) pointed that there is practical demand to simulate adversary driving scenarios in order to test the safety of autonomous vehicles.

Our work is the first attempt to solve non-communicating adversary driving. We use a reinforcement learning approach. Driving has a clear temporal nature, *the current action has an effect on choosing the actions in the future*. Reasoning which action to apply by considering its long-term effects is usually called a temporal credit assignment, which is usually modeled as a reinforcement learning problem. In most recent reinforcement learning applications, there is a deep neural networks that maps an input state to an optimal policy over primitive actions. 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. In a simulated driving environment, deep reinforcement learning was found to be much inferior to classical planning and supervised learning, in both the performance and the amount of training time (Dosovitskiy et al., 2017).

On the other hand, autonomous driving field has already practised a rich set of classical planning methods. It is worth pointing out that the problem of classical 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. The problem of classical planning methods is the existence of logic holes and corner cases that were not considered at the time of developing. As a result, updating, testing and maintaining these software modules is the true darkness of this method: huge intensive human engineering labor is involved. However, the knowledge already learned in classical planning methods should be inherited and reused. Implemented classical planning softwares in the autonomous driving industry have logic holes but they perform well in the general scenarios that they are developed for, usually accounting for perhaps a large percentage of everyday driving. How do we reuse these state-or-art planning softwares and solving their logic holes automatically without human engineering?

To summarize, in autonomous driving, the field has implemented a rich set of classical planning methods but they have logic holes that originate from either oversight or mistakes in the process of software engineering. Deep reinforcement learning that learns a policy over primitive actions is slow to train but they can explore the action space for the best action though large numbers of simulations. 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 gives an action suggestion. Our reinforcement learning agent will be able to select over the action suggestions by classical planning methods as well as the primitive actions. Our method 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 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 where 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 the similar failure cases in the future.

Our work opens the door to a novel architecture solution for autonomous driving: building a decision hierarchy of skills using classical planning or learning-based methods, and calling them as augmented actions by reinforcement learning. To provide contextual background, we discuss classical planning, learning-based control, reinforcement learning, the end-to-end and the hierarchical decision making structure in the remainder of this section.

A. Classical Planning and Learning-based Control

We categorize control methods into two classes. The first is classical planning, where a control policy is derived from a system model. Methods including proportional integral derivative (PID) controllers , model-predictive controller (MPC) and rapidly-exploring random tree (RRT) are examples of classical planning. The second class is learning-based control, usually used in pair with a deep neural networks. Learning-based control is *sample based* or *data driven*. Both supervised learning and reinforcement learning are learning-based methods. Supervised learning systems aim to replicate the behavior of human experts. However, expert data is often expensive, and may also impose a ceiling on the performance of the systems. By contrast, reinforcement learning is trained from explorative experience and is able to outperform human capabilities. Note that though, supervised learning can be useful in training certain skills of driving. For example, a supervised learning procedure is applied to imitate speed control by human drivers via regulating the parameters in a linear relationship between speed and shock to the vehicle on off-road terrains (Stavens et al., 2007).

The advantage of classical planning is that algorithms are easy to program, and have good performances though often with significant efforts in parameter tuning. For example, ORCA (Van Den Berg et al., 2011) has many parameters and difficult to tune. Classical planning methods usually require significant domain knowledge, and they are often sensitive to

the uncertainties in the real-world environment (Long et al., 2017). Learning-based control, on the other hand, enables mobile robots to continuously improve their proficiency and adapt to the statistics of real-world environments with the data collected from human experts or simulated interactions.

B. Classical Planning and Reinforcement Learning

Classical planning methods have already been widely adopted in autonomous driving. Recent interests in using reinforcement learning also arise in this new application field. We comment that this is not incidental. Specifically, there are a few common fundamental principles in the core ideas of classical planning and reinforcement learning.

First, temporal relationship between the actions selected at successive time steps is considered in both 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, there is a cost function defined over a time horizon for the next few actions. The cost function is one special case of the (negative) reward function in reinforcement learning. MPC relies a system model and an optimization procedure to plan the next few optimal actions. The collision avoidance algorithm using risk level sets 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 (Pierson et al., 2018). 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 (Lawitzky et al., 2013). 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 autonomous driving field (Urmson et al., 2007; Montemerlo et al., 2008; Shalev-Shwartz et al., 2016). In reinforcement learning, low-level options and a high-level policy over options are separately learned (Bacon et al., 2016). 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 (Peng et al., 2017).

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 (Kuwata et al., 2008). 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.

C. Autonomous Driving and Reinforcement Learning

The prospect of developing a superhuman level driving agent using reinforcement learning is intriguing. Training reinforcement learning-equipped vehicles on the road is not practical, and so far the efforts have been mainly in simulators. A truthful and easily configurable simulator is crucial to develop reinforcement learning-based agents. The literature has seen many recent efforts on this. For example, CARLA is an open-source driving simulator that is specifically designed to develop for training learning-based agents (Dosovitskiy et al., 2017), with a benchmark comparison of both supervised learning and reinforcement learning agents. DeepTraffic implements a reinforcement learning agent for lane control in high-way driving, and hosts a competition for fine tuning the parameters for their implemented algorithm (Fridman et al., 2018). In this paper, we also developed a high-way lane control simulator that can be easily customized.

A small wheel-vehicle trained with reinforcement learning navigate with collision avoidance in a pedestrian-rich environment (Chen et al., 2017). Long-term driving strategies are generated by a hierarchical temporal abstraction graph (Shalev-Shwartz et al., 2016). A robot RC car was trained to navigate in an complex indoor environment using an efficient method that predicts multiple steps on a computation graph (Kahn et al., 2018).

D. End-to-End and Hierarchical Decision Making

The end-to-end approach is the state-of-art architecture for learning-based agents, with remarkable success in hard AI games due to reinforcement learning (Mnih et al., 2015a; Silver et al., 2017b) and well practised with supervised learning for high-way steering control in autonomous driving (Net-Scale Technologies, 2004; Pomerleau, 1989; Bojarski et al., 2016). Such end-to-end systems usually use a deep neural networks that takes in a raw, high-dimensional state observation as input and predicts the best primitive action at the moment. The end-to-end approach is flat, containing only *a single layer of decision hierarchy*. On the other hand, there is also evidence that most autonomous driving architectures follow a hierarchical design in the decision making module (Montemerlo et al., 2008; Urmson et al., 2007).

Our insight is that hierarchical decision making structure is more practical for autonomous driving. Imagine a safety driver sitting at the back of the wheel in a self-driving car. Monitoring an end-to-end steering system and reading the numerical steering values in real time is not practical for a fast intervention response. However, a hierarchically designed steering system can tell the safety driver a keeping-lane behavior is going to occur in the next two seconds. It is easier to monitor such behaviors in real time, and is able to interrupt timely in emergent situations. Not only for safety drivers, system designers need to understand autonomous behaviors in order to improve programs. Future passengers will also be more comfortable if they can understand the real-time behavior of the vehicle they are sitting in.

However, the current hierarchical design of the decision making module in driving softwares is highly rule and heuristic based, for example, the use of finite-state-machines for behavior management (Montemerlo et al., 2008), and heuristic search algorithms for obstacle handling (Montemerlo et al., 2008). Remarkably similarly, these algorithms have

also dominated in the early development of many AI fields, yet they were finally outperformed in Chess (Lai, 2015; Silver et al., 2017a), Checker (Samuel, 1959; Chellapilla and Fogel, 2001; Schaeffer et al., 2007), and Go (Silver, 2009), by reinforcement learning agents which use value function to evaluate states and training the value function using temporal difference methods aiming to achieve the largest future rewards (Sutton, 1984). The paradigm of playing games against themselves and with zero human knowledge in the form of rules or heuristics has helped reinforcement learning agents achieving superhuman-level performance (Tesauro, 1995; Silver et al., 2017b).

The remainder of this paper is organized as follows. Section II contains the details about our method. In Section III, we conduct experiments on lane changing in an adversary setting where the other vehicles may not give the way. Section IV discusses future work and concludes the paper.

II. Our Method

We consider a Markov Decision Process (MDP) of a state space $\mathcal{S}$, an action space $\mathcal{A}$, 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 $\gamma \in [0, 1)$. In this paper we treat the reward "function" R as a random variable to emphasize its stochasticity. Bandit setting is a special case of the general RL setting, where we usually only have 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). 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 (Bellman (2013)),

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

Q-learning and DQN. Based on the Bellman optimality operator, Watkins and Dayan (1992) proposed Q-learning to learn the optimal state-action value function Q^* for control. At each time step, we update $Q(s, a)$ as

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

where α is a step size and (s, a, r, s') is a transition. There have been many work extending Q-learning to linear function approximation (Sutton and Barto (2018); Szepesvári (2010)). Mnih et al. (2015b) combined Q-learning with deep neural network function approximators, resulting the Deep-Q-Network (DQN). Assume the Q function is parameterized by a network θ, at each time step, DQN performs a stochastic gradient descent to update θ minimizing the loss

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

where θ^- is target network (Mnih et al. (2015b)), 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 (Mnih et al. (2015b)), which is a first-in-first-out queue storing previously experienced transitions.

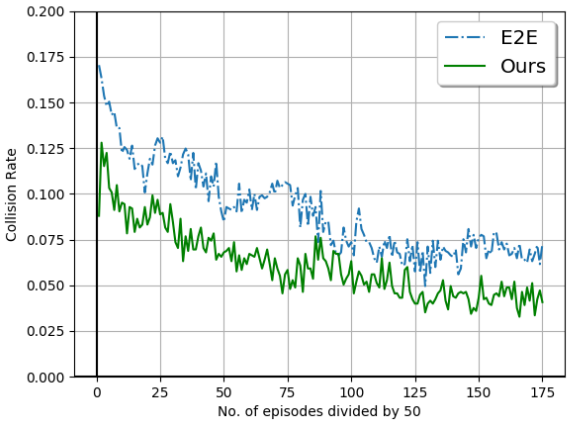

Fig. 2: Learning curves for collision rate: our method vs. the end-to-end reinforcement learning. With the augmented planning method into the action space, our method learns much faster for collision avoidance.

Augmented action space with classical planning methods. The State-of-art implementation of reinforcement learning uses an action space over the primitive actions, and a neural networks 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 augmented action space from both primitive actions and action query functions by classical planning methods.

III. Experiment

Our task is to control an ego vehicle in a lane changing task that moves itself to the rightmost lane without collision. This scenario happens frequently when we drive close to freeway exits in everyday life.

A. The Adversary Lane-change Simulator

The driving simulator consists of 4 lanes in a 2D space. Each lane is subdivided into 3 corridors. There were 19 vehicles in total within a 200 meter range. All the vehicles do not communicate with each other. Seven other vehicles can change lane randomly with probability 0.01 at each time step. When they change lane, there is no safety function applied, which poses a great challenge to control the ego vehicle safely. Faster vehicles than the ego vehicle disappear

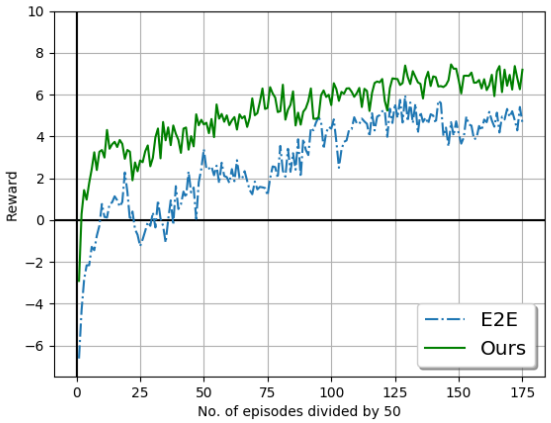

Fig. 3: Learning curves for the reward: our method vs. the end-to-end reinforcement learning. Our method achieves much larger rewards in the same amount of training time.

from the top of the window and then reappear at the bottom at random lanes with a random speed ranging from 20 km/h to 80 km/h. In this way we ensure a diverse traffic congestion. Vehicle types include car and motorcycle. A car occupies three corridors and a motorcycle occupies one corridor. We map the pixels of simulator into meters. A car is represented as a $(width = 2m, height = 4m)$ rectangle and a motorcycle is a $(0.6m, 1.5m)$ rectangle.

State Representation. We used occupancy grid as state representation (Fridman et al., 2018). The grid columns correspond to the current lane plus two lanes on each of the left and right sides of the car. At each time step, the simulator returns the observations of the positions, speeds, distances of the other vehicles in ego-centric view. It also returns collision and safety breaking events. (We set the safety distance threshold to two meters from the front and back of the ego vehicle.)

Along the y-direction, we take 50 meters in the front and 50 meters in the back of the ego car and discretize the zone with one meter per cell resulting in a grid of shape $(5, 100)$. The value of a cell is the speed of the vehicle in the cell; if no occupying vehicle in the cell, the cell value is zero. We normalize the occupancy grid matrix by diving with the speed limit.

Reward. Whenever the ego agent reaches the rightmost lane, a positive reward of 10.0 is observed. For collisions, a -10.0 negative reward is given. For each safety distance breaking event, a negative reward of -1.0 is observed. If the agent fails in reaching the rightmost lane within $8,000$ simulation steps, a negative reward of -10.0 is given. A constant reward -0.001 is given at the other time steps to encourage reaching the goal quickly.

Reinforcement learning agents need to interact with the simulator continuously through episodes. For each interaction 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.

Classical Planning Methods. To the best of our knowledge, there is no state-of-art classical planning methods that work for this non-communicating adversary driving scenario. We implemented three planning methods by assuming all the other vehicles do not change lane.

Method P1: 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, switch right; otherwise, follow the front vehicle in the current lane with a PID controller for a target speed. If there is no vehicle in the front and the right lane is occupied, a target speed of the speed limit is applied.

Method P2: This method is more complex than Method 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 none of the vehicles will run into the breaking distance of the ego vehicle.

Method P3 is an implementation of the risk level sets (Pierson et al., 2018). For the correctness check of our implementation, we tested it in a simplified scenario where all the other vehicles do not change lane. We noted that our implementation was able to ensure collision free driving as claimed in their paper.

Note that Methods P2 and P3 are just for reader's information about how competitive they can be relative to Method P1 in the adversary setting. They are not used as a serious comparison to our method because they are developed for non-adversary settings. In our method, Method P1 is used as an action function to augment the action space. Without loss of generality, our method can also work with other classical planning methods added into the action space.

B. Algorithm Setup

Action Space. For the End to End (E2E) DQN agent, the actions are "accelerate", "no action", "decelerate" and "switch right". 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. The "switch right" action will start changing to the right lane lane with a fixed linear speed. It requires a few simulator steps in order to reach the right lane. For our method, the action space is augmented with Method P1.

The E2E DQN agent's neural network: The 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 4 (the number of actions) outputs, which is the Q values for the four actions given the 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 0.1 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 E2E agent,

TABLE I: *The adversary lane changing task: Performance of our method, end-to-end reinforcement learning, human and three planning methods.*

	Ours	E2E	human	P1	P2	P3
collision	2.1%	6.0%	16.0%	14.2%	11.6%	9.9%
success	85.0%	70.1%	79.2%	69.4%	69.6%	71.7%
avr. speed	54.7	57.6	48.0	55.2	54.1	58.0

except that the output layer has 5 outputs, which include the Q values for the four same actions as the E2E agent plus the Q value estimate for Method P1. The learning rate, buffer size, and discount factor, target network update frequency and exploration is completely the same as the E2E agent.

C. Results

Figure 2 shows the learning curves. For every 50 episodes, we computed the collision rate. 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 E2E agent. With the augmented planning method (Method P1) providing action suggestion, we effectively reduce the amount of the time and samples in order to learn a good collision avoidance policy. Figure 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 E2E agent and our agent. In addition, we also implemented a gaming system using Logitech G29 consisting driving wheels, acceleration and deceleration paddles, to collect human performance data as shown in Figure 4. 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 I shows the performance of our method compared to the E2E agent and human. Our method performs better than both the E2E agent and human, achieving a low collision rate of 2.1%. This low rate was achieved with a similar average speed to E2E 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 E2E (70.1%) and human (79.2%). It seems human testers tend to drive at slow speeds to reach a good success rate. Because collision is unavoidable in this adversary setting, the performance of our method is very impressive. Note that in the end of training shown in Figure 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 in the end of training, there is still a random action selection with 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 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 significantly by using reinforcement learning for action exploration. Thus our method achieves the goal of reusing classical planning as skills to speed up learning. The other planning methods P2 and P3, although perform better than P1, still cannot solve the adversary task with a satisfactory performance.

Fig. 4: The system (rotated 90 degrees) used for collecting driving performance data from human testers: A logitech driving wheel, acceleration and braking paddles, and a chair.

Figure 1 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 forth 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.

D. Knowledge Learned for Driving

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 5 shows a few sampled moments. The values for the Q values (outputs from the DQN networks) are printed in the caption. 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).

Figure 6 shows the $Q(s, a = switch\_right)$ at a number of successive moments. The left color plots shows the values of switching right within the time window. 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.

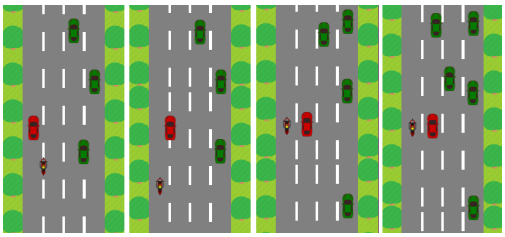

Fig. 5: Sampled moments: Q values for the actions. In the order of "accelerate", "no action", "deceleration", "switching right": the first moment (accelerating), the action values are, $[0.851, 0.841, 0.829, 0.844]$; the second moment (decelerating), the action values are, $[1.030, 1.042, 1.043, 1.036]$; 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, $[1.316, 1.324, 1.334, 1.319]$.

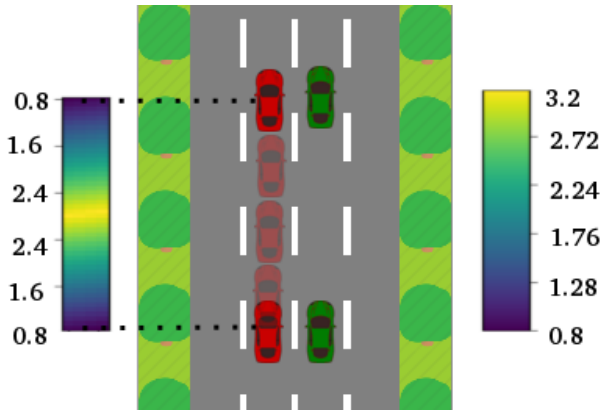

Fig. 6: The left color plot shows the values of switching right within the time window: the middle moments have the largest values for switching right; while at the two ends, the values are small, indicating the switching right is not favorable because collision will occur. The right color bar is the color legend. The middle shows the trace of the car in the time window that corresponds to the left color plot (dotted line). It shows that the best moment to switch right is near the middle line of the two vehicles on the right.

IV. Conclusion

In this paper, we studied an adversary driving scenario which is 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 to form an augmented action space for reinforcement learning agents. The key finding in this paper is that this method learns faster for collision avoidance and performs better than the end-to-end reinforcement learning agent that uses primitive actions. The comparison with human testers is exciting, which shows our new method performs better than the average performance of three testers. A future work of this paper is to compare with human testers in a first-person view.

Acknowledgment

We thank Borislav Mavrin and Hao Chen for helpful discussions of this paper. Thanks to Peyman Yadmellat, Masoud S. Nosrati, and Yunfei Zhang for their contribution to the driving simulator. We thank Joelle Pineau's research group at Mcgill University on an early version of the simulator.

References

Abeysirigoonawardena, Y., Shkurti, F., and Dudek, G. (2019). Generating adversarial driving scenarios in high-fidelity simulators. In *ICRA*.

Bacon, P., Harb, J., and Precup, D. (2016). The option-critic architecture. *CoRR*, abs/1609.05140.

Bellman, R. (2013). *Dynamic programming*. Courier Corporation.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to end learning for self-driving cars. *CoRR*, abs/1604.07316.

Chellapilla, K. and Fogel, D. B. (2001). Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, 5(4):422–428.

Chen, Y. F., Everett, M., Liu, M., and How, J. P. (2017). Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*.

Fridman, L., Terwilliger, J., and Jenik, B. (2018). Deeptraffic: Crowdsourced hyperparameter tuning of deep reinforcement learning systems for multi-agent dense traffic navigation. In *NIPS*.

Kahn, G., Villaflor, A., Ding, B., Abbeel, P., and Levine, S. (2018). Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *ICRA*, pages 1–8. IEEE.

Kuwata, Y., Fiore, G., Teo, J., Frazzoli, E., and How, J. (2008). Motion planning for urban driving using rrt. In *IROS*, pages 1681–1686.

Lai, M. (2015). Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*.

Lawitzky, A., Althoff, D., Passenberg, C. F., Tanzmeister, G., Wollherr, D., and Buss, M. (2013). Interactive scene prediction for automotive applications. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1028–1033. IEEE.

Long, P., Liu, W., and Pan, J. (2017). Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015a). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015b). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

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

Net-Scale Technologies, I. (July 2004). Autonomous off-road vehicle control using end-to-end learning.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4):41:1–41:13.

Pierson, A., Schwarting, W., Karaman, S., and Rus, D. (2018). Navigating congested environments with risk level sets. In *ICRA*, pages 1–8. IEEE.

Pomerleau, D. A. (1989). Alvinn, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is solved. *Science*, 317(5844):1518–1522.

Shalev-Shwartz, S., Ben-Zrihem, N., Cohen, A., and Shashua, A. (2016). Long-term planning by short-term prediction. *CoRR*, abs/1602.01580.

Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

Silver, D. (2009). *Reinforcement Learning and Simulation-Based Search in Computer Go*. PhD thesis, University of Alberta.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017b). Mastering the game of go without human knowledge. *Nature*, 550:354359.

Stavens, D., Hoffmann, G., and Thrun, S. (2007). Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In *IJCAI*, pages 2218–2224.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst. AAI8410337.

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

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

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

Van Den Berg, J., Guy, S. J., Lin, M., and Manocha, D. (2011). Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*.