Robotics and Autonomous Systems 114 (2019) 1–18

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning

Changxi You [a], Jianbo Lu [b], Dimitar Filev [b], Panagiotis Tsiotras [c,*]

[a] School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0150, USA
[b] Research & Advanced Engineering, Ford Motor Company, Dearborn, MI 48121, USA
[c] School of Aerospace Engineering and the Institute for Robotics & Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332-0150, USA

HIGHLIGHTS

- A new MDP is proposed to model highway traffic.
- The optimal policies for overtaking/tailgating are obtained using RL.
- Reward functions in the formulation of MaxEnt IRL can take any nonlinear form.
- New MaxEnt deep IRL algorithms are proposed to solve the model-free MDP problem.

ARTICLE INFO

Article history:
Available online 15 January 2019

Keywords:
Reinforcement learning
Inverse reinforcement learning
Deep neural-network
Maximum entropy
Path planning
Autonomous vehicle

ABSTRACT

Autonomous vehicles promise to improve traffic safety while, at the same time, increase fuel efficiency and reduce congestion. They represent the main trend in future intelligent transportation systems. This paper concentrates on the planning problem of autonomous vehicles in traffic. We model the interaction between the autonomous vehicle and the environment as a stochastic Markov decision process (MDP) and consider the driving style of an expert driver as the target to be learned. The road geometry is taken into consideration in the MDP model in order to incorporate more diverse driving styles. The desired, expert-like driving behavior of the autonomous vehicle is obtained as follows: First, we design the reward function of the corresponding MDP and determine the optimal driving strategy for the autonomous vehicle using reinforcement learning techniques. Second, we collect a number of demonstrations from an expert driver and learn the optimal driving strategy based on data using inverse reinforcement learning. The unknown reward function of the expert driver is approximated using a deep neural-network (DNN). We clarify and validate the application of the maximum entropy principle (MEP) to learn the DNN reward function, and provide the necessary derivations for using the maximum entropy principle to learn a parameterized feature (reward) function. Simulated results demonstrate the desired driving behaviors of an autonomous vehicle using both the reinforcement learning and inverse reinforcement learning techniques.

1. Introduction

More than six million motor vehicle crashes occurred in the US in 2015 alone, of which 27 percent resulted in injury or death [1]. From 2014 to 2015 the total number of vehicle crashes increased by 3.8 percent, and the number of fatal crashes increased by 7 percent [2]. Another study, sponsored by NHTSA, investigated 723 crashes and showed that driver behavioral error caused or contributed to 99 percent of these crashes [3]. Given the increased sophistication of automotive active safety systems, these studies suggest that driver behavior still remains the most important factor contributing to road accidents. In order to understand, characterize and, if possible, predict driver behavior in traffic, researchers have proposed different driver models based on several methodologies over the past four decades [4]. Nevertheless, driver modeling is still a difficult task since driver behavior is affected by different individual factors, such as gender, age, experience, driver's aggression, etc. The most promising idea to eliminate driver behavioral error may be to completely free the driver from the burden of driving, that is, to develop fully autonomous vehicles.

An autonomous vehicle is able to detect the environment and navigate without the driver's input, by using a variety of sensing techniques such as radar, lidar, ultrasound, localization and computer vision, along with advanced control techniques that can analyze the sensory data, in order to plan and achieve the desired path to the desired destination. Autonomous vehicles are expected

* Corresponding author.
E-mail addresses: cyou6@gatech.edu (C. You), jlu10@ford.com (J. Lu), dfilev@ford.com (D. Filev), tsiotras@gatech.edu (P. Tsiotras).

https://doi.org/10.1016/j.robot.2019.01.003
0921-8890/

to significantly improve traffic congestion, reduce collisions and resulting injuries, enhance mobility for the children, the elderly and the disabled, and reduce the need for parking space in cities [5]. Due to the rapid development of sensing and computing technologies over the past two decades, research in the field of autonomous vehicles has shown great progress, and related self-driving vehicle technology has matured significantly in the recent years [5].

The first autonomous vehicle was developed by Carnegie Mellon University's Navlab in 1988, and it was able to achieve lane-following using camera images [6]. Navlab completed the first autonomous coast-to-coast trip across the United States in 1995, traveling 2849 miles between Pittburgh and San Diego at an average speed of 63.8 mph [7]. Another important milestone in the self-driving vehicle technology was the DARPA Grand Challenge, which was held three times between 2004 and 2007 [8]. In these races the vehicles were required to drive autonomously in an off-road course (2004 and 2005) or an urban area course (2007) without any human intervention. These tests showed that fully autonomous off-road driving and fully autonomous urban driving are indeed technologically possible. Since then, many commercial companies, startups, and research organizations have launched their own development of autonomous vehicles.

Google started the self-driving car project in 2009 (called Waymo after 2016) and has already tested autonomous vehicles for about 8 million miles on public roads in six states. Waymo introduced a minivan based on a mass-production platform for the purpose of full autonomy [9]. The ride-sharing company Uber tested its first self-driving program in the mobility service sector in Pittsburgh in 2016, and plans to eventually replace all its drivers with self-driving cars in the not-so-distant future [10]. Tesla currently provides auto steering, lane changing and parking capabilities in their "Autopilot" system and plans to bring semi-autonomous and autonomous vehicle features to the mass market with the 2017 Model 3 [11]. In 2016, Ford became the first automaker to test its autonomous vehicles on snow and in darkness, and plans to deliver a commercially available fully autonomous vehicle by 2021 [12]. Volvo introduced the first large-scale autonomous drive project and plans to give 100 customers early-access to autonomous XC90 on Swedish public roads by 2017[13]. Although there were about 44 large corporations and numerous automotive driving startups working on autonomous vehicles by May 2017 [14], vehicles currently permitted on public roads are not fully autonomous and they all require a driver to take over control of the vehicle at a moment's notice.

The main technical issues in developing fully autonomous vehicles exist at three levels, namely, perception, planning and control, as shown in Fig. 1. The perception level comprises of sensing and filtering of environmental data. The sensing system consists of a number of sensors and provides information about the vehicle's state and the environment. The filtering system denoises the signals from the sensing system and provides a reasonable estimate for the unmeasurable states [4,15–17]. The planning level completes three tasks, which include mission planning, where the vehicle solves a routing problem in order to complete a task, decision making, where the vehicle chooses an appropriate action for the next time step from an available action set, and path planning, where the vehicle plans its future trajectory as a function of space or time [18–21]. Finally, the control level receives the signals from the planning level, maintains the stability of the vehicle, and tracks the desired path.

Many vehicle control techniques have been developed to enhance stability and handling performance, such as differential braking [22,23], torque vectoring [24,25], active steering [26,27] and integrated chassis control [28,29]. Particularly, driver assist systems [30–33] have been developed to provide better lane-keeping and tracking control. Numerous control techniques are also available to use at the control level. Higher-level path planning and decision making is another essential part for developing fully autonomous vehicles.

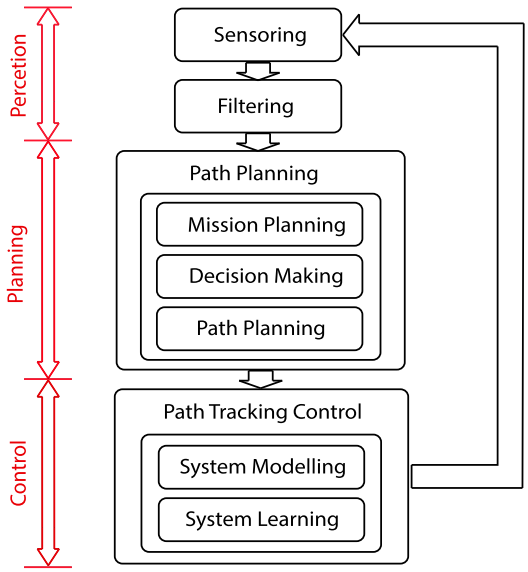

Fig. 1. Autonomous control architecture at different levels.

For example, in order to generate a smooth path for an autonomous vehicle, Choi et al. [34,35] presented a series of path planning algorithms based on Bézier curves. The planned paths have continuous curvature and satisfy the road boundary constraints. Shim et al. [36] used a parameterized 6th-order polynomial to represent a smooth path, and planned a feasible path for the autonomous vehicle satisfying both the initial/final conditions and the constraint conditions. They implemented their path-planning algorithm in static/moving obstacle avoidance tasks and designed the tracking control module using model predictive control techniques. Instead of planning a path geometrically by solving an optimization problem [34–36], one can also design a path using optimal control theory. Mousavi et al. [37] applied an extended Kalman filter to predict the future trajectory of an autonomous vehicle, and used a linear time-varying model predictive control scheme to determine the optimal path and the associated optimal control. This approach was designed to achieve collision avoidance and stochastic target tracking in a dynamic environment. Similar work was devoted to developing fast path planning or decision making algorithms to achieve fully autonomous driving in real world scenarios. Ulbrich and Maurer [38] used a partial observable Markov decision process (POMDP) to model the decision making for lane changes, and implemented a two-step algorithm in real-time to obtain the optimal action for an autonomous vehicle in an urban driving task. Kuwata et al. [20] proposed a real-time path planning algorithm based on Rapidly-exploring Random Trees (RRTs). This algorithm was implemented on an autonomous vehicle which completed a 60 mile simulated military supply mission in the 2007 DARPA Urban Challenge. A more extensive survey on path planning for autonomous vehicles can be found in [8,39].

Other techniques using ideas from artificial intelligence (AI) have also been developed to solve planning problems for autonomous vehicles. These include supervised learning [40], deep learning [41] and reinforcement learning [42]. Lange et al. [43] used a deep neural encoder to extract feature representations from the raw visual input of camera images for a racing vehicle, and successfully learned the optimal control actions (i.e., steering, accelerating and braking) using reinforcement learning. The control performance was even better than an experienced human player,

in the sense that the car was able to move along a closed track as fast as possible without crashing. The approach in [43] concentrated on improving the driving performance of a single vehicle without considering the traffic. Shalev-Schwartz et al. [18] took the traffic into consideration and divided the planning problem into two phases. They first modeled the state transition of the traffic using a deep neural network, such that they could apply supervised learning to predict the near future states of the system. Subsequently, they used a recurrent neural network to model the trajectory and learn the optimal driving policy of the autonomous vehicle. This approach does not rely on any Markovian assumption, and hence it is considered to be robust to the stochastic behavior of the environment. The learning procedure was validated using both an adaptive cruise control task and a roundabout merging task.

Application of reinforcement learning requires knowledge of the reward function, which needs to be carefully designed. An alternative is to learn the optimal driving strategy using demonstrations of the desired driving behaviors. Abbeel and Ng [44] used a driving simulator to collect two minutes of driving data from an expert driver, and assumed that the reward function of this expert driver is a linear combination of a number of known features. In order to recover the reward function and the expert driving policy, they proposed a max-margin algorithm along with a projection algorithm to solve the inverse reinforcement learning problem. Although one can approximately recover expert driving behaviors using this approach, the matching between the optimal policy/reward and the features is ambiguous, as indicated by Ziebart and his colleagues [45]. In order to address this ambiguity, Ziebart introduced the maximum entropy principle (MEP) to uniquely match the rewards with the features. He proposed the maximum entropy inverse reinforcement learning (MaxEnt IRL) algorithm [45,46], which was shown to be computationally efficient in [45]. It was implemented on a routing problem (mission planning). The researchers in [47–50] developed different versions of the MaxEnt IRL algorithm based on [45,46]. Among these, the authors of [49] formulated the maximum entropy inverse reinforcement learning problem using a deep neural network (DNN) to represent the unknown reward function. All the above formulations of the MaxEnt IRL problem require complete knowledge of the environment dynamics, and they all employ a state reward instead of a state–action–reward so that they cannot learn a complicated driving behavior showing preference on certain actions.

This paper focuses on the problem of planning for autonomous vehicles in traffic. Specifically, we wish to reproduce the decision making of an expert driver, that is, we wish to duplicate the optimal driving strategy involving several typical driver actions such as lane-shifting, lane and speed maintaining, accelerating and braking, by also considering the stochastic driving behaviors of the environmental vehicles in traffic.

The contribution of this paper is summarized as follows: (1) We propose a new MDP model to represent the stochastic behaviors of the environmental vehicles in highway traffic. This model differs from previous similar MDP models [51–54] in the sense that we take the road geometry into consideration in order to compare and analyze different driving strategies during cornering. Another advantage is that the model is easily scalable to have more vehicles and more lanes in traffic. Unlike the MDP models [53,54] that need to discretize the velocity of each vehicle in traffic, which, consequentially, tend to make the problem have a large state space, we remove the velocities of the vehicles from the MDP model and consider them either in the perception layer or in the control layer. The optimal control policy for the proposed MDP is solved using both reinforcement learning (RL) and inverse reinforcement learning (IRL). (2) We generalize the formulation of the MaxEnt IRL by using a reward function in the form of a linear combination of the parameterized features, and we show, for the first time, that the reward function in the MaxEnt IRL formulation can take any nonlinear form. Previous results of MaxEnt IRL either assumed that the reward function can be represented using a linear combination of fixed features [45–48,50], or directly used a deep neural network (DNN) to represent the reward function in the MaxEnt IRL formulation without clarifying the relation between the DNN and the parameterized feature functions [49]. (3) We propose three new MaxEnt deep IRL algorithms to solve the model-free MDP problem. Although several researchers have proposed different versions of MaxEnt IRL algorithms [45–50], these algorithms cannot be used to solve a model-free problem. Specifically, we use a deep neural network to approximate the state–action–reward, instead of the state reward, as in most of existing MaxEnt IRL formulations.

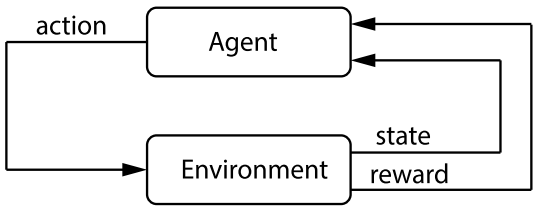

Fig. 2. The agent–environment interaction.

The rest of the paper is organized as follows: Section 2 introduces the traffic model using a stochastic MDP. Section 3 designs the reward function and solves the MDP problem using Q-learning. Section 4 introduces the maximum entropy principle and formulates the inverse optimal control problem. Section 5 summarizes and refines the MaxEnt deep IRL algorithms, and Section 6 implements both RL and IRL algorithms, and analyzes the results. Finally, Section 7 summarizes the results of this study.

2. Traffic modeling

In this section we first introduce a Markov decision process (MDP) to model the interaction between the autonomous vehicle and the surrounding vehicles in traffic. In the following sections we use both RL and IRL techniques to solve the MDP problem for the optimal policy that achieves the desired driving behaviors. We start with a brief summary of MDPs.

2.1. Markov decision process

Markov decision processes (MDPs) are used in a wide area of applications such as robotics, economics, manufacturing and automatic control. An MDP is a mathematical framework that probabilistically models the interaction between an agent and the environment, pioneered by the work of Bellman [55]. The agent is assumed to be a learner or decision maker, who interacts with the environment [42]. It receives a reward and a representation of the environment's state at each time step, and exerts an action on the environment that may change its future state. This interaction between the agent and the environment is shown in Fig. 2.

A typical MDP is represented using a 6-tuple $(S, A, \mathscr{T}, \gamma, D, R)$, where S is a (finite) set of possible states that represent a dynamic environment, A is a (finite) set of available actions that the agent can select at a certain state,[1] $\mathscr{T}$ is the state transition probability matrix that provides the probability of the system transition between every pair of the states, $\gamma \in [0, 1)$ is the discount rate that guarantees the convergence of total returns, D is the initial-state distribution, and R is the reward function that specifies the reward gained at a specific state by taking a certain action.

[1] WLOG we will assume that all actions are available in each state.

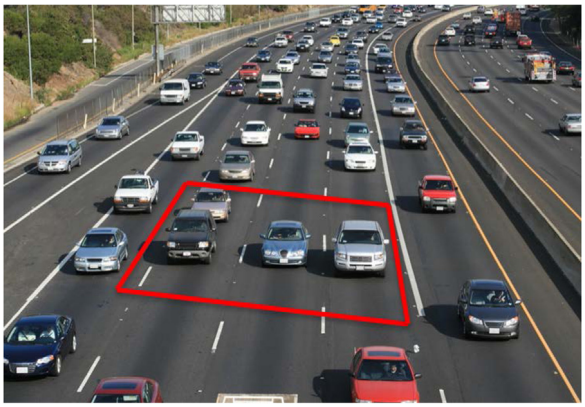

Fig. 3. The traffic on multi-lane road.

MDPs assume that the effect of taking an action at a given state only depends on the present state–action pair, and not on the previous states and actions, that is,

$$\mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = \mathbb{P}(s_{t+1}|s_t, a_t). \tag{1}$$

Eq. (1) is called Markov property [56].

The core problem of an MDP is to find a policy π for the agent, where the policy $\pi : S \rightarrow A$ specifies the action to take at the current state s_t. The goal is to find the optimal policy π^* that maximizes the cumulative discounted reward over an infinite horizon:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\Big], \tag{2}$$

where γ is the discount rate, and the term $R(s_t, \pi(s_t))$ represents the reward the agent receives by taking an action determined by policy π at the present state s_t. Given a policy π, the MDP in (1) is reduced to a Markov chain with transition probabilities $\mathbb{P}^{\pi}$, given by

$$\mathbb{P}^{\pi}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_t, \pi(s_t)). \tag{3}$$

2.2. System modeling

The MDP to be used to model the traffic is based on the following observations. Consider a typical scenario of traffic on a multi-lane road as shown in Fig. 3. Each vehicle moves in the middle of each lane with the average speed of the traffic flow.

Let us now consider the driving behavior of the blue vehicle in the middle of the red rectangle shown in Fig. 3. There are several actions the blue vehicle can take. For instance, it can maintain its current speed, accelerate or brake to occupy the vacant positions ahead of it or behind it, or move to the left or to the right lane if there is no chance for a collision. Assuming that each driver intends to maximize a certain reward function, if one can obtain the reward function of an "expert" driver by either constructing it manually or from observing real driving data, one should be able to reproduce this expert driver's behaviors using reinforcement learning techniques [42].

In the following, we designate the vehicle we want to control as the host vehicle (HV), and all remaining vehicles in traffic as the environmental vehicles (EVs). We assume that the drivers of different vehicles do not communicate with one another, and also that the vehicles do not share data with each other. Hence, the MDP system has only a single actively controlled agent. The available action set for each vehicle in traffic is given by $A \triangleq$ {"maintain", "accelerate", "brake","left-turn", "right-turn"}.

2.2.1. State definition

By considering the positions of the HV, and the number and positions of the EVs around the HV, we define the state of the MDP as shown in Fig. 4.

In Fig. 4, we use the white dashed lines to divide the road into small cells and use the green vehicle to denote the HV. The states of the MDP represent either of the three conditions shown in Fig. 4: (1) the HV is in the middle lane of the road, where we use nine cells to represent the state, and (2) and (3) where the HV is next to the road boundaries and we use six cells to represent the current state. Taking all possible combinations into account, the number of the internal-lane states is $2^8 = 256$, and the number of the left (right)-boundary states is $2^5 = 32$. Hence the total number of the states of the MDP is $256+2\times 32 = 320$. It is worth mentioning that each vehicle is viewed as a point of mass such that it will not split across multiple cells. Note that the approach can be easily extended to highways with any number of lanes and vehicles.[2]

Fig. 5 shows a possible overtaking behavior of the HV (green car) during a left-turn corner. The HV driver may prefer overtaking the pink car in front from the left rather than from the right. In order to investigate the effect of the road geometry on the observed driving behaviors of different drivers, in this work we take the road curvature into account and consider three kinds of roads, namely, left-turn, right-turn, and straight roads. The total number of the states is therefore $320 \times 3 = 960$. It is worth mentioning that, although we only consider three kinds of road geometries, one can, similarly, divide the road characteristics into more classes, as needed. For instance, one could also take into account different slopes of the roads, such as downhill, uphill, flat roads etc.

2.2.2. State transitions

We want to model the state transition process by mimicking the traffic in real world scenarios. To this end, we make the following assumptions: (1) the number of lanes n is free and greater than equal to two ($n \geq 2$), (2) the number of EVs N is free but no larger than eight, given the cell geometry of Fig. 4 ($0 \leq N \leq 8$), (3) the EVs have their own policies that may be different from the HV, (4) the EVs take a random action, (5) no collision arises from the actions of the EVs, and (6) each vehicle takes a single action at each time step.

The state transition procedure from the current state s_t to the next state s_{t+1} is given in two steps: First, the HV observes the current state s_t and selects an action $\pi(s_t)$ following its current policy. Second, the EVs respond to the action of the HV, and take an action following their own policies in a random sequence. The new positions of the HV and the EVs around the HV define the next state s_{t+1}. This state transition process is demonstrated in Fig. 6.

The current state s_t is defined using the nine cells in the red rectangle on the left graph in Fig. 6. Based on s_t, the HV may brake or switch to the right lane but these actions will result in a collision. The available safe actions of the HV are maintaining, accelerating and switching to the left lane. For instance, suppose that the HV accelerates and occupies the cell in front of it. As a consequence, the red rectangle also moves since the EVs surrounding the HV change. Next, all EVs respond to the action of the HV and take an action following a certain policy in a random order (see Algorithm 7). The next state s_{t+1} is obtained after all vehicles complete their actions (see the red rectangle on the right graph in Fig. 6).

3. Reinforcement learning

In order to obtain the desired driving policy for the HV, in this section we design the reward function and use reinforcement learning techniques to solve the MDP problem formulated in Section 2.

[2] The traffic model is also possible to be used for modeling urban traffic by adjusting the state definition. For instance, the cell size may be defined to change with the size of each EV and its velocity relative to the HV.

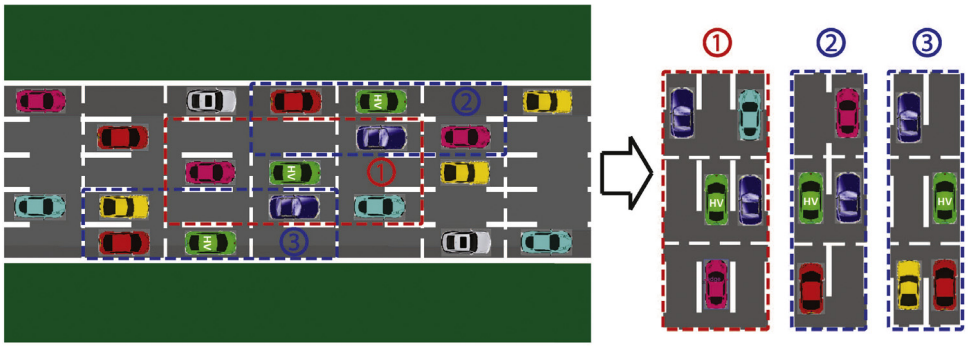

Fig. 4. The cells and the definition of the state: ① 9-cell internal-lane state, ② 6-cell left-boundary state and ③ 6-cell right-boundary state.

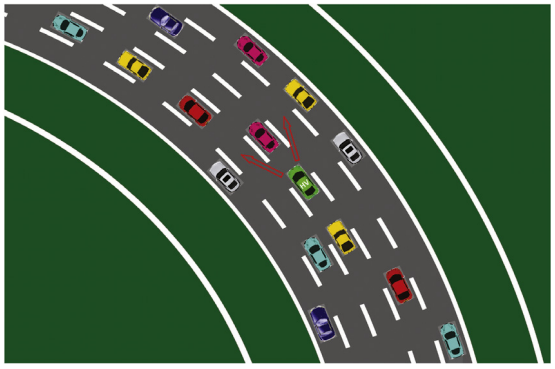

Fig. 5. Overtaking during cornering.

3.1. Reinforcement learning algorithms

The main methods used in reinforcement learning can be classified into two categories, namely, tabular solution methods and function approximation methods. The tabular solution methods are suitable for solving MDP problems with a finite (small) number of states and actions. Such methods mainly include dynamic programming, Monte Carlo and temporal-difference learning [42].

Classical dynamic programming algorithms use the value function to organize the search for the optimal policies. Such algorithms include value iteration and policy iteration methods. These algorithms require perfect knowledge of the environment, and cannot be easily applied to problems having continuous state and action spaces. Unlike dynamic programming, Monte Carlo does not require complete knowledge of the environment, but only the agent's experience, namely, the sample sequences of the states, actions and rewards from actual or simulated interactions of the agent with the environment. Monte Carlo methods are based on averaging sample returns in an episode-by-episode manner, which means that the learning of the values and the corresponding policies is performed only upon completion of each episode. Hence, Monte Carlo methods cannot update the values and policies in an on-line fashion. Temporal-difference learning is a combination of the ideas from dynamic programming and Monte Carlo methods. The most obvious advantage of temporal-difference learning over Monte Carlo is that it can be naturally implemented in an on-line fashion. Unlike Monte Carlo methods, one does not need to wait till the completion of every episode to receive the return. The most obvious advantage of temporal-difference learning over dynamic programming is that it does not require full knowledge of the environment and hence it can be implemented without a model. The two main temporal-difference learning algorithms are Sarsa and Q-learning [42,57]. The main difference between Sarsa and Q-learning is the future (state–action) values they refer to in order to update the current (state–action) values. Sarsa stands for "state–action–reward–state–action" and uses the value of the real action the agent takes in the next step following the current control policy in order to update the value of the action at the present state. Q-learning explores the maximum possible action value the agent can have in the next step, and uses this value to update the value of the action at the present state. Hence, Sarsa is an on-policy algorithm, while Q-learning is an off-policy algorithm. For many problems, both Sarsa and Q-learning are able to learn a good policy with good performance. Q-learning can potentially provide a better policy where death can be easily caused in each episode using the ϵ-greedy policy in an on-policy method [57]. The term "death" indicates the termination of an episode caused by the agent arriving at certain states (i.e., the goal state). However, Q-learning is also known to diverge in certain cases where function approximation is used [57,58]. More details on the tabular solution methods can be found in Chapters 4–6 of [42].

Function approximation methods are used to address large or continuous state space problems, where one may use a series of (nonlinear) functions to represent the values, policies and rewards. Theoretically, all methods used in the area of supervised learning are possible to use in reinforcement learning as function approximators, such as artificial neural network [59], naive Bayes [60], Gaussian processes [61], or support vector machines [62].

Since the MDP model in Section 2 has a finite number of states and actions and assumes that the agent cannot predict the behavior of the EVs, we prefer to use a tabular, model-free method to solve the corresponding optimal control problem in (2). In the following section, we first define the reward function, and then use Q-learning to learn the optimal policy that maximizes the cumulative discounted future rewards.

3.2. Reward function

The design of the reward function is a difficult task, since the driver behavior is hard to characterize and the real reward function is unknown. The reward function also differs from driver to driver and it may be even changing with time. A widely used approach to design the reward function is to represent it as a function of some manually chosen features. These features depend on the action of the agent and the state of the environment. We use a linear combination of the features to represent the reward function [44–47,63]:

$$R(s, a) = w^{\mathrm{T}} \Phi(s, a), \tag{4}$$

where w is the weight vector, and $\Phi(s, a)$ is the feature vector with each component representing a single feature point in the state–action space. Possible choices of feature points may be the binary values indicating whether a certain argument is true or not. In this work we define the features in $\Phi(s, a)$ as follows:

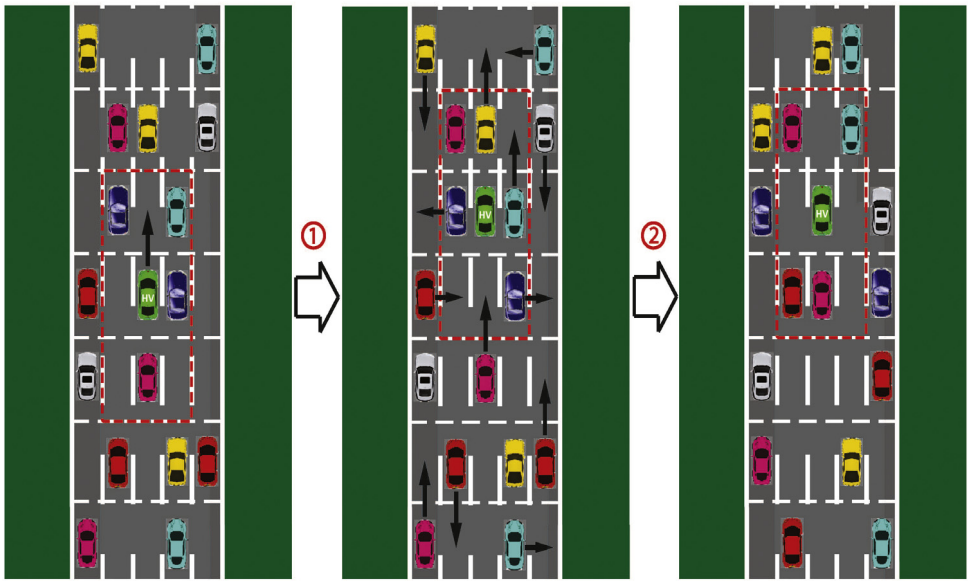

Fig. 6. State transition process.

(1) Action features. The driver may prefer taking certain actions than others if he receives a higher reward from these actions.

(2) Position of the HV. It indicates if the HV is driving next to the road boundaries. The driver may prefer to drive in different lanes, depending on the road geometry.

(3) Overtaking strategy. This feature is used to achieve different overtaking behaviors of drivers during cornering. The driver may have a different preference in regards to overtaking the car in front either from the left or from the right.

(4) Tailgating. The value of this feature is "true" if the HV is behind an EV and "false" otherwise.

(5) Collision incident. Collision occurs if the HV and a EV appear in the same cell.

One can design the weight vector w to encourage or penalize certain features using the given reward function, and then use reinforcement learning to learn the corresponding optimal policy by maximizing the total reward. Another idea is to design the reward function using a parameterized function approximator such as a Gaussian process [47,63] or a DNN [49]. The parameters of the function approximator are hard to design manually since they may not be directly related to features that have clear physical meaning, and hence they can only be learned from data. This approach will be discussed in Section 5.

3.3. Q-learning

Q-learning was first introduced in Watkins's Ph.D. thesis in 1989 [64]. In 1992 Watkins and Dayan proved that Q-learning converges to the optimum action values with probability 1 if all actions are repeatedly sampled in all states [65]. Different variants of Q-learning were developed to solve various reinforcement learning problems, such as double Q-learning that reduces overestimation [66], deep Q-learning that uses a deep neural-network to represent the value function for large state space problems [67], fuzzy Q-learning that uses fuzzy logic rules to interpret and refine the imprecise environment knowledge [68], and minimax-Q [69], Nash-Q [70], correlated-Q [71] and friend-or-foe-Q [72] that solve multi-agent reinforcement learning problems.

Next, we briefly introduce the basic Q-learning algorithm. To this end, we need to introduce two important concepts used extensively in the reinforcement learning literature, namely, the state value and the state–action value (also referred to as the action value). The value of a state s_t under policy π is denoted as $V^\pi(s_t)$, which indicates the expected discounted cumulative reward starting at state s_t and then following policy π, that is,

$$V^\pi(s_t) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \Big| s_t, \pi\Big], \tag{5}$$

where γ is the discount rate, and

$$R_t \triangleq R(s_t, a_t) = \sum_{s_{t+1}\in S} \mathbb{P}(s_{t+1}|s_t, a_t) R(s_t, a_t, s_{t+1}), \tag{6}$$

stands for the immediate reward the agent receives by taking action a_t at present state s_t, and where the term $R'(s_t, a_t, s_{t+1})$ represents the reward the agent receives by taking action a_t at present state s_t to obtain the next state s_{t+1} (see Section 5.1 for more discussion).

The values of two sequential states of the MDP are related and satisfy the following equation,

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}\Big[R_t + \gamma V^\pi(s_{t+1}) \Big| s_t, \pi\Big] \\ &= \sum_{a_t\in A} \pi(s_t, a_t) \sum_{s_{t+1}\in S} \mathbb{P}(s_{t+1}|s_t, a_t) \\ &\quad \times \Big(R'(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1})\Big), \end{aligned} \tag{7}$$

Eq. (7) is called the Bellman evaluation equation. The optimal policy π^* maximizes the associated value function at each state, which, mathematically, can be determined by solving the following problem,

$$\pi^* = \arg\max_\pi V^\pi(s), \qquad \forall s \in S. \tag{8}$$

The associated value function V^* corresponding to the optimal policy π^* satisfies the Bellman optimality equation [42],

$$\begin{aligned} V^*(s_t) &= \max_{a_t\in A} \mathbb{E}\Big[R_t + \gamma V^*(s_{t+1}) \Big| s_t, \pi\Big] \\ &= \max_{a_t\in A} \sum_{s_{t+1}\in S} \mathbb{P}(s_{t+1}|s_t, a_t)\Big(R'(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\Big). \end{aligned} \tag{9}$$

The state–action value is denoted as $Q^\pi(s_t, a_t)$. In contrast to the state value V^π, the state–action value Q^π emphasizes the value of

the choice of the first action starting at the current state. $Q^\pi(s_t, a_t)$ indicates the expected discounted cumulative reward starting at state s_t, taking action a_t and then following policy π, afterwards

$$Q^\pi(s_t, a_t) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \Big| s_t, a_t, \pi\Big]. \tag{10}$$

Similarly with (7), the evaluation equation for the state–action value Q^π is derived as follows,

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}\Big[R_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \Big| s_t, a_t, \pi\Big] \\ &= \sum_{s_{t+1}\in S} \mathbb{P}(s_{t+1}|s_t, a_t)\Big(R'(s_t, a_t, s_{t+1}) \\ &\quad + \gamma \sum_{a_{t+1}\in A} \pi(s_{t+1}, a_{t+1}) Q^\pi(s_{t+1}, a_{t+1})\Big). \end{aligned} \tag{11}$$

The state value V^π in (5) and the state–action value Q^π in (10) are related and they satisfy the Bellman equation as follows,

$$V^\pi(s_t) = \sum_{a_t\in A} \pi(s_t, a_t) Q^\pi(s_t, a_t). \tag{12}$$

The optimal policy π^* satisfying (8) also satisfies the following equation, which means that one can determine π^* by either solving (8) or, equivalently, by solving (13),

$$\pi^* = \arg\max_\pi \; Q^\pi(s, a), \qquad \forall (s, a) \in S \times A. \tag{13}$$

The optimal state–action value Q^* corresponding to the optimal policy π^* satisfies the following Bellman optimality equation,

$$\begin{aligned} Q^*(s_t, a_t) &= \mathbb{E}\Big[R_t + \gamma \max_{a_{t+1}\in A} Q^*(s_{t+1}, a_{t+1}) \Big| s_t, a_t, \pi\Big] \\ &= \sum_{s_{t+1}\in S} \mathbb{P}(s_{t+1}|s_t, a_t) \\ &\quad \times \Big(R'(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}\in A} Q^*(s_{t+1}, a_{t+1})\Big). \end{aligned} \tag{14}$$

The above definitions and equations in (5)–(14) are the basis for understanding most reinforcement learning algorithms such as value iteration, policy iteration, Sarsa and Q-learning. The Q-learning algorithm works directly on the Q values. The update law of the Q values can be expressed as follows [42,64],

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(R_t + \gamma \max_{a_{t+1}\in A} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big), \tag{15}$$

where $\alpha \in [0, 1]$ is the learning rate (step size), which determines how much the newly acquired information overrides the current Q values. If $\alpha = 0$ one learns nothing since $Q(s_t, a_t)$ remains the same. If $\alpha = 1$ one abandons the old Q value and keeps only the newly learned value $R_t + \gamma \max_{a_{t+1}\in A} Q(s_{t+1}, a_{t+1})$. In particular, if $\alpha = \alpha(t)$ is time-varying and equals to $1/(t+2)$, where $t+2$ represents the total number of visits to the state–action pair (s_t, a_t), one obtains the sample-average result for all observed Q values. The well-known conditions on α to guarantee convergence of the Q values with probability 1 are given as follows [42],

$$\sum_{t=0}^{\infty} \alpha(t) = \infty, \qquad \sum_{t=0}^{\infty} \alpha^2(t) < \infty, \tag{16}$$

where the first condition in (16) is used to overcome the random fluctuations, and the second condition in (16) guarantees convergence as $t \to \infty$. However, the conditions in (16) are seldom used in practice. For instance, a constant step size $\alpha \neq 0$ does not satisfy (16), but it has been shown to perform well in many problems. In this paper we also use a constant step size (i.e., $\alpha = 0.75$) in all the examples.

The discount rate $\gamma \in [0, 1)$ in (15) describes the importance of future rewards for the agent. Specifically, $\gamma = 0$ indicates that the agent only considers the immediate reward after taking action a_t, and hence the agent is "myopic". The agent becomes more "far-sighted" as γ approaches 1, since more cumulative future rewards are taken into account to update the Q values. A value of $\gamma \geq 1$ may lead to divergence.

We summarize the Q-learning algorithm in Algorithm 1. In Section 6 we design the weight vector w in the reward function (4), and the values of the parameters α, γ and ϵ, in Algorithm 1 to learn the optimal policy π^*.

Algorithm 1: Q-Learning Algorithm.

Input: $S, A, \alpha, \gamma, \epsilon, R$
Output: Q^*, π^*

$Q \leftarrow Q_0$
$Q(s_{\text{final}}, \cdot) \leftarrow 0$
Converge $\leftarrow$ False
while not Converge **do**
  $s \leftarrow s_0$
  EpisodeOver $\leftarrow$ False
  while not EpisodeOver **do**
    $a \leftarrow \max_{a\in A} Q(s, a)$ (i.e., ϵ-greedy)
    $s' \leftarrow$ state after taking action a
    if $s' \in s_{\text{final}}$ **then**
      EpisodeOver $\leftarrow$ True
    else
      $Q(s, a) \leftarrow Q(s, a) + \alpha\Big(R(s, a) + \gamma \max_{a\in A} Q(s', a) - Q(s, a)\Big)$
    $s \leftarrow s'$
  if Q converges **then**
    Converge $\leftarrow$ True
$Q^* \leftarrow Q$
$\pi^*(s) = \max_{a\in A} Q^*(s, a)$

4. Maximum entropy principle

In Section 3 the driver's reward was designed and we learned the desired driving policy using reinforcement learning. Nonetheless, in some cases one may have little knowledge about the reward function, and it is hard to design the required reward function to achieve the desired driving behavior. The approach in Section 3 is not convenient to use if the prior knowledge of the reward function is not sufficient. However, one can avoid designing the reward function and directly learn the optimal driving policy from demonstrations performed by an expert driver. This type of problem is called inverse reinforcement learning or inverse optimal control [47,73].

Before proceeding with the discussion on inverse reinforcement learning, we introduce the maximum-entropy principle, and explain how this principle can be used to recover the unknown reward function, which can then be used to learn the driving policy using the given demonstrations.

4.1. Maximum entropy principle

The maximum entropy principle was first introduced by Jaynes [74], and since then it has been used in many areas of computer science and statistical learning. In the basic maximum entropy formulation, one is given a set of samples from a target distribution and a set of constraints on this distribution, and then

one estimates this distribution using the maximum entropy distribution that satisfies these constraints [75]. Mathematically, the idea can be demonstrated as the following theorem.

Theorem 4.1. *Suppose $x_i \in \mathscr{X}$, $i = 1, \ldots, n$ are independent and identically distributed (i.i.d) samples from a certain distribution $x_i \sim p^*$. Let*

$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} f_j(x_i), \qquad j = 1, \ldots, m, \tag{17}$$

where $f_j : \mathscr{X} \to \mathbb{R}$ are real-valued functions and $\hat{\mu}_j$ are the empirical expectations of f_j. The maximum entropy estimate $\hat{p}$ of the distribution p^* satisfies

$$\hat{p} = \arg\max_{p} \int_{\mathscr{X}} -p(x)\log p(x)\nu(dx), \tag{18a}$$

$$\text{subject to} \quad \mathbb{E}[f_j(x)] = \int_{\mathscr{X}} p(x) f_j(x)\nu(dx) = \hat{\mu}_j, \tag{18b}$$

$$\int_{\mathscr{X}} p(x)\nu(dx) = 1, \quad j = 1, \ldots, m, \tag{18c}$$

where ν is a base measure. The solution $\hat{p}$ of (18) is given by

$$\hat{p}(x) = \frac{1}{Z(\theta)} e^{\sum_{j=1}^{m}\theta_j f_j(x)}, \tag{19}$$

where $Z(\theta)$ is the partition function having the following form

$$Z(\theta) = \int_{\mathscr{X}} e^{\sum_{j=1}^{m}\theta_j f_j(x)}\nu(dx), \tag{20}$$

and where $\theta_j \in \mathbb{R}$ are parameters satisfying the following equations,

$$\int_{\mathscr{X}} \frac{f_j(x)}{Z(\theta)} e^{\sum_{k=1}^{m}\theta_k f_k(x)}\nu(dx) = \hat{\mu}_j. \tag{21}$$

The maximum entropy principle finds a distribution satisfying the constraints with the largest remaining uncertainty, so that one does not introduce any additional assumptions or biases into the computation of $\hat{p}$.

In inverse reinforcement learning problems, one is given a number of time histories of the agent's behaviors consisting the past states and actions. These past states and actions are usually called demonstrations. Ziebart [45] first applied the maximum entropy principle to solve inverse reinforcement learning problems, for cases where the reward function depends only on the current state, and it was represented via a linear combination of feature functions, namely,

$$R(s) = \sum_i w_i\phi_i(s) = w^\top \Phi(s), \tag{22}$$

where w and $\Phi(s)$ are the weight and feature vectors, respectively. Note that in [45] the feature vector $\Phi(s)$ is a function of state s only, and the actions were not considered. The probability of a demonstration $\zeta \triangleq \{s_0, a_0, \ldots, s_T, a_T\}$ over all paths of duration T is calculated following Theorem 4.1 [45],

$$\mathbb{P}(\zeta|w) = \frac{1}{Z(w)} e^{\sum_{s\in\zeta} w^\top \Phi(s)}, \tag{23a}$$

$$\mathbb{P}(\zeta|w) = \frac{1}{Z(w)} e^{\sum_{s\in\zeta} w^\top \Phi(s)} \prod_{(s,a,s')\in\zeta} \mathbb{P}(s'|s,a), \tag{23b}$$

where the partition function $Z(w)$ is a normalization constant, and (23a) and (23b) provide the solutions corresponding to a deterministic MDP, where the future state can be uniquely determined with the given action at the present state, and a stochastic MDP, where the future state is unpredictable with the given action at the present state, respectively.

Note that in order to simplify the expressions, in the following we use the notations $s \in \zeta$, $(s,a) \in \zeta$ and $(s,a,s') \in \zeta$ to denote the cases when the state s, the state–action pair (s,a) and the state–action–state triple (s,a,s') are demonstrated in ζ, respectively. Either equation in (23) presents a distribution over paths (demonstrations) and indicates that the probability of a path is proportional to the exponential of its total reward, which implies that the paths having higher rewards are more preferable by the agent.

The goal of an inverse reinforcement learning problem is to find the optimal weight w^*, such that the likelihood of the observed demonstrations is maximal under the distribution in (23). In the following, we formulate the inverse reinforcement learning problem using different reward structures. The necessary derivations are provided for IRL problems satisfying the following requirements: (1) Instead of using the state reward $R(s)$ and the state feature $\Phi(s)$ as in [45,49], we use $R(s,a)$ and $\Phi(s,a)$, and (2) We focus only on the stochastic MDP. The demonstrations are required to start from the same state s_0 and are observed over the same time horizon ranging from $t = 0$ to $t = T$. We use the following notation: $\mathscr{D}$ denotes the set of demonstrations, N denotes the number of demonstrations in $\mathscr{D}$, $\Omega \supseteq \mathscr{D}$ denotes the complete path space, and Φ_ζ denotes the feature counts along the path $\zeta \in \mathscr{D}$ which is given by $\Phi_\zeta = \sum_{(s,a)\in\zeta} \Phi(s,a)$.

4.2. Nonparameterized features

The (nonparameterized) features $\Phi(s,a)$ are functions of only the states and actions. One may then consider reward functions as a linear combination of features in the following form

$$R(w; s, a) = w^\top \Phi(s,a). \tag{24}$$

In order to explicitly show the dependency of the reward function on the unknown weight vector w, we use the notation $R(w; s, a)$ instead of $R(s,a)$ in (24). It follows from [74] that maximizing the entropy of the distribution over Ω subject to the feature constraints from observations $\mathscr{D}$ implies the maximization of the likelihood of $\mathscr{D}$ under the maximum entropy distribution in (23b), that is,

$$\begin{aligned} w^* &= \arg\max_w \mathscr{L}_{\mathrm{D}}(w) = \arg\max_w \frac{1}{N}\sum_{\zeta\in\mathscr{D}} \log \mathbb{P}(\zeta|w) \\ &= \arg\max_w \frac{1}{N}\Big(\sum_{\zeta\in\mathscr{D}}\Big(w^\top\Phi_\zeta + \sum_{(s,a,s')\in\zeta}\mathbb{P}(s'|s,a)\Big)\Big) - \log Z(w). \end{aligned} \tag{25}$$

In order to use gradient-based optimization methods to solve the problem in (25), we take the partial derivative of $\mathscr{L}_{\mathrm{D}}$ with respect to the partial derivative of w, to obtain

$$\begin{aligned} \frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial w} &= \frac{1}{N}\sum_{\zeta\in\mathscr{D}}\Phi_\zeta - \frac{1}{Z(w)}\sum_{\zeta\in\Omega}\Phi_\zeta e^{w^\top\Phi_\zeta}\prod_{(s,a,s')\in\zeta}\mathbb{P}(s'|s,a) \\ &= \tilde{\Phi} - \sum_{\zeta\in\Omega}\mathbb{P}(\zeta|w)\Phi_\zeta = \tilde{\Phi} - \mathbb{E}[\Phi_\zeta], \end{aligned} \tag{26}$$

where $\tilde{\Phi} \triangleq \frac{1}{N}\sum_{\zeta\in\mathscr{D}}\Phi_\zeta$ is the expected empirical feature count, and the expectation of the feature count Φ_ζ can be calculated by

$$\mathbb{E}[\Phi_\zeta] = \sum_{s\in S}\sum_{a\in A}\mathbb{E}[\mu(s,a)]\Phi(s,a), \tag{27}$$

and where the term $\mathbb{E}[\mu(s,a)]$ denotes the expected state–action pair visitation counts. One may refer to Algorithm 3 in Section 5.2.2 for the calculation of $\mathbb{E}[\mu(s,a)]$.

4.3. Parameterized features

The use of nonparameterized features in Section 4.2 requires one to design the features manually, which may be a difficult task, in general, since it may not always be possible to approximate a certain unknown reward function having a complicated form. Hence, we consider the use of parameterized features instead of nonparameterized features, so that one can refine the feature design by optimizing the parameters of the features.

In order to formulate the maximum entropy IRL problem, we still consider a reward function given as a linear combination of the features, but the features now depend explicitly on a parameter vector θ,

$$R(w, \theta; s, a) = w^\intercal \Phi(\theta; s, a). \tag{28}$$

Instead of tuning only the weight vector w in (25), we also tune the vector θ associated with w to maximize the likelihood $\mathscr{L}_\mathrm{D}$ as follows,

$$\begin{aligned} w^*, \theta^* &= \arg\max_{w,\theta} \mathscr{L}_\mathrm{D}(w, \theta) = \arg\max_{w,\theta} \frac{1}{N} \sum_{\zeta \in \mathscr{D}} \log \mathbb{P}(\zeta \,|\, w, \theta) \\ &= \arg\max_{w,\theta} \frac{1}{N} \Big(\sum_{\zeta \in \mathscr{D}} \Big(w^\intercal \Phi_\zeta(\theta) + \sum_{(s,a,s') \in \zeta} \mathbb{P}(s' | s, a) \Big) \Big) \\ &\quad - \log Z(w, \theta). \end{aligned} \tag{29}$$

The derivations of $\partial \mathscr{L}_\mathrm{D} / \partial w$ are similar with (26)–(27) and hence are omitted. Thus, we only show the derivation of $\partial \mathscr{L}_\mathrm{D} / \partial \theta$, which following the chain rule yields

$$\frac{\partial \mathscr{L}_\mathrm{D}(w, \theta)}{\partial \theta} = \sum_{s \in S} \sum_{a \in A} \frac{\partial \mathscr{L}_\mathrm{D}(w, \theta)}{\partial R(w, \theta; s, a)} \frac{\partial R(w, \theta; s, a)}{\partial \theta}, \tag{30}$$

where

$$\begin{aligned} \frac{\partial \mathscr{L}_\mathrm{D}(w, \theta)}{\partial R(w, \theta; s, a)} &= \frac{\frac{1}{N} \sum_{\zeta \in \mathscr{D}} \sum_{(\hat{s}, \hat{a}) \in \zeta} R(w, \theta; \hat{s}, \hat{a})}{\partial R(w, \theta; s, a)} \\ &\quad - \frac{1}{Z(w, \theta)} \sum_{\zeta \in \Omega} e^{w^\intercal \Phi_\zeta} \prod_{(\hat{s}, \hat{a}, \hat{s}') \in \zeta} \mathbb{P}(\hat{s}' | \hat{s}, \hat{a}) \\ &\quad \times \frac{\partial \sum_{(\hat{s}, \hat{a}) \in \zeta} R(w, \theta; \hat{s}, \hat{a})}{\partial R(w, \theta; s, a)} \\ &= \mu_\mathrm{D}(s, a) - \sum_{\zeta \in \Omega} \mathbb{P}(\zeta \,|\, w, \theta) \frac{\partial \sum_{(\hat{s}, \hat{a}) \in \zeta} R(w, \theta; \hat{s}, \hat{a})}{\partial R(w, \theta; s, a)} \\ &= \mu_\mathrm{D}(s, a) - \mathbb{E}[\mu(s, a)], \end{aligned} \tag{31}$$

and where $\mu_\mathrm{D}(s, a)$ is the expected empirical state–action pair visitation counts over the demonstrations $\mathscr{D}$. The expression $\partial R(w, \theta; s, a) / \partial \theta$ in (30) is given by

$$\frac{\partial R(w, \theta; s, a)}{\partial \theta} = \frac{\partial w^\intercal \Phi(\theta; s, a)}{\partial \theta} = w^\intercal \frac{\partial \Phi(\theta; s, a)}{\partial \theta}, \tag{32}$$

where the (i, j) entry of the matrix $\partial \Phi(\theta; s, a) / \partial \theta$ is defined as follows

$$\left[\frac{\partial \Phi(\theta; s, a)}{\partial \theta} \right]_{i,j} = \frac{\partial \phi_i(\theta; s, a)}{\partial \theta_j}, \tag{33}$$

and where $\phi_i(\theta; s, a)$ is the ith element of $\Phi(\theta; s, a)$.

Eqs.(30)–(33) provide the necessary ingredients to improve the design of the feature functions by tuning their parameters. One interesting application of these results is to use a DNN having multiple outputs to represent the features (see Fig. 7). In Fig. 7 the reward $R(s_\mathrm{t}, a_\mathrm{t})$ is given by a linear combination of features represented by a DNN, where the inputs s_t^i and a_t^j of the DNN represent the ith and jth elements of the n-dimension state vector and the m-dimension action vector, respectively. One can then calculate

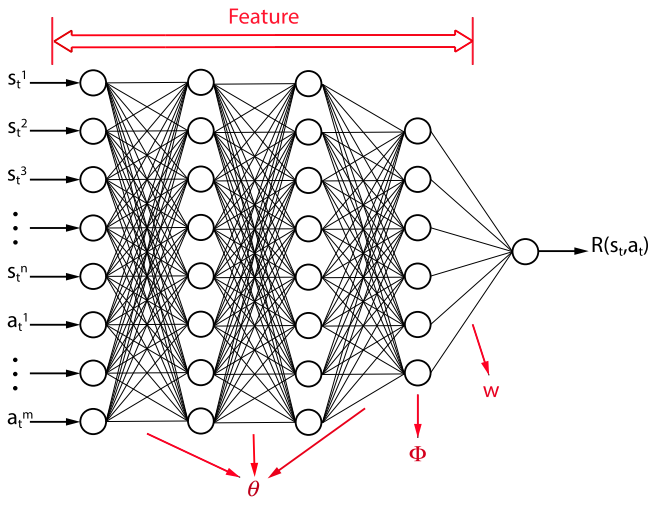

Fig. 7. Deep neural-network feature function and reward.

$\partial R(w, \theta; s, a) / \partial \theta$ by back propagating the network following the delta rule [76].

It is worth mentioning that one can let $w = 1$ and directly use a single DNN feature to represent the reward function. In this case the maximum entropy distribution over the path space Ω is obtained by tuning only the parameters θ in the DNN. For instance, Wulfmeier [49] used a DNN to express the state reward $R(s)$ and determined the maximum entropy distribution of the path ζ by training a DNN.

5. Inverse reinforcement learning

Based on the theoretical derivation in Section 4, we next summarize the inverse reinforcement learning algorithm, which was used in this work to learn the optimal policy from driving demonstrations.

One can use a DNN as a parameterized reward function in order to apply the maximum entropy principle to solve the inverse reinforcement learning problem. In the following, we first discuss the structure of the DNN reward function, and next, we introduce two new IRL algorithms to learn the unknown parameters of the DNN.

5.1. Reward approximator

In order to recover the unknown reward function from demonstrations, we use a universal approximator that has the ability to represent any function, such as a DNN with multiple layers [77,78].

We consider three definitions of the reward functions from the literature, namely, the state reward $R : S \to \mathbb{R}$ [44,45,48,49], the state–action–reward $R : S \times A \to \mathbb{R}$ [19,42,47] and the state–action–state reward $R : S \times A \times S \to \mathbb{R}$ [42]. We denote the corresponding reward functions as $R(s)$, $R(s, a)$ and $R(s, a, s')$, respectively. $R(s)$ is used when the agent wants to reach a goal state or avoid certain dangerous states by taking any action. This definition indicates that the agent has no specific preference over the existing actions. In contrast, $R(s, a)$ takes the action into consideration, so that it can be used to show the agent's preference on a specific action. The last definition $R(s, a, s')$ takes into consideration also the resulting state s' after the agent takes action a at the present state s. Nevertheless, since the resulting state s' depends on the response of the environment after the agent takes action a, the agent can only make a decision according to the expected reward of

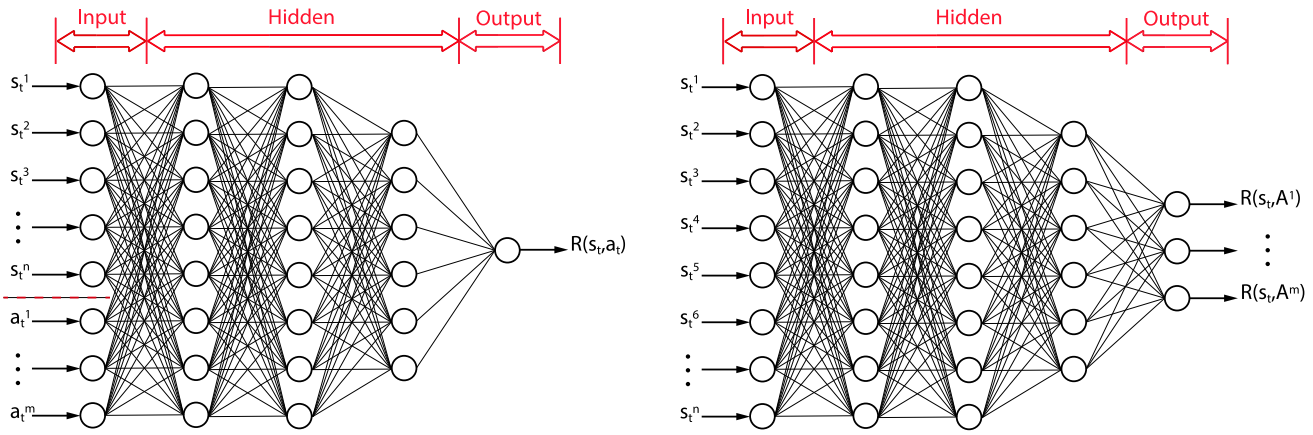

Fig. 8. Structures of the deep neural-network reward functions.

taking action a without knowing the future state s'. Thus $R(s, a, s')$ and $R(s, a)$ are expected to be equivalent in terms of learning the same policy. See also Eq. (6).

According to the above discussion, we will use the state–action–reward $R(s, a)$ in order to reproduce driving behaviors having different preference on the available actions.

The two structures of the DNNs we have in mind are shown in Fig. 8. In the first structure we use both the state s_t and the action a_t as the input, where s_t^i and a_t^j represent the ith and jth elements of the n-dimensional state vector and the m-dimensional action vector at time step t, respectively. In the second structure we use only the state s_t as the input to the network, and use different channels of the output to represent the reward corresponding to different actions in the action set A. The output $R(s_t, A^j)$, $j = 1, \ldots, 5$ represents the reward received by the agent by taking action A^j at present state s_t, where A^j represents the jth action in the action set A. The present action a_t is not an explicit input for the second structure. Both of these two DNN structures in Fig. 8 can be used as an approximator for the state–action–reward function, and one can take either form that is most convenient for the learning task at hand.

5.2. MaxEnt deep IRL algorithm

We summarize the IRL algorithm used to learn the unknown parameters of the DNN in Fig. 8, using the maximum entropy principle based on the results in Section 4. Since the calculation of the expected state–action visitation number requires knowledge of the model, we first discuss the model learning.

5.2.1. Model learning

We consider a totally model-free case, where no knowledge about the state transition model $\mathbb{P}(s'|s, a)$ is available. The idea of model learning is that one can analyze the visitation count of each state–action–state triple and calculate the probability for each possible result of the state transitions, which is given by

$$\mathbb{P}(s'|s, a) = \frac{\nu(s, a, s')}{\sum_{s' \in S} \nu(s, a, s')}, \tag{34}$$

where $\nu(s, a, s')$ is the total number of the state transition from s to s' by taking action a. The probability $\mathbb{P}(s'|s, a)$ approaches its actual value as the state visitation count $\nu(s, a, s')$ approaches infinity. Model learning can be implemented along with the Q-learning (see Algorithm 1). The algorithm for Q-learning with model learning is summarized in Algorithm 2.

Algorithm 2: Q-Learning with Model Learning.

Input: $S, A, \alpha, \gamma, \epsilon, R, \nu_0$
Output: $Q^*, \pi^*, \nu, \mathbb{P}$

$\nu \leftarrow \nu_0$
$Q(s_0, a_0) \leftarrow Q_0$
$Q(s_{\text{final}}, \cdot) \leftarrow 0$
Converge $\leftarrow$ False
while not Converge **do**
  $s \leftarrow s_0$
  EpisodeOver $\leftarrow$ False
  while not EpisodeOver **do**
    $a \leftarrow \max_{a \in A} Q(s, a)$ (i.e., ϵ-greedy)
    $s' \leftarrow$ state after taking action a
    $\nu(s, a, s') \leftarrow \nu(s, a, s') + 1$
    if $s' \in s_{\text{final}}$ **then**
      EpisodeOver $\leftarrow$ True
    else
      $Q(s, a) \leftarrow Q(s, a) + \alpha\Big(R(s, a) + \gamma \max_{a \in A} Q(s', a) - Q(s, a)\Big)$
    $s \leftarrow s'$
  if Q converges **then**
    Converge $\leftarrow$ True
$Q^* \leftarrow Q$
$\pi^*(s) = \max_{a \in A} Q^*(s, a)$
$\mathbb{P}(s'|s, a) \leftarrow \frac{\nu(s,a,s')}{\sum_{s' \in S} \nu(s,a,s')}$

5.2.2. IRL algorithm

In this section we summarize the MaxEnt Deep IRL algorithm according to (29)–(33). To this end, we first introduce the following algorithm to calculate the expected state–action visitation counts $\mathbb{E}[\mu(s, a)]$ in (31), using the model learning result $\mathbb{P}(s'|s, a)$ from Algorithm 2.

In Algorithm 3 the state s_{final} denotes the terminal state or the goal state, after which the state of the system will not change, meaning that no future state transitions can occur at s_{final}. Next, one calculates the gradient $\partial \mathscr{L}_D / \partial \theta$ using (38), and updates the parameters θ of the DNN using gradient ascent. The expression of $\Delta\theta$ can be calculated as follows,

Algorithm 3: Expected State-Action Visitation Counts.

Input: $T, S, A, \pi(s,a), \mathbb{P}(s|s',a)$
Output: $\mathbb{E}[\mu(s,a)]$
Calculate expected state/state–action visitation counts:
$\mathbb{E}[\mu(s_0)] \leftarrow 1$
$\mathbb{E}[\mu(s_0,a)] \leftarrow \pi(s_0,a)$
for $i = 1 : T$ **do**
  $\mathbb{E}_i[\mu(s_{\text{final}})] \leftarrow 0$
  $\mathbb{E}_i[\mu(s_{\text{final}},a)] \leftarrow 0$
  $\mathbb{E}_{i+1}[\mu(s)] \leftarrow \sum_{s'\in S}\sum_{a\in A}\mathbb{P}(s|s',a)\pi(s',a)\mathbb{E}_i[\mu(s')]$
  $\mathbb{E}_{i+1}[\mu(s,a)] \leftarrow \pi(s,a)\mathbb{E}_{i+1}[\mu(s)]$
$\mathbb{E}[\mu(s,a)] \leftarrow \sum_{i=1}^{T}\mathbb{E}_i[\mu(s,a)]$

$$\Delta\theta = \lambda\frac{\partial\mathscr{L}_{\mathrm{D}}}{\partial\theta}, \tag{35}$$

where λ is the learning rate (time step). One can also introduce a weight decay term as a model regularizer into $\Delta\theta$, such as an L_1 regularizer [45], an L_2 regularizer [49], and other forms of regularizers [50,79]. The regularizers are used to convexify the problem, mitigate overfitting, or introduce other properties to the optimization problem such as monotonicity by adding a self-defined cost term.

The proposed MaxEnt Deep IRL algorithm is summarized in Algorithm 4.

Algorithm 4: MaxEnt Deep IRL Algorithm.

Input: $\mu_{\mathrm{D}}(s,a), T, S, A, \alpha, \beta, \gamma, \lambda, \epsilon, \nu_0$
Output: $\pi^*, \theta^*, R^*, \mathbb{P}$
$\theta \leftarrow \theta_0$
$\nu \leftarrow \nu_0$
Converge $\leftarrow$ False
while not Converge **do**
  Update reward function:
  $R \leftarrow NN(\theta)$
  Update policy:
  $\pi, \nu, \mathbb{P} \leftarrow$ Q-learning with model learning$(S, A, \alpha, \gamma, \epsilon, R, \nu)$ **from** Algorithm 2
  Calculate expected state/state–action visitation counts:
  $\mathbb{E}[\mu(s_0)] \leftarrow 1$
  $\mathbb{E}[\mu(s_0,a)] \leftarrow \pi(s_0,a)$
  for $i = 1 : T$ **do**
    $\mathbb{E}_i[\mu(s_{\text{final}})] \leftarrow 0$
    $\mathbb{E}_i[\mu(s_{\text{final}},a)] \leftarrow 0$
    $\mathbb{E}_{i+1}[\mu(s)] \leftarrow \sum_{s'\in S}\sum_{a\in A}\mathbb{P}(s|s',a)\pi(s',a)\mathbb{E}_i[\mu(s')]$
    $\mathbb{E}_{i+1}[\mu(s,a)] \leftarrow \pi(s,a)\mathbb{E}_{i+1}[\mu(s)]$
  $\mathbb{E}[\mu(s,a)] \leftarrow \sum_{i=1}^{T}\mathbb{E}_i[\mu(s,a)]$
  Determine Maximum-Entropy gradients:
  $\frac{\partial\mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)} \leftarrow \mu_{\mathrm{D}}(s,a) - \mathbb{E}[\mu(s,a)]$
  Update neural-network weights:
  $\frac{\partial R(\theta;s,a)}{\partial\theta} \leftarrow$ backward propagating neural-network
  $\frac{\partial\mathscr{L}_{\mathrm{D}}}{\partial\theta} \leftarrow \frac{\partial\mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)}\cdot\frac{\partial R(\theta;s,a)}{\partial\theta}$
  $\theta \leftarrow \theta + \lambda\frac{\partial\mathscr{L}_{\mathrm{D}}}{\partial\theta} + \beta\theta$ (in case of using an L_2 regularizer)
  if θ converges **then**
    Converge $\leftarrow$ True
    $\theta^* \leftarrow \theta$
$R^* \leftarrow NN(\theta^*)$
$\pi^* \leftarrow$ Q-learning$(S, A, \alpha, \gamma, \epsilon, R^*, \nu)$ **from** Algorithm 2

5.3. IRL algorithm refinement

Learning of the model may not yield good results before one has a large number of visitations for each state–action pair. The error of the state transition probability $\mathbb{P}(s'|s,a)$ may lead to errors in calculating $\partial\mathscr{L}^{\mathrm{D}}/\partial R(s,a)$ in Algorithm 4, which leads to further errors in calculating the gradients $\partial\mathscr{L}^{\mathrm{D}}/\partial\theta$ that are used to update the parameters θ in the neural-network. To address this issue, one can pre-learn the model until it converges before using it in Algorithm 4. Nevertheless, the demonstrations in $\mathscr{D}$ may not be enough to represent the environment's random behavior, especially in the case where the system is complicated and the demonstrations are required to represent the long-term behavior of the stochastic system. One can then either split the demonstrations into small pieces to avoid a large error in predicting the long-term behavior of the system, or, alternatively, try to avoid using the state transition terms in the calculation of the gradients in (26) and (31).

In this work we regenerate a number of new sets of the demonstrations $\mathscr{D}_\tau = \{\zeta_\tau^i, i = 1,\dots,N_\tau\}$ using the original demonstrations $\mathscr{D}$. The element ζ_τ^i satisfies the following conditions: (1) ζ_τ^i starts at the state $\tau \in S$, (2) The length of ζ_τ^i is constant ΔT for all $\tau \in S$, and (3) There exists a path $\zeta \in \mathscr{D}$ such that $\zeta_\tau^i \subseteq \zeta$. The corresponding path space for ζ_τ^i is denoted as Ω_τ. We then maximize the entropy of the joint distribution over all Ω_τ subject to the constraints from the demonstrations $\mathscr{D}_\tau$,

$$\begin{aligned}\theta^* &= \arg\max_\theta \mathscr{L}_{\mathrm{D}}(\theta) = \arg\max_\theta \sum_{\tau\in S}\frac{1}{N_\tau}\sum_{\zeta\in\mathscr{D}_\tau}\log\mathbb{P}(\zeta|\theta)\\ &= \arg\max_\theta \sum_{\tau\in S}\Big(\frac{1}{N_\tau}\Big(\sum_{\zeta\in\mathscr{D}_\tau}\Big(\sum_{(s,a)\in\zeta}R(\theta;s,a)\\ &\quad+\sum_{(s,a,s')\in\zeta}\mathbb{P}(s'|s,a)\Big)\Big) - \log Z_\tau(\theta)\Big),\end{aligned} \tag{36}$$

where the partition function Z_τ is given by

$$Z_\tau(\theta) = \sum_{\zeta\in\Omega_\tau} e^{\sum_{(s,a)\in\zeta}R(\theta;s,a)}. \tag{37}$$

The partial derivative of $\mathscr{L}_{\mathrm{D}}$ with respect to the partial derivative of θ is given by

$$\frac{\partial\mathscr{L}_{\mathrm{D}}(\theta)}{\partial\theta} = \sum_{s\in S}\sum_{a\in A}\frac{\partial\mathscr{L}_{\mathrm{D}}(\theta)}{\partial R(\theta;s,a)}\frac{\partial R(\theta;s,a)}{\partial\theta}, \tag{38}$$

where

$$\begin{aligned}\frac{\partial\mathscr{L}_{\mathrm{D}}(\theta)}{\partial R(\theta;s,a)} &= \sum_{\tau\in S}\Bigg(\frac{\frac{1}{N_\tau}\sum_{\zeta\in\mathscr{D}_\tau}\sum_{(\hat{s},\hat{a})\in\zeta}R(\theta;\hat{s},\hat{a})}{\partial R(\theta;s,a)}\\ &\quad-\frac{1}{Z_\tau(\theta)}\sum_{\zeta\in\Omega_\tau}e^{\sum_{(\hat{s},\hat{a})\in\zeta}R(\theta;\hat{s},\hat{a})}\\ &\quad\times\prod_{(\hat{s},\hat{a},\hat{s}')\in\zeta}\mathbb{P}(\hat{s}'|\hat{a},\hat{s})\frac{\partial\sum_{(\hat{s},\hat{a})\in\zeta}R(\theta;\hat{s},\hat{a})}{\partial R(\theta;s,a)}\Bigg)\\ &= \sum_{\tau\in S}\Bigg(\mu_{\mathrm{D}_\tau}(s,a) - \sum_{\zeta\in\Omega_\tau}\mathbb{P}(\zeta|\theta)\frac{\partial\sum_{(\hat{s},\hat{a})\in\zeta}R(\theta;\hat{s},\hat{a})}{\partial R(\theta;s,a)}\Bigg)\\ &= \sum_{\tau\in S}\big(\mu_{\mathrm{D}_\tau}(s,a) - \mathbb{E}[\mu_\tau(s,a)]\big),\end{aligned} \tag{39}$$

and where μ_{D_τ} is the expected empirical state–action pair visitation counts over the demonstrations $\mathscr{D}_\tau$. The term $\partial R/\partial\theta$ in (38) can be obtained by backward propagating the DNN.

Algorithm 5: Single-Step Joint Maximum-Entropy Deep IRL Algorithm.

Input: $\pi_{\mathrm{D}}(s,a), S, A, \alpha, \beta, \gamma, \lambda, \epsilon, \nu_0$
Output: $\pi^*, \theta^*, R^*, \mathbb{P}$

$\theta \leftarrow \theta_0$
$\nu \leftarrow \nu_0$
Converge $\leftarrow$ False
while not Converge **do**
  Update reward function:
  $R \leftarrow NN(\theta)$
  Update policy:
  $\pi, \nu, \mathbb{P} \leftarrow$ Q-learning with model learning$(S, A, \alpha, \gamma, \epsilon, R, \nu)$ **from** Algorithm 2
  Determine Maximum-Entropy gradients:
  $\frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)} \leftarrow \pi_{\mathrm{D}}(s,a) - \pi(s,a)$
  Update neural-network weights:
  $\frac{\partial R(\theta;s,a)}{\partial \theta} \leftarrow$ backward propagating neural-network
  $\frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial \theta} \leftarrow \frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)} \cdot \frac{\partial R(\theta;s,a)}{\partial \theta}$
  $\theta \leftarrow \theta + \lambda \frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial \theta} + \beta\theta$
  if θ converges **then**
    Converge $\leftarrow$ True
    $\theta^* \leftarrow \theta$
$R^* \leftarrow NN(\theta^*)$
$\pi^*, \mathbb{P} \leftarrow$ Q-learning$(S, A, \alpha, \gamma, \epsilon, R^*, \nu)$ **from** Algorithm 2

The expected state–action pair visitation counts $\mathbb{E}[\mu_\tau(s,a)]$ is calculated over ΔT steps. Specifically, we let $\Delta T = 1$ and consider only the one-step action case, such that we avoid using the unknown model transition probabilities to calculate $\partial \mathscr{L}_{\mathrm{D}}(\theta)/\partial R(\theta; s, a)$ in (39), which is given by

$$\begin{aligned}\frac{\partial \mathscr{L}_{\mathrm{D}}(\theta)}{\partial R(\theta;s,a)} &= \sum_{\tau\in S}\left(\frac{\frac{1}{N_\tau}\sum_{(\hat{s},\hat{a})\in\mathscr{D}_\tau} R(\theta;\hat{s},\hat{a})}{\partial R(\theta;s,a)}\right.\\ &\quad\left. - \frac{1}{Z_\tau(\theta)}\sum_{(\hat{s},\hat{a})\in\Omega_\tau} e^{R(\theta;\hat{s},\hat{a})}\frac{\partial R(\theta;\hat{s},\hat{a})}{\partial R(\theta;s,a)}\right)\\ &= \mu_{\mathrm{D}_{\mathrm{s}}}(s,a) - \mathbb{P}(s,a|\theta) \triangleq \pi_{\mathrm{D}}(s,a) - \pi(s,a), \end{aligned} \tag{40}$$

where $\mu_{\mathrm{D}_{\mathrm{s}}}$ is the expected empirical state–action pair visitation counts over the demonstrations $\mathscr{D}_{\mathrm{s}}$, which, can be defined as the expected empirical policy $\pi_{\mathrm{D}}(s,a)$. The result of (40) indicates that, by using the demonstrations with $\Delta T = 1$, the maximum entropy IRL formulation in (36) learns a reward function $R(\theta; s, a)$ such that the learned policy equals the expected empirical policy, namely, $\pi(s,a) = \pi_{\mathrm{D}}(s,a)$.

We summarize the refined algorithms using (39) and (40), respectively, which are given by Algorithms 5 and 6. As a special case when the data length is $\Delta T = 1$, the single-step IRL algorithm in Algorithm 5 totally avoids calculating the expected state–action visitation counts, and hence the policy is learned without any knowledge of the model. The selection among the three IRL algorithms proposed in this paper depends on the data and the MDP model one has for the problem. For instance, if the behavior of the MDP model is not difficult to predict (i.e., deterministic MDP) and one has collected a long set of data starting from the same initial state, one can use Algorithm 4 to recover the reward function and learn the policy. If the data have different lengths and were collected with different initial states, one may have to reorganize the data and consider using Algorithms 5 and 6, especially for problems having complicated stochastic behavior.

Algorithm 6: Multiple-Step Joint Maximum-Entropy Deep IRL Algorithm.

Input: $\mu_{\mathrm{D_{fi}}}(s,a), \Delta T, S, A, \alpha, \beta, \gamma, \lambda, \epsilon, \nu_0$
Output: $\pi^*, \theta^*, R^*, \mathbb{P}$

$\theta \leftarrow \theta_0$
$\nu \leftarrow \nu_0$
Converge $\leftarrow$ False
while not Converge **do**
  Update reward function:
  $R \leftarrow NN(\theta)$
  Update policy:
  $\pi, \nu, \mathbb{P} \leftarrow$ Q-learning with model learning$(S, A, \alpha, \gamma, \epsilon, R, \nu)$ **from** Algorithm 2
  Calculate expected state/state–action visitation counts:
  for τ in S **do**
    $\mathbb{E}[\mu_\tau(s_0 = \tau)] \leftarrow 1$
    $\mathbb{E}[\mu_\tau(s_0 = \tau, a)] \leftarrow \pi(s_0, a)$
    for $i = 1 : \Delta T$ **do**
      $\mathbb{E}_i[\mu_\tau(s_{\text{final}})] \leftarrow 0$
      $\mathbb{E}_i[\mu_\tau(s_{\text{final}}, a)] \leftarrow 0$
      $\mathbb{E}_{i+1}[\mu_\tau(s)] \leftarrow \sum_{s'\in S}\sum_{a\in A}\mathbb{P}(s|a,s')\pi(s',a)\mathbb{E}_i[\mu_\tau(s')]$
      $\mathbb{E}_{i+1}[\mu_\tau(s,a)] \leftarrow \pi(s,a)\mathbb{E}_{i+1}[\mu_\tau(s)]$
    $\mathbb{E}[\mu_\tau(s,a)] \leftarrow \sum_{i=1}^{\Delta T}\mathbb{E}_i[\mu_\tau(s,a)]$
  Determine Maximum-Entropy gradients:
  $\frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)} \leftarrow \sum_{\tau\in S}\left(\mu_{\mathrm{D}_\tau}(s,a) - \mathbb{E}[\mu_\tau(s,a)]\right)$
  Update neural-network weights:
  $\frac{\partial R(\theta;s,a)}{\partial \theta} \leftarrow$ backward propagating neural-network
  $\frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial \theta} \leftarrow \frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial R(\theta;s,a)} \cdot \frac{\partial R(\theta;s,a)}{\partial \theta}$
  $\theta \leftarrow \theta + \lambda \frac{\partial \mathscr{L}_{\mathrm{D}}}{\partial \theta} + \beta\theta$
  if θ converges **then**
    Converge $\leftarrow$ True
    $\theta^* \leftarrow \theta$
$R^* \leftarrow NN(\theta^*)$
$\pi^* \leftarrow$ Q-learning$(S, A, \alpha, \gamma, \epsilon, R^*, \nu)$ **from** Algorithm 2

6. Results and analysis

In this section we implement the previous RL and IRL algorithms for the traffic model of Section 2 and analyze the results. We first build a traffic simulator that is used to simulate the behaviors of the EVs.

6.1. Traffic simulator

The traffic simulator is developed using Pygame, a free and open source python programming language library that is used to develop multimedia applications (i.e., games). The highway road is constructed using a series of connected straight road and curve road segments, where each segment has five lanes. Since each EV is treated as a point of mass, we do not distinguish the types of the vehicles on the simulator (i.e., truck, sedan). The HV is represented using a green car, and the EVs are represented using the cars in other colors. The simulator only provides the top view from a camera over the HV.

Each EV in the simulator implements a random policy. For each EV, we define a nominal state s^{EV} using all the vehicles (HV and EVs) around it and find the safe actions for this EV not leading to a collision. We then generate a random policy for this EV by assigning a random probability to each action in the safe action set. Algorithm 7 shows the procedures to generate a random policy π^{EV} for an EV in the simulation.

Table 1
The selected features and the weights for reinforcement learning.

$\Phi(s, a)$	w_1	Interpretation	w_2	Interpretation
Maintain	0	NA	0	NA
Accelerate	0.075	Prefer accelerating	0.05	Prefer accelerating
Brake	−0.625	Avoid braking	−0.5	Avoid braking
Left-turn	−0.05	Reduce lane-shifting	−0.025	Reduce lane-shifting
Right-turn	−0.05	Reduce lane-shifting	−0.025	Reduce lane-shifting
HV position	0	NA	0	NA
Overtake	0.05	Prefer inner overtaking	0.025	Prefer inner tailgating
Tailgate	0	NA	0.225	Prefer tailgating
Collision	−0.15	Avoid collision	−0.15	Avoid collision

Algorithm 7: Generate EV policy.

Input: s^{EV}
Output: π^{EV}
1: **Generate five random number between [0,1]:**
2: $\pi_0^{\mathrm{EV}} \leftarrow \mathrm{random}(5)$
3: $\pi^{\mathrm{EV}} \leftarrow \mathrm{sort}(\mathrm{i}_0^{\mathrm{EV}})$
4: **Check availability of each action:**
5: **for** $a \in A$ **do**
6: **if** a is **not** available for $s_{\mathrm{t}}^{\mathrm{EV}}$ **then**
7: $\pi^{\mathrm{EV}}(a|s^{\mathrm{EV}}) \leftarrow 0$
8: $\pi^{\mathrm{EV}} \leftarrow \mathrm{normalize}(\pi^{\mathrm{EV}})$ **over** available actions

For each EV, we only implement Lines 1–3 once to initialize its policy π_0^{EV}. Since the safe action set for each EV is changing along with the simulation process, π_0^{EV} needs to change according to the new driving environment in order to avoid collision. To this end, we update the policy using Lines 4–8 when the safe action set and the state change. By taking the lane-switching actions, both the HV and the EVs are able to visit the 9-cell internal-lane states and the 6-cell boundary states during the simulation.

6.2. Driving behavior from reinforcement learning

We show two different driving behaviors using RL, namely, overtaking and tailgating. To this end, we use the features defined in Section 3.2 and design the weights w_1 and w_2 to achieve the two desired driving behaviors, respectively. The weights are provided in Table 1.

The desired driving behavior by designing w_1 is to show overtaking, which can be described as follows: (1) The HV accelerates to occupy the front cell if it is available; (2) The HV maintains its velocity if there is an EV in front of it and no overtaking is possible; (3) The HV overtakes the front EV if only one side is available for overtaking, by lane-shifting first and then accelerating and maintaining constant speed; (4) The HV overtakes the front EV from the inner side of the corner if both the left and right sides are available for overtaking; (5) The HV does not change lane unless for overtaking; (6) The HV does not brake to occupy the rear cell. (7) No collision is allowed.

The desired driving behavior by designing w_2 is to demonstrate tailgating, which can be described as follows: (1) The HV maintains its velocity if there is an EV in front of it; (2) The HV accelerates to occupy the front cell if it is available and no tailgating will occur by changing lanes; (3) The HV changes lane to tailgate an EV if there is no EV in front of it; (4) The HV prefers to tailgate the vehicle in the lane closer to the inner curb of the road in a corner; (5) The HV does not change lanes unless for tailgating; (6) The HV does not brake to occupy the rear cell; (7) No collision is allowed.

We implemented the Q-learning algorithm (Algorithm 1 or 2) to learn the optimal policies using both w_1 and w_2, with learning rate $\alpha = 0.75$, discount rate $\gamma = 0.5$, and $\epsilon = 8\mathrm{e}^{-2}$ for the ϵ-greedy principle. The EVs' behaviors were generated using Algorithm 7. In order to show the speed of the learning process, we compare the policy during the learning process with the convergent result and define the policy error using the number of states that have different optimal actions. Fig. 9 shows the convergence behaviors of the policies π_1^* and π_2^* corresponding to w_1 and w_2, respectively. One sees that, after 5000 to 6000 episodes the policies π_1 and π_2 get stabilized with the current setup of α, γ and ϵ. In both cases, it takes less than five minutes to obtain the results shown in Fig. 9 on a dual-core 2.27 GHz Intel Xeon processor running 64-bit Windows 10 Enterprise operating system and programmed using Python 3.6.

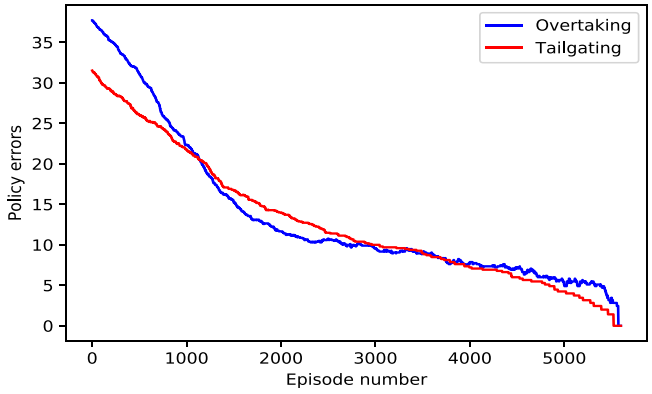

Fig. 9. The convergence performance of the policy π in the learning process.

Next, we implemented the policies π_1^* and π_2^* in simulation. We show only the simulated result from the implementation of π_1^* (overtaking). The tailgating results by implementing the IRL algorithms are given in the next section. Fig. 10 shows four different driving scenarios (each single row of pictures).

The first row of Fig. 10 shows a scenario where there is a vacant space in front of the HV (green). The HV accelerates to occupy the front space and then maintains its distance behind the yellow vehicle. The second row shows a scenario where there is one vehicle in front of the HV and both the left and right lanes are available for the HV to overtake the front vehicle. Since the road is straight, the HV is free to use either the left or the right lane to complete the overtaking task. One sees from this figure that the HV first switches to the left lane, and then accelerates to overtake the front yellow vehicle, which switches to the right lane, until meeting the blue vehicle in the front. The third scenario shows one vehicle (red) in front of the HV but the right lane of the HV is occupied by another vehicle (pink). The HV can only use the left lane to overtake the front vehicle. The last scenario shows another driving scenario during cornering, which has one vehicle (cyan) in front of the HV and both the left and right lanes of the HV are available to use for overtaking. The HV first switches to the right lane, which is closer to the inner curb of the corner, and then tries to overtake the cyan vehicle by accelerating. Overtaking is not completed since the cyan vehicle also moves forward. All these driving behaviors in the simulation using π_1^* agree with the desired behaviors, which validates the design of the reward function and the approach to find the optimal policy.

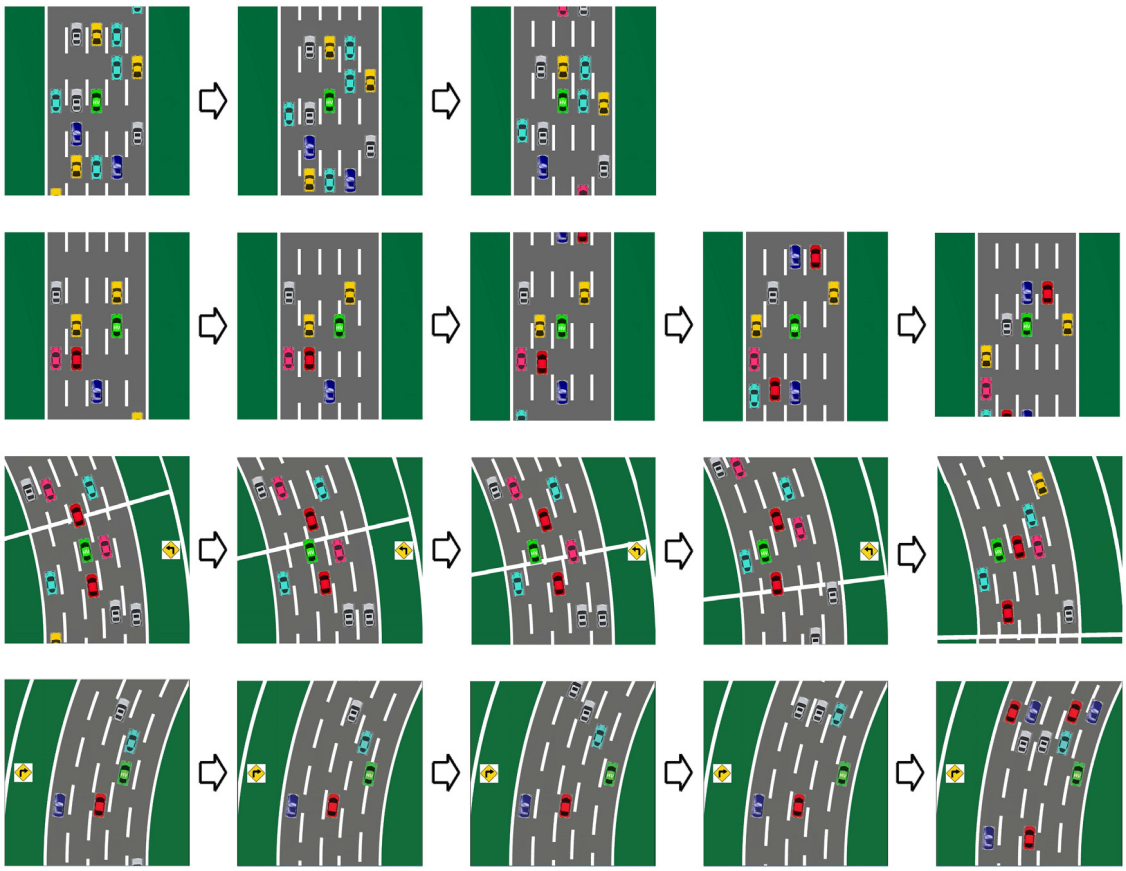

Fig. 10. Overtaking scenarios in simulation by implementing π_1^*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.3. Driving behavior from inverse reinforcement learning

In this section we use the simulated data by implementing π_1^* and π_2^* to learn the reward functions represented using DNNs, and obtain the estimated policies $\hat{\pi}_1^*$ and $\hat{\pi}_2^*$ for π_1^* and π_2^*, respectively, using the maximum entropy principle.

We first select a structure of the DNN as shown in Fig. 8 to represent the reward function. We use the second structure due to its convenience, since it uses only the state s_t as the input of the DNN and provides the rewards for taking every available action in the action set A. One then just needs to select the right output channel according to the current action a_t. The state vector s_t contains the information of the positions of nine vehicles (one HV and eight EVs) and the type of the road, hence, the DNN requires ten input channels to receive the ten-dimensional state s_t. The DNN requires five output channels since there are five actions in A. We define a DNN with the numbers of the neurons in each layer given by $[10, 20, 20, 20, 5]$, which has three hidden layers and each hidden layer has twenty neurons. We use the hyperbolic tangent activation function (tanh) instead of the sigmoid activation function, since the tanh function is unbiased at the origin, and it has a larger range [-1,1] than the sigmoid function which has a range [0,1].

Next, we collect simulated data by implementing the π_1^* and π_2^* learned in Section 6.2 on the traffic simulator. The initial state s_0 is shown with the rectangle zone in Fig. 11, where the HV is located in the middle lane of a five-lane road surrounding by three EVs. For either π_1^* and π_2^*, we collected 500 demonstrations with a fixed simulation period $T = 1500$.

We then implemented the three MaxEnt IRL algorithms in Algorithms 4–6, respectively, to learn the policy using the simulated data. The parameters to setup the algorithms are as follows:

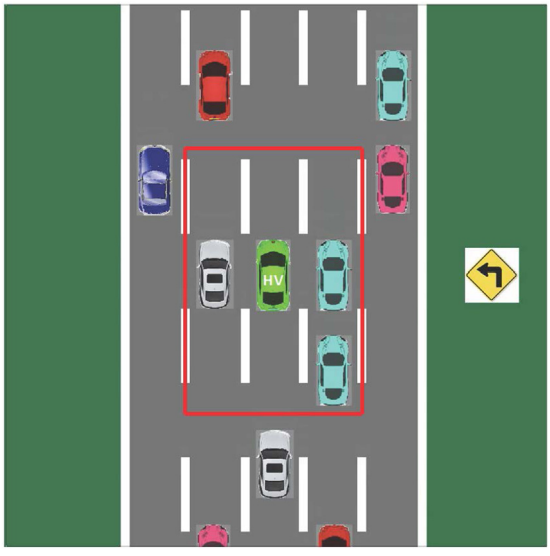

Fig. 11. The initial setup for simulation.

learning rate for Q-learning $\alpha = 0.75$, discount rate $\gamma = 0.5$, DNN learning rate $\lambda = 5e^{-3}$, regularizer coefficient $\beta = -1e^{-4}$, and $\epsilon = 8e^{-2}$ in the ϵ-greedy principle. A summary of the results is shown in Table 2.

One notices from Table 2 that Algorithm 4 does not converge, due to the large data length and the stochastic system behavior. The proposed refined algorithms (Algorithms 5 and 6) provide

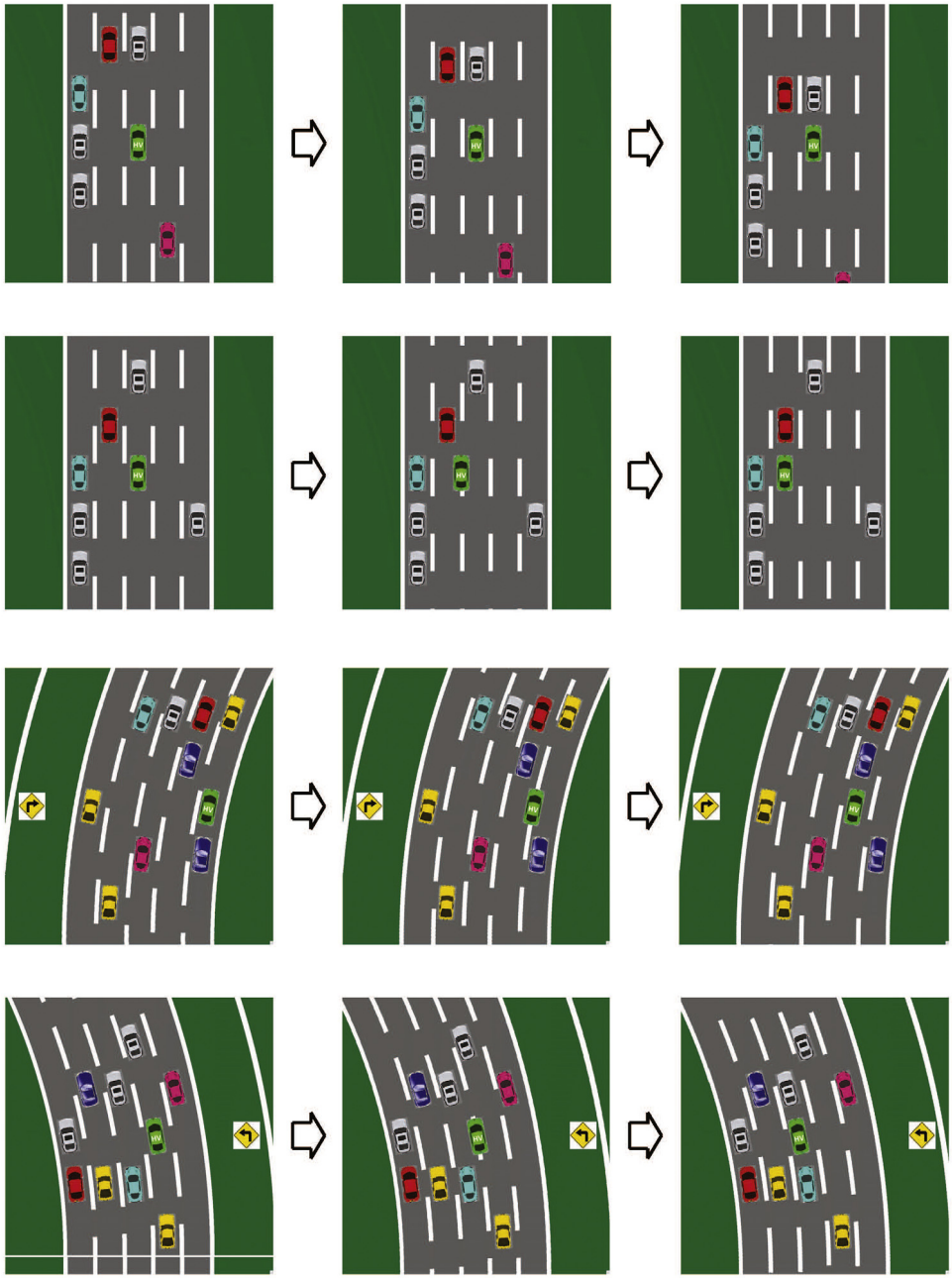

Fig. 12. The tailgating in simulation by implementing $\hat{\pi}_2$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
IRL results summary.

	Data length	Convergence	Time	Policy recovery
Algorithm 4	$T = 1500$	No	NA	NA
Algorithm 5	$\Delta T = 1$	Yes	1 h	≥99%
Algorithm 6	$\Delta T = 5$	Yes	1–3 h	≥99%

better convergence performance and learn the policy effectively. In contrast to Algorithm 6, Algorithm 5 saves some running time since one skips the learning of the model and avoids calculating the expected state–action visitation counts.

We also implemented the learned policies $\hat{\pi}_1^*$ and $\hat{\pi}_2^*$ in simulation.[3] We show only the tailgating behavior by implementing $\hat{\pi}_2^*$ (see Fig. 12), since the result by implementing $\hat{\pi}_1^*$ is similar with π_1^*, which has already been shown in Fig. 10.

Fig. 12 shows four driving scenarios. The first row shows a driving scenario having a vacant space in front of the HV and the HV cannot tailgate any EV by changing lanes. The HV accelerates to occupy the space behind the gray vehicle in front of it. The second driving scenario shows that the HV changes the lane to the left to

[3] Movies for both overtaking and tailgating by implementing $\hat{\pi}_1^*$ and $\hat{\pi}_2^*$ are available at: https://www.youtube.com/watch?v=I3ecd9DXmBQ and https://www.youtube.com/watch?v=lVZcRR-Q2PE.

tailgate the red vehicle. The third driving scenario is similar with the second, but it shows a driving behavior during cornering. The last driving scenario shows that there are two EVs in front of the HV in the neighboring lanes, and the HV can change to either the left or the right lane to tailgate an EV. By designing w_2 we have constructed a policy π_2^* to tailgate the EV closer to the inner curb of the road in a corner. One sees from this simulation that the HV changes to the left lane to tailgate the gray vehicle in a left-turn corner. All these driving behaviors using $\hat{\pi}_2$ agree with the desired tailgating behaviors we want to achieve using π_2^*, which validates the effectiveness of the IRL algorithms proposed in this paper.

7. Conclusion

We use a stochastic Markov decision process to model the traffic, and achieve desired driving behaviors using both reinforcement learning and inverse reinforcement learning. The definition of the state and the MDP traffic model are flexible and can be used to model traffic with any number of lanes and any number of EVs. We also take the road geometry into consideration such that the driving policy may change depending on the road curvature. Although the definition of the state is easily scalable and the MDP problem can be solved efficiently, this model does not distinguish different vehicle velocities and it treats each vehicle as a point of mass. Additional work will be required in order to apply the result of this paper in a real-world driving scenario. For instance, one may dynamically change the size of the MDP state according to the (relative) velocities of the vehicles in traffic.

By designing the driver's reward function, we are able to show typical driving behaviors such as overtaking and tailgating, using the Q-learning algorithm to learn the corresponding optimal policies. We have demonstrated these policies using a road with five lanes and with each EV implementing a random policy. In order to be able to recover the policy and the reward function from data, we propose three new model-free inverse reinforcement learning algorithms based on the maximum entropy principle. Instead of using a state reward as in most of the existing literature [44,45,49], we use a state–action–reward, which is capable for the design of more diverse driving behaviors. This is the first work to generalize the formulation of the maximum entropy inverse reinforcement learning problem with any parameterized, continuously differentiable function approximators (i.e., a DNN). In order to refine the inverse reinforcement learning algorithm, we show that long demonstrations are hard to use for this problem if one has limited knowledge of the (stochastic) system behavior. The error stems from two factors: First, the capacity of the demonstrated data may not be enough to represent the stochastic behavior of the system. Second, the prediction error for a stochastic system is accumulated and becomes large for long term prediction horizons in a model-free problem. We refine our inverse reinforcement learning algorithm by maximizing the entropy of the joint distribution over short data pieces. The proposed algorithms are validated in simulation.

Future work will focus on designing the necessary controls to achieve the driving behavior in a high-fidelity simulation or a real driving task. Since the environment state may not be perfectly observed by the agent, one may consider to use a partially observable MDP for decision making under uncertainty of the true environment state. Other possible extensions introduce multiple agents incorporated into the MDP traffic model to coordinate multiple vehicles simultaneously to better control the traffic flow (i.e., traffic congestion mitigation).

Acknowledgments

This work is supported by the Ford Motor Company, USA and National Science Foundation, USA award CPS-1544814.

References

[1] NHTSA, Traffic Safety Facts 2015: A Compilation of Motor Vehicle Crash Data from the Fatality Analysis Reporting System and the General Estimates System, Tech. Rep. DOT HS 812 384, Department of Transportation, National Highway Traffic Safety Administration, Washington, DC, USA, 2015.
[2] NHTSA, et al., 2015 motor vehicle crashes: overview, in: Traffic Safety Facts Research Note, vol. 2016, 2016, pp. 1–9.
[3] D. Hendricks, J. Fell, M. Freedman, The Relative Frequency of Unsafe Driving Acts in Serious Traffic Crashes, Report no: DOT-HS-809-206, 2001.
[4] C. You, J. Lu, P. Tsiotras, Nonlinear driver parameter estimation and driver steering behavior analysis for ADAS using field test data, IEEE Trans. Hum.-Mach. Syst. 47 (5) (2017) 686–699.
[5] S.D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y.H. Eng, D. Rus, M.H. Ang, Perception, planning, control, and coordination for autonomous vehicles, Machines 5 (1) (2017) 6.
[6] C. Thorpe, M.H. Hebert, T. Kanade, S.A. Shafer, Vision and navigation for the Carnegie-Mellon NAVLAB, IEEE Trans. Pattern Anal. Mach. Intell. 10 (3) (1988) 362–373.
[7] T. Jochem, D. Pomerleau, B. Kumar, J. Armstrong, PANS: A portable navigation platform, in: Proceedings of the Intelligent Vehicles' 95 Symposium, Detroit, MI, September 25–26 1995, pp. 107–112.
[8] B. Paden, M. Čáp, S.Z. Yong, D. Yershov, E. Frazzoli, A survey of motion planning and control techniques for self-driving urban vehicles, IEEE Trans. Intel. Veh. 1 (1) (2016) 33–55.
[9] A.J. Hawkins, Google's new self-driving minivans will be hitting the road at the end of January 2017, 2017, [Online]. Available: https://www.theverge.com/2017/1/8/14206084/google-waymo-self-driving-chrysler-pacifica-minivan-detroit-2017.
[10] J. Berr, Uber's audacious plan to replace human drivers, 2016, [Online]. Available: https://www.cbsnews.com/news/ubers-audacious-plan-to-replace-human-drivers.
[11] C. Thompson, Tesla just revealed new cars and model 3 will have fully self-driving hardware, 2016, [Online]. Available: http://www.businessinsider.com/tesla-announces-new-autopilot-self-driving-2016-10.
[12] D. Lee, Ford's self-driving car 'coming in 2021', 2016, [Online]. Available: http://www.bbc.com/news/technology-37103159.
[13] V. Carlström, Volvo just launched the world's most ambitious autonomous driving trial in Gothenburg, 2017, [Online]. Available: http://nordic.businessinsider.com/volvo-just-launched-the-worlds-most-ambitious-autonomous-driving-trial-in-gothenburg-2017-1.
[14] Auto Tech, 44 corporations working on autonomous vehicles, 2017, [Online]. Available: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list.
[15] E.A. Wan, R. Van Der Merwe, The unscented Kalman filter for nonlinear estimation, in: Adaptive Systems for Signal Processing, Communications, and Control Symposium, Alberta, Canada, October 1–4, 2000, pp. 153–158.
[16] A. Farina, B. Ristic, D. Benvenuti, Tracking a ballistic target: comparison of several nonlinear filters, IEEE Trans. Aerosp. Electron. Syst. 38 (3) (2002) 854–867.
[17] G. Chowdhary, R. Jategaonkar, Aerodynamic parameter estimation from flight data applying extended and unscented Kalman filter, Aerosp. Sci. Technol. 14 (2) (2010) 106–117.
[18] S. Shalev-Shwartz, N. Ben-Zrihem, A. Cohen, A. Shashua, Long-term planning by short-term prediction, 2016, arXiv preprint arXiv:1602.01580.
[19] S. Brechtel, T. Gindele, R. Dillmann, Probabilistic MDP-behavior planning for cars, in: 14th International IEEE Conference on Intelligent Transportation Systems, ITSC, Washington, DC, October 5–7 2011, pp. 1537–1542.
[20] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, J.P. How, Real-time motion planning with applications to autonomous urban driving, IEEE Trans. Control Syst. Technol. 17 (5) (2009) 1105–1118.
[21] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res. 30 (7) (2011) 846–894.
[22] K. Yi, T. Chung, J. Kim, S. Yi, An investigation into differential braking strategies for vehicle stability control, Proc. Inst. Mech. Eng. D 217 (12) (2003) 1081–1093.
[23] S. Di Cairano, H.E. Tseng, D. Bernardini, A. Bemporad, Vehicle yaw stability control by coordinated active front steering and differential braking in the tire sideslip angles domain, IEEE Trans. Control Syst. Technol. 21 (4) (2013) 1236–1248.
[24] L. De Novellis, A. Sorniotti, P. Gruber, A. Pennycott, Comparison of feedback control techniques for torque-vectoring control of fully electric vehicles, IEEE Trans. Veh. Technol. 63 (8) (2014) 3612–3623.
[25] L. De Novellis, A. Sorniotti, P. Gruber, L. Shead, V. Ivanov, K. Hoepping, Torque vectoring for electric vehicles with individually controlled motors: state-of-the-art and future developments, in: 26th Electric Vehicle Symposium, Los Angeles, CA, May 6–9 2012.
[26] J. Ackermann, T. Bünte, D. Odenthal, Advantages of active steering for vehicle dynamics control, in: Proceedings of the 32nd International Symposium on Automotive Technology and Automation, Vienna, Austria, June 14–18 1999, pp. 263–270.

[27] P. Falcone, F. Borrelli, J. Asgari, H.E. Tseng, D. Hrovat, Predictive active steering control for autonomous vehicle systems, IEEE Trans. Control Syst. Technol. 15 (3) (2007) 566–580.
[28] Y.A. Ghoneim, W.C. Lin, D.M. Sidlosky, H.H. Chen, Y.-K. Chin, Integrated chassis control system to enhance vehicle stability, Int. J. Veh. Des. 23 (1–2) (2000) 124–144.
[29] Y. Kou, Development and Evaluation of Integrated Chassis Control Systems (Ph.D. dissertation), The University of Michigan, 2010.
[30] J.C. McCall, M.M. Trivedi, Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation, IEEE Trans. Intell. Transp. Syst. 7 (1) (2006) 20–37.
[31] S.D. Keen, D.J. Cole, Application of time-variant predictive control to modelling driver steering skill, Veh. Syst. Dynam. 49 (4) (2011) 527–559.
[32] S. Zafeiropoulos, P. Tsiotras, Design of a lane-tracking driver steering assist system and its interaction with a two-point visual driver model, in: American Control Conference, Portland, OR, June 4–6, 2014, pp. 3911–3917.
[33] C. You, P. Tsiotras, Optimal two-point visual driver model and controller development for driver-assist systems for semi-autonomous vehicles, in: American Control Conference, Boston, MA, July 6–8 2016, pp. 5976–5981.
[34] J.-w. Choi, R. Curry, G. Elkaim, Path planning based on Bézier curve for autonomous ground vehicles, in: World Congress on Engineering and Computer Science, San Francisco, CA, October 22–24 2008, pp. 158–166.
[35] J.-w. Choi, R.E. Curry, G.H. Elkaim, Continuous curvature path generation based on Bézier curves for autonomous vehicles, Int. J. Appl. Math. 40 (2) (2010).
[36] T. Shim, G. Adireddy, H. Yuan, Autonomous vehicle collision avoidance system using path planning and model-predictive-control-based active front steering and wheel torque control, Proc. Inst. Mech. Eng. D J. Automob. Eng. 226 (6) (2012) 767–778.
[37] M.A. Mousavi, Z. Heshmati, B. Moshiri, LTV-MPC based path planning of an autonomous vehicle via convex optimization, in: 21st Iranian Conference on Electrical Engineering, ICEE, Mashhad, Iran, May 14–16 2013, pp. 1–7.
[38] S. Ulbrich, M. Maurer, Probabilistic online pomdp decision making for lane changes in fully automated driving, in: 16th International IEEE Conference on Intelligent Transportation Systems, Hague, Netherlands, October 6–9 2013, pp. 2063–2067.
[39] C. Katrakazas, M. Quddus, W.-H. Chen, L. Deka, Real-time motion planning methods for autonomous on-road driving: state-of-the-art and future research directions, Transp. Res. C Emerg. Technol. 60 (2015) 416–442.
[40] T. Hastie, R. Tibshirani, J. Friedman, Overview of supervised learning, in: The Elements of Statistical Learning, Springer, 2009, pp. 9–41.
[41] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[42] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, vol. 1, no. 1, MIT Press, Cambridge, 1998.
[43] S. Lange, M. Riedmiller, A. Voigtlander, Autonomous reinforcement learning on raw visual input data in a real world application, in: International Joint Conference on Neural Networks, IJCNN, Brisbane, QLD, Australia, June 10–15 2012, pp. 1–8.
[44] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the 21st International Conference on Machine Learning, Banff, Canada, July 4–8 2004, p. 1.
[45] B.D. Ziebart, A.L. Maas, J.A. Bagnell, A.K. Dey, Maximum entropy inverse reinforcement learning, in: AAAI, vol. 8, Chicago, IL, 2008, pp. 1433–1438.
[46] B.D. Ziebart, A.L. Maas, J.A. Bagnell, A.K. Dey, Human behavior modeling with maximum entropy inverse optimal control, in: AAAI Spring Symposium: Human Behavior Modeling, 2009, p. 92.
[47] S. Levine, V. Koltun, Continuous inverse optimal control with locally optimal examples, 2012, arXiv preprint arXiv:1206.4617.
[48] K.M. Kitani, B.D. Ziebart, J.A. Bagnell, M. Hebert, Activity forecasting, in: European Conference on Computer Vision, Florence, Italy, October 7–13 2012, pp. 201–214.
[49] M. Wulfmeier, P. Ondruska, I. Posner, Maximum entropy deep inverse reinforcement learning, 2015, arXiv preprint arXiv:1507.04888.
[50] C. Finn, S. Levine, P. Abbeel, Guided cost learning: Deep inverse optimal control via policy optimization, in: International Conference on Machine Learning, New York, NY, June 19–24 2016, pp. 49–58.
[51] M. Ardelt, P. Waldmann, F. Homm, N. Kaempchen, Strategic decision-making process in advanced driver assistance systems, IFAC Proc. Volumes 43 (7) (2010) 566–571.
[52] R. Zheng, C. Liu, Q. Guo, A decision-making method for autonomous vehicles based on simulation and reinforcement learning, in: International Conference on Machine Learning and Cybernetics, vol. 1, Tianjian, China, July 14–17 2013, pp. 362–369.
[53] N. Li, D. Oyler, M. Zhang, Y. Yildiz, A. Girard, I. Kolmanovsky, Hierarchical reasoning game theory based approach for evaluation and testing of autonomous vehicle control systems, in: IEEE 55th Conference on Decision and Control, Las Vegas, NV, December 12–14 2016, pp. 727–733.
[54] D.W. Oyler, Y. Yildiz, A.R. Girard, N.I. Li, I.V. Kolmanovsky, A game theoretical model of traffic with multiple interacting drivers for use in autonomous vehicle development, in: American Control Conference, ACC, 2016, Boston, MA, July 6–8 2016, pp. 1705–1710.
[55] R. Bellman, A Markovian decision process, J. Math. Mech. (1957) 679–684.
[56] P. Brémaud, Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues, vol. 31, Springer Science & Business Media, 2013.
[57] A. Defazio, T. Graepel, A comparison of learning algorithms on the arcade learning environment, 2014, arXiv preprint arXiv:1410.8620.
[58] L. Baird, et al., Residual algorithms: Reinforcement learning with function approximation, in: Proceedings of the 12th International Conference on Machine Learning, Miami, Florida, December 4–7 1995, pp. 30–37.
[59] S.-C. Wang, Artificial neural network, in: Interdisciplinary Computing in Java Programming, Springer, 2003, pp. 81–100.
[60] J.M. Bernardo, A.F. Smith, Bayesian Theory, John Wiley & Sons, Canada, 2001.
[61] J.Q. Shi, T. Choi, Gaussian Process Regression Analysis for Functional Data, CRC Press, 2011.
[62] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000.
[63] S. Levine, Z. Popovic, V. Koltun, Nonlinear inverse reinforcement learning with Gaussian processes, in: Advances in Neural Information Processing Systems, 2011, pp. 19–27.
[64] C.J.C.H. Watkins, Learning from Delayed Rewards (Ph.D. dissertation), King's College, Cambridge, 1989.
[65] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[66] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning. in: AAAI, 2016, pp. 2094–2100.
[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.
[68] H.R. Berenji, Fuzzy Q-learning: a new approach for fuzzy dynamic programming, in: Proceedings of the 3rd IEEE Conference on Fuzzy Systems, Orlando, FL, June 26–29 1994, pp. 486–491.
[69] M.L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Proceedings of the 11th International Conference on Machine Learning, vol. 157, New Brunswick, NJ, July 10–13 1994, pp. 157–163.
[70] J. Hu, M.P. Wellman, Nash Q-learning for general-sum stochastic games, J. Mach. Learn. Res. 4 (Nov) (2003) 1039–1069.
[71] A. Greenwald, K. Hall, R. Serrano, Correlated Q-learning, in: Proceedings of the 12th International Conference on Machine Learning, vol. 3, Washington, DC, August 21–24 2003, pp. 242–249.
[72] M.L. Littman, Friend-or-foe Q-learning in general-sum games, in: Proceedings of the 18th International Conference on Machine Learning, vol. 1, Williamstown, MA, June 28 – July 1 2001, pp. 322–328.
[73] A.Y. Ng, S.J. Russell, Algorithms for inverse reinforcement learning, in: Proceedings of the 17th International Conference on Machine Learning, Stanford, CA, June 29–2000, pp. 663–670.
[74] E.T. Jaynes, Information theory and statistical mechanics, Phys. Rev. 106 (4) (1957) 620–630.
[75] M. Dudık, R.E. Schapire, Maximum entropy distribution estimation with generalized regularization, in: International Conference on Computational Learning Theory, San Diego, CA, June 13–15 2006, pp. 123–138.
[76] R. Hecht-Nielsen et al, Theory of the backpropagation neural network, Neural Netw. 1 (Suppl. 1) (1988) 445–448.
[77] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[78] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (2) (1991) 251–257.
[79] J. Audiffren, M. Valko, A. Lazaric, M. Ghavamzadeh, Maximum entropy semi-supervised inverse reinforcement learning, in: International Joint Conferences on Artificial Intelligence, Buenos Aires, Argentina, July 25–31 2015, pp. 3315–3321.

Changxi You received his B.S. and M.S. from the Department of Automotive Engineering, Tsinghua University of China, and a second M.S. from the Department of Automotive Engineering, RWTH-Aachen University of Germany. He is currently a Ph.D. student at the School of Aerospace Engineering, Georgia Institute of Technology. His current research interests focus on system identification, aggressive driving and control of (semi)autonomous vehicle.

Jianbo Lu received his B.S. degree in mechanical engineering from the Central South University, Changsha, China, and the M.S. degree in mechanical engineering from Arizona State University, and his Ph.D. degree in aeronautics and astronautics from Purdue University. He is currently a Technical Expert in advanced vehicle controls at Ford Motor Company, Dearborn, MI, USA. He holds more than 100 U.S. patents and numerous pending patent applications, and has published more than 70 journal and conference articles. His research interests include automotive controls and sensing, adaptive vehicle systems, driver assistance systems, smart mobility, and semiautonomous and autonomous systems. Dr. Lu received the Henry Ford Technology Reward twice.

Dr. Dimitar P. Filev is a Senior Technical Leader — Intelligent Control & Information Systems, Ford Research & Advanced Engineering. He is conducting research in modeling and control of complex systems, intelligent control, fuzzy and neural systems, and their applications to automotive engineering. He is recipient of the 2008 Norbert Wiener Award of the IEEE SMC Society, the 2007 IFSA Outstanding Industrial Applications Award, and the highest Ford Motor Company corporate awards — he was awarded 5 times with the Henry Ford Technology Award for development and implementation of advanced automotive technologies and he received the 2010 Inaugural Haren Gandhi Research & Innovation Award for his long term research contributions. He is past president of NAFIPS and serves presently as VP for Cybernetics of the IEEE SMC Society. Dr. Filev is a Fellow of IEEE and IFSA. He received his Ph.D. degree in Electrical Engineering from the Czech Technical University in Prague in 1979.

Panagiotis Tsiotras is a Dean's Professor in the School of Aerospace Engineering at the Georgia Institute of Technology (Georgia Tech), and the Director of the Dynamics and Controls Systems Laboratory (DCSL) in the same school as well as an Associate Director of the Institute for Robotics and Intelligent Machines at Georgia Tech. He received his Ph.D. degree in Aeronautics and Astronautics from Purdue University in 1993 and also holds degrees in Mathematics and Mechanical Engineering. He has held visiting research appointments at MIT, JPL, INRIA Rocquencourt, and Mines ParisTech. His research interests include optimal control of nonlinear systems and ground, aerial and space vehicle autonomy.

He has served in the Editorial Boards of the Transactions on Automatic Control, the IEEE Control Systems Magazine, the AIAA Journal of Guidance, Control and Dynamics and the journal Dynamics and Control. He is the recipient of the NSF CAREER award and the Outstanding Aerospace Engineer award from Purdue. He is a Fellow of AIAA, and a Senior Member of IEEE, and a member of the Phi Kappa Phi, Tau Beta Pi, and Sigma Gamma Tau Honor Societies.