IET Intelligent Transport Systems

Special Issue: AI Applications to Intelligent Vehicles for Advancing Intelligent Transport Systems

IET Journals
The Institution of Engineering and Technology

Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data

ISSN 1751-956X
Received on 24th May 2019
Revised 7th October 2019
Accepted on 20th January 2020
E-First on 11th February 2020
doi: 10.1049/iet-its.2019.0317
www.ietdl.org

Jingliang Duan[1], Shengbo Eben Li[1] ✉, Yang Guan[1], Qi Sun[1], Bo Cheng[1]

[1]*School of Vehicle and Mobility, Tsinghua University, Beijing 100084, People's Republic of China*

✉ *E-mail: lisb04@gmail.com*

Abstract: Decision making for self-driving cars is usually tackled by manually encoding rules from drivers' behaviours or imitating drivers' manipulation using supervised learning techniques. Both of them rely on mass driving data to cover all possible driving scenarios. This study presents a hierarchical reinforcement learning method for decision making of self-driving cars, which does not depend on a large amount of labelled driving data. This method comprehensively considers both high-level manoeuvre selection and low-level motion control in both lateral and longitudinal directions. The authors firstly decompose the driving tasks into three manoeuvres, including driving in lane, right lane change and left lane change, and learn the sub-policy for each manoeuvre. Then, a master policy is learned to choose the manoeuvre policy to be executed in the current state. All policies, including master policy and manoeuvre policies, are represented by fully-connected neural networks and trained by using asynchronous parallel reinforcement learners, which builds a mapping from the sensory outputs to driving decisions. Different state spaces and reward functions are designed for each manoeuvre. They apply this method to a highway driving scenario, which demonstrates that it can realise smooth and safe decision making for self-driving cars.

1 Introduction

Autonomous driving is a promising technology to enhance road safety, ease the road congestion, decrease fuel consumption and free human drivers. In the architecture of the self-driving car, decision making is a key component to realise autonomy. To date, most decision-making algorithms can be categorised into two major paradigms: the rule-based method and imitation-based method. The former is usually tackled by manually encoding rules from driver behaviours, whereas the latter is committed to imitate drivers' manipulation using supervised learning techniques. To achieve safe decision making under different traffic conditions, both of them rely on mass naturalistic driving data to cover all possible driving scenarios for the purpose of testing or training [1, 2].

The rule-based method has been widely investigated during the last two decades, which is typically hierarchically structured into manoeuvre selection and path planning [3]. Montemerlo *et al.* [4] adopted finite state machines (FSMs) as a mechanism to select manoeuvre for Junior in DARPA. The FSM possessed 13 states, including lane-keeping, parking lot navigation, etc., which was then used to switch between different driving states under the guidance of manually modelling behaviour rules. Furda and Vlacic [5] combined FSM with multi-criteria decision making for driving manoeuvre execution. The most appropriate manoeuvre was selected by considering the sensor measurements, vehicular communications, traffic rules and a hierarchy of objectives during driving with manually specified weights. Glaser *et al.* [6] presented a vehicle path-planning algorithm that adapted to traffic on a lane-structured infrastructure such as highways. They firstly defined several feasible trajectories expressed by a polynomial with respect to the environment, aiming at minimising the risk of collision. Then they evaluated these trajectories using additional performance indicators such as travel time, traffic rules, consumption and comfort to output one optimal trajectory in the following seconds. Kala and Warwick [7] presented a planning algorithm to make behaviour choice according to the obstacle motion in a relatively unstructured road environment. They modelled the various manoeuvres in an empirical formula and adopted one of them to maximise the separation from other vehicles. Despite the popularity of rule-based methods, manually encoding rules can introduce a high burden for engineers, who need to anticipate what is important for driving and foresee all the necessary rules for safe driving. Therefore, this method is not always feasible due to the highly dynamic, stochastic and sophisticated nature of the traffic environment and the enormous number of driving scenarios [8–10].

The imitation-based method is an effective remedy to eliminate the engineers' burden. This is because the driving decision model can be learned directly by mimicking drivers' manipulation using supervised learning techniques, and no hand-crafted rules are needed [11]. This idea dates back to the late 1980s when Pomerleau built the first end-to-end decision-making system for NAVLAB [12]. This system used a fully-connected network, which took images and a laser range finder as input and steering angles as output. Inspired by this work, Lecun *et al.* [13] trained a convolutional neural network (CNN) to make a remote truck successfully drive on unknown open terrain while avoiding any obstacles such as rocks, trees, ditches, and ponds. The CNN constructed a direct mapping from the pixels of the video cameras to two values which were interpreted directly as a steering wheel angle. With the development of deep learning, Chen *et al.* [14] trained a deep CNN using data recorded from 12 h of human driving in a video game. This CNN mapped an input image to a small number of key perception indicators that directly related to the affordance of a road/traffic state for driving. These methods have not been applied in practice until Bojarski *et al.* (2016) built an end-to-end learning system which could successfully perform lane keeping in traffic on local roads with or without lane markings and on highways. A deep CNN called PilotNet was trained using highway road images from a single front-facing camera only paired with the steering angles generated by a human driving a data-collection car [2]. A shortcoming of the aforementioned works is that their capability is derived from large amounts of hand-labelled training data, which come from cars driven by humans while the system records the images and steering angles. In many real applications, the required naturalistic driving data can be too huge due to various driving scenarios, large amounts of participants and complex traffic. Besides, different human drivers may make completely different decisions in the same situation, which results in an ill-posed problem that is confusing when training a regressor.

To eliminate the demand for labelled driving data, some researches have made attempts to implement reinforcement

learning (RL) on decision making for self-driving cars. The goal of RL is to learn policies for sequential decision problems directly using samples from the emulator or experiment, by optimising a cumulative future reward signal [15]. Compared with the imitation-based method, RL is a self-learning algorithm which allows the self-driving car to optimise its driving performance by trial-and-error without reliance on manually designed rules and human driving data [16–18]. Lillicrap *et al.* [19] presented a deep deterministic policy gradient model to successfully learn deterministic control policies in a TORCS simulator with an RGB image of the current frame as inputs. This algorithm maintained a parameterised policy network, which specified the current policy by deterministically mapping states to a specific action such as acceleration. Mnih *et al.* [20] proposed an asynchronous advantage actor–critic algorithm (A3C) which enabled the vehicle to learn a stochastic policy in TORCS. The A3C used parallel actor–learners to update a shared model which could stabilise the learning process without experience replay. However, decision making for self-driving cars remains a challenge for RL because it requires long decision sequences or complex policies [21]. Specifically, decision making for self-driving cars includes higher-level manoeuvre selection (such as a lane change, driving in lane etc.) and low-level motion control in both lateral and longitudinal directions [22]. However, current works of RL only focus on low-level motion control layer. It is hard to solve driving tasks that require many sequential decisions or complex solutions without considering the high-lever manoeuvre [1].

The main contribution of this paper is to propose a hierarchical RL (H-RL) method for decision making of self-driving cars, which does not depend on a large amount of labelled driving data. This method comprehensively considers both high-level manoeuvre selection and low-level motion control in both lateral and longitudinal directions. We firstly decompose the driving tasks into three manoeuvres, including driving in lane, right lane change and left lane change, and learn the sub-policy for each manoeuvre. Different state spaces and reward functions are designed for each manoeuvre. Then, a master policy is learned to choose the manoeuvre policy to be executed in the current state. All policies including master policy and manoeuvre policies are represented by fully-connected neural networks (NNs) and trained by using an asynchronous parallel RL (APRL) algorithm, which builds a mapping from the sensory outputs to driving decisions.

The rest of this paper is organised as follows: Section 2 proposes the H-RL method for decision making of self-driving cars. Section 3 introduces the design of experiment, state space and return function. Section 4 discusses the training results for each manoeuvre and driving task. Section 5 concludes this paper.

2 Methodology

2.1 Preliminaries

We model the sequential decision-making problem for self-driving as a Markov decision process (MDP) which comprises: a state space $\mathcal{S}$, an action space $\mathcal{A}$, transition dynamics $p(s_{t+1}|s_t, a_t)$ and reward function $r(s_t, a_t): \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ [15]. At each time step t, the vehicle observes a state s_t, takes an action a_t, receives a scalar reward r and then reaches the next state s_{t+1}. A stochastic policy $\pi(a|s): \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is used to select actions, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on measures on $\mathcal{A}$. The reward r is often designed by human experts, which is usually assigned a lower value when the vehicle enters a bad state such as collision, and a higher value for a normal state such as maintaining a safe distance from the preceding car. This process continues until the vehicle reaches a terminal state, such as breaking traffic regulations or arriving at the destination.

The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the sum of discounted future reward from time step t, where $\gamma \in (0, 1]$ is a discount factor that trades off the importance of immediate and future rewards. The goal of RL is to maximise the expected return from each state s_t. In general, the expected return for the following policy π from the state s is defined as $V_\pi(s) = E[R_t|s_t = s]$, often referred to as a value function. Then the RL problem can now be expressed as finding a policy π to maximise the associated value function $V_\pi(s)$ in each state.

This study employs deep NN to approximate both the value function $V_\pi(s; w)$ and stochastic policy $\pi_\theta(s)$ in an actor–critic (AC) architecture, where θ and w are the parameters of these two NNs. The AC architecture consists of two structures: (i) the actor, and (ii) the critic [23, 24]. The actor corresponds to the policy network $\pi_\theta(s)$, mapping states to actions in a probabilistic manner. And the critic corresponds to the value network $V_\pi(s; w)$, mapping states to the expected cumulative future reward. Thus, the critic addresses the problem of prediction, whereas the actor is concerned with choosing control. These problems are separable, but are solved by iteratively solving the Bellman optimality equation based on generalised policy iteration framework [25].

2.2 Hierarchical parallel RL algorithm

2.2.1 H-RL architecture: Previous related RL studies typically adopted an end-to-end network to make decisions for autonomous vehicles. To complete a driving task consisting of various driving behaviours, complicated reward functions need to be designed by human experts. Inspired by the options framework [26, 27], we propose an H-RL architecture to eliminate the burden of engineers by decomposing the driving decision making processes into two levels. The key insight behind our new method is illustrated in Fig. 1. The H-RL decision-making framework consists of two parts: (i) the high-level manoeuvre selection and (ii) low-level motion control. In the manoeuvre selection level, a master policy is used to choose the manoeuvre to be executed in the current state. In the motion control level, the corresponding manoeuvre policy would be activated and output front wheel angle and acceleration commands to the actuators. In the example shown in Fig. 1, the master policy chooses manoeuvre 1 as the current manoeuvre, then the corresponding manoeuvre policy would be activated and output front wheel angle and acceleration commands to the actuators. The combination of multiple manoeuvres can constitute diverse driving tasks, which means that the trained manoeuvre policy can also be applied to other driving tasks. Therefore, hierarchical architecture also has better transferability than end-to-end RL in the field of driving decision.

To learn each policy of the H-RL, we should first decompose the driving task into several driving manoeuvres, such as driving in lane, left/right lane change, etc. It should be noted that a driving manoeuvre may include many similar driving behaviours. For example, the driving-in-lane manoeuvre is a combination of many similar behaviours, including lane-keeping, car-following and free driving. Then, we can learn the sub-policy, which is also called manoeuvre policy, for each manoeuvre driven by independent sub-goals. The reward function for sub-policy learning can be easily designed by considering only the corresponding manoeuvre instead of the whole driving task. Then we learn a master policy to activate certain manoeuvre policy according to the driving task. Although we need to consider the entire driving task while training the master policy, the associated reward function can also be very simple because you do not have to worry about how to control the actuators to achieve each manoeuvre. In addition, if the input to each policy is the whole sensory information, the learning algorithm must determine which parts of the information are relevant. Hence, we propose to design different meaningful indicators as the state representative for different policies.

Furthermore, the motion of the vehicle is controlled jointly by the lateral and longitudinal actuators, and the two types of actuators are relatively independent. This means that the state and reward of the lateral and longitudinal controllers can also be designed separately. Therefore, each manoeuvre in this study contains a steer policy network (SP-Net) and an accelerate policy network (AP-Net), which carry out lateral and longitudinal controls, respectively. Noted that the vehicle dynamics used in this paper still consider the coupling between the lateral and longitudinal controllers. The corresponding value networks of these two control policy networks are called SV-Net and AV-Net, respectively. In other words, SP-Net and SV-Net are concerned with lateral control, while AP-Net and

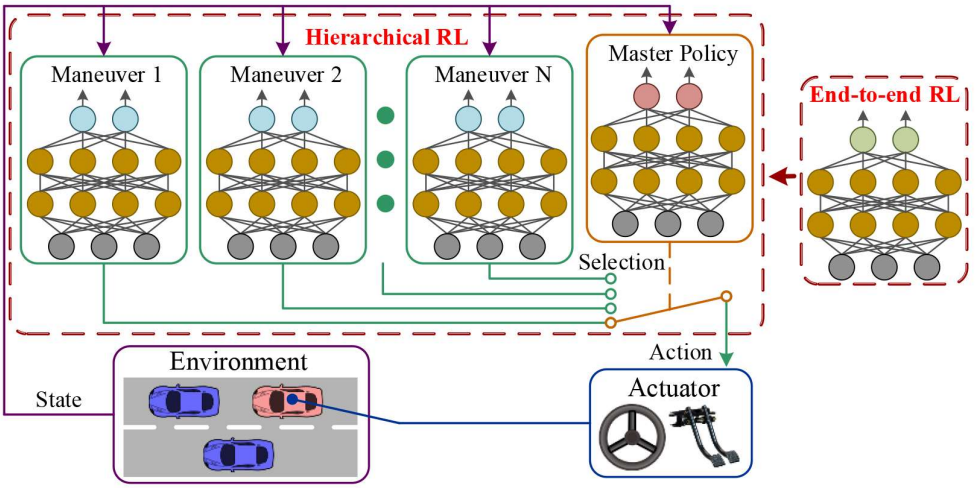

Fig. 1 *Hierarchical RL for self-driving decision-making*

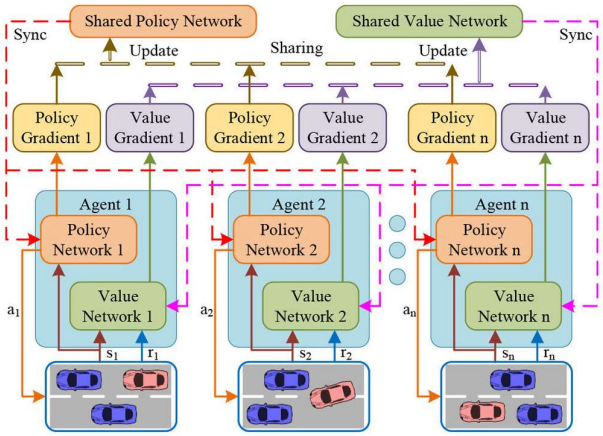

Fig. 2 *Policy training using APRL*

AV-Net are responsible for longitudinal control. On the other hand, the master policy only contains one manoeuvre policy network and one manoeuvre value network. In summary, if a manoeuvre is selected by the master policy, the relevant SP-Net and AP-Net would work simultaneously to apply control.

2.2.2 Parallel training algorithm: Inspired by the A3C algorithm, we use asynchronous parallel car-learners to train the policy $\pi(a|s;\theta)$ and estimate the state value $V(s;w)$, as shown in Fig. 2 [20]. Each car-learner contains its own policy networks and value networks, and makes a decision according to their policy outputs. The multiple car-learners interact with different parts of the environment in parallel. Meanwhile, each car-learner computes gradients with respect to the parameters of the value network and the policy network at each step. Then, the average gradients are applied to update the shared policy network and shared value network at each step. These car-learners synchronise their local network parameters from the shared networks before they make new decisions. We refer to this training algorithm as APRL.

For each car-learner, the parameters w of the value network $V(s;w)$ are tuned by iteratively minimising a sequence of loss functions, where the loss function at time step t is defined as follows:

$$L_t(w) = (R_t - V(s_t;w))^2 \tag{1}$$

where $R_t - V(s_t;w)$ is usually called temporal-difference (TD) error. The expected accumulated return R_t is estimated in the forward view using N-step return instead of full return:

$$R_t = \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n};w) \tag{2}$$

This means that the update gradients for value and policy networks at a time t are estimated after n actions. The specific gradient update for the parameter w of $V(s;w)$ after state s_t is

$$\mathrm{d}w = (R_t - V_w(s_t))\nabla_w V_w(s_t) \tag{3}$$

The policy gradient of the AC method is $(R_t - V_{w_t}(s_t))\nabla_\theta \log \pi(a_t|s_t;\theta)$ [23]. Besides, the entropy of the policy $\sum_a(-\pi_\theta(a_t|s_t)\log \pi_\theta(a_t|s_t))$ is added to the objective function to regularise the policy towards larger entropy, which promotes exploration by discouraging premature convergence to suboptimal policies. In summary, the total gradient consists of the policy gradient term and the entropy regularisation term. Then, the specific gradient update for the parameter θ of the policy network after state s_t is

$$\begin{aligned}\mathrm{d}\theta = {} & (R_t - V_w(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t) \\ & + \beta\nabla_\theta \sum_a (-\pi_\theta(a_t|s_t)\log \pi_\theta(a_t|s_t))\end{aligned} \tag{4}$$

where β is the hyper-parameter that trades off the importance of different loss components. The pseudocode of APRL is presented in Algorithm 1 (see Fig. 3). The parameters of the shared value and policy networks are represented by w and θ, while those of the value and policy networks for specific car-learners are denoted by w' and θ'.

Initialize w, θ, w', θ', step counter $t \leftarrow 0$ and state $s_0 \in S$

repeat

  for each car-learner **do**

    Synchronize car-learner-specific parameters $w' = w$ and $\theta' = \theta$

    Perform a_t according to policy $\pi(a_t|s_t;\theta')$ and store action a_t

    Receive and store reward r_t and the next state s_{t+1}

    $\tau \leftarrow t - n + 1$

    if $\tau \geq 0$ **then**

      $R \leftarrow \sum_{k=\tau}^{min(\tau+N-1,t_{max})} \gamma^{k-\tau} r_k + \gamma^{k-\tau+1} V(s_{t+1};w')$

      Calculate gradients wrt w':

        $dw' = (R - V_{w'}(s_\tau))\nabla_{w'} V_{w'}(s_\tau)$

      Calculate gradients wrt θ':

        $d\theta' = (R - V_{w'}(s_\tau))\nabla_{\theta'} \log \pi_{\theta'}(a_\tau|s_\tau) + \beta\nabla_{\theta'} \sum_a (-\pi_{\theta'}(a_\tau|s_\tau) \log \pi_{\theta'}(a_\tau|s_\tau))$

    end if

  end for

  $t \leftarrow t + 1$

  Update w using $\overline{dw'}$

  Update θ using $\overline{d\theta'}$

until Convergence

Fig. 3 *Algorithm 1: APRL algorithm*

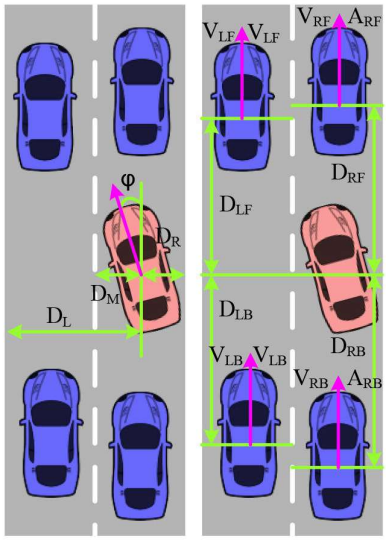

Fig. 4 *Illustration of state representations*

3 Experiment design

3.1 Driving task description

This paper focuses on two-lane highway driving. The inner (or left) lane is a high-speed lane with the speed limit from 100 to 120 km/h, and the outer (or right) lane is a low-speed lane with the speed limit from 60 to 100 km/h [28]. The self-driving car would be initialised at a random position of these two lanes. The destination is also a random set on the inner lane or outer lane with a random distance from 500 to 1000 m. The self-driving car needs to learn how to reach the destination as fast as possible without breaking traffic rules or colliding with other cars. If the initial position and destination are both on the low-speed lane, the car has to change lane at least twice during the driving task. At first, the car needs to change to the high-speed lane and keeps driving for a while, and then switches back to the low-speed lane when it approaches the destination. Continuous surrounding traffic is presented in both lanes to imitate real traffic situations, and is randomly initialised at the beginning of the task. The continuous traffic flow means that there are always other cars driving around the ego vehicle. The movement of other cars is controlled by the car-following model presented by Toledo *et al.* [29]. The dynamics of the self-driving car is approximated by a dynamic bicycle model [30]. In addition, the system frequency is 40 Hz.

3.2 Action space

Previous RL studies on autonomous vehicle decision making usually take the front wheel angle and acceleration as the policy outputs [19, 20]. This is unreasonable because the physical limits of the actuators are not considered. To prevent large discontinuity of the control commands, the outputs of manoeuvre policy SP-Net and AP-Net are front-wheel angle increment and acceleration increment, respectively, i.e.

$$A_{\text{sub}} = \begin{cases} -0.8{:}0.2{:}0.8^\circ/\text{Hz} & \text{SP} - \text{Net} \\ -0.2{:}0.05{:}0.2\,(\text{m/s}^2)/\text{Hz} & \text{AP} - \text{Net} \end{cases} \tag{5}$$

The outputs of SP-Net and AP-Net are designed with reference to the previous researches [31–33]. On the other hand, the highway driving task in this study is decomposed into three manoeuvres: driving in lane, left lane change and right lane change. As mentioned above, the driving-in-lane manoeuvre is a combination of many behaviours, including lane-keeping, car-following and free driving. These three manoeuvres are proved to account for >98% of the highway driving [34]. Therefore, the action space the master policy is defined as follows:

$$A_{\text{master}} = \begin{cases} 0 & \text{driving in lane} \\ 1 & \text{left lane change} \\ 2 & \text{right lane change} \end{cases} \tag{6}$$

3.3 State representation and reward function

We propose four types of states to represent driving scenario and task: states related to the host car, states related to the road, states related to other cars and states related to the destination. It should be noted that we only involve four surrounding cars in this problem: a preceding car and a following car in each lane. In total, we propose 26 indicators to represent the driving state, some of which are illustrated in Fig. 4. A complete list of the states is given in Table 1. Each state is added with noise from uniform distribution before it is observed by the car-learner. We select a part of the states from the list as inputs to the relevant policy networks and value networks. The state space and reward function of each network are varied and depend on the type of manoeuvre and action.

3.3.1 Driving in lane: For the driving-in-lane manoeuvre, the host car only needs to concern the traffic in its current lane from an ego-centric point of view. Therefore, we only need to design the state to model the current lanes. Furthermore, the SP-Net of this manoeuvre is only used for lateral control, and the state space for this situation does not need to include the information of the preceding car. Hence, as shown in Table 1, the state space for the lateral control and longitudinal control is different.

Intuitively, the reward for lateral control and longitudinal control should also be different. For lateral control learning, we use a reward function which provides a positive reward +1 at each step if the car drives in the lane, and a penalty of 0 for a terminal state of crossing the lane lines. For longitudinal control learning, we provide a positive reward proportional to the car's velocity if the speed is in the range of speed limit. The car would receive a penalty according to the extent of overspeed or underspeed, and a penalty of 0 for a terminal state of colliding with the preceding vehicle. In summary, the reward function of driving-in-lane manoeuvre r_d can be expressed as follows:

$$r_{d,\text{lateral}} = \begin{cases} 0 & \text{crossing lane} \\ 1 & \text{else} \end{cases}$$

$$r_{d,\text{longit.}} = \begin{cases} 2 - 0.01(V_{\#\text{U}} - V) & V_{\#L} \le V \le V_{\#\text{U}} \\ 0.5 + 0.01(V_{\#\text{U}} - V) & -20 \le V_{\#\text{U}} - V \le 0 \\ 0.5 + 0.01(V - V_{\#L}) & -40 \le V - V_{\#L} \le 0 \\ 0 & \text{else or collisions} \end{cases}$$

where the subscript $\# \in \{\text{R}, \text{L}\}$ represents the right and left lanes.

3.3.2 Right lane change: For the right-lane-change manoeuvre, the car needs to concern the traffic in both the current lane and the right adjacent lane when making decisions. Therefore, we need to encode the information of the two lanes to define the state space for lane-change manoeuvres. Compared with the driving-in-lane manoeuvre, the lateral and longitudinal controls in lane change share the same state (see Table 1) and similar reward function. The sub-policy of lane change may contain both lane change and driving in lane functions. Such coupling of functions may lead to ambiguity in policy execution. Hence, a clear definition of lane-change manoeuvre has been given to completely decouple the lane change manoeuvre and driving-in-lane manoeuvre. We designate the moment when the car centre crosses the right lane line as the completion point of a right lane change. Note that completion does not necessarily mean success. If the car has crossed the right lane line, the learned sub-policy of Section 3.3.1 is then used to keep the car driving in the current lane. A lane-change manoeuvre would be marked as a success only if the host car does not collide with other cars or cross the lane lines in the next 5 s (200 steps). Otherwise, the lane change manoeuvre would be labelled as a failure.

For both the lateral and longitudinal controls, we use a reward function assigning a penalty of -0.5 at each step and -100 for a terminal state of a failed lane-change manoeuvre. Meanwhile, the car would receive a reward $+100$ for a successful right lane change. In addition, the lateral control module would receive a penalty of -100 for a terminal state of crossing the left lane markings, and the longitudinal control module would receive a penalty of -100 for a terminal state of colliding with the preceding car. In summary, the reward function of right-lane-change manoeuvre r_r can be expressed as follows:

$$r_{\text{r,lateral}} = \begin{cases} 100 & \text{success label} \\ -100 & \text{failure label} \\ -0.5 & \text{else} \end{cases}$$

$$r_{\text{r,longit.}} = \begin{cases} 100 & \text{success label} \\ -100 & \text{failure label or colliding before lane} - \text{change} \\ -0.5 & \text{else} \end{cases}$$

3.3.3 Left lane change: The state space and reward function of the left-lane-change manoeuvre is similar to Section 3.3.2, except that the host vehicle needs to concern the left adjacent lane instead of right. The state space of this manoeuvre is also listed in Table 1 and the reward function is omitted.

Table 1 State representation

Type	Note	Unit	Max noise[a]	Manoeuvre 0				Manoeuvre 1	Manoeuvre 2	Master task
				Left lane		Right lane				
				SP-Net	AP-Net	SP-Net	AP-Net			
host car[b]	δ	°	0.2	✓		✓		✓	✓	✓
	V	km/h	0.5	✓	✓	✓	✓	✓	✓	✓
	A	m/s^2	0.1	✓	✓	✓	✓	✓	✓	✓
road[c]	ϕ	°	0.5	✓		✓		✓	✓	✓
	D_L	M	0.05	✓					✓	✓
	D_M	m	0.05	✓		✓		✓	✓	✓
	D_R	m	0.05			✓		✓		✓
	L_C	—	—							✓
	V_{LU}	km/h	-		✓			✓		✓
	V_{LL}	km/h	-		✓			✓		✓
	V_{RU}	km/h	-				✓		✓	✓
	V_{RL}	km/h	-				✓		✓	✓
other cars	D_{LF}	m	1		✓			✓	✓	✓
	V_{LF}	m /h	1		✓			✓	✓	✓
	A_{LF}	m/s^2	0.3		✓			✓	✓	✓
	D_{LB}	M	1					✓		✓
	V_{LB}	km/h	1					✓		✓
	A_{LB}	m/s^2	0.3					✓		✓
	D_{RF}	m	1				✓	✓	✓	✓
	V_{RF}	km/h	1				✓	✓	✓	✓
	A_{RF}	m/s^2	0.3				✓	✓	✓	✓
	D_{RB}	m	1						✓	✓
	V_{RB}	km/h	1						✓	✓
	A_{RB}	m/s^2	0.3						✓	✓
destination[d]	L_{D}	—	—							✓
	D_{D}	m	1							✓

[a]Each state is added with noise from a uniform distribution $\mathcal{U}$ [−max noise, + max noise] before it is observed by car.

[b] δ: front wheel angle; V: longitudinal velocity; A: longitudinal acceleration.

[c] L_C: current lane (0 for left, 1 for right); V_{LU} and V_{LL}: V – upper/lower speed limit of the left lane, V_{RU} and V_{RL} are similar.

[d] L_D: destination lane (0 for left 1, for right); D_D: distance to the destination.

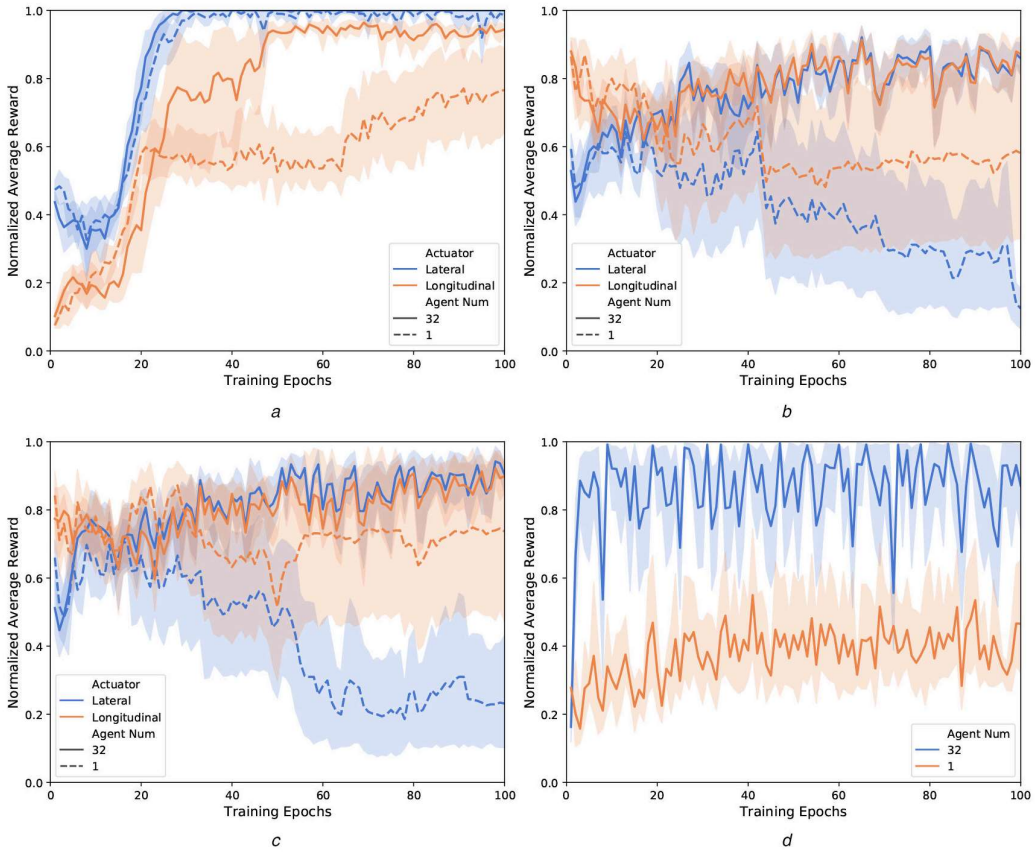

Fig. 5 *Training performance (one epoch corresponds to 1000 updates for the shared networks)*
(a) Driving-in-lane manoeuvre, ***(b)*** Right-lane-change manoeuvre, ***(c)*** Left-lane-change manoeuvre, ***(d)*** Manoeuvre selection

3.3.4 Driving task: Given the three existing manoeuvre policies, the car needs to concern all the states in Table 1 when making a high-level manoeuvre selection. First, we use a reward function to provide a positive reward of +1 at each step if the car drives in the high-speed lane and +0.5 for the low-speed lane. Of course, any manoeuvre leading to irregularities and collisions would receive a penalty of 0 along with a terminal signal. Unlike the learning of manoeuvre policies, the master policy should consider the destination information. Hence, the car would receive a final reward of +200 if it reaches the destination. Otherwise, it would receive a penalty of 0. In summary, the reward function of the driving task r_t can be expressed as follows:

$$r_t = \begin{cases} 1 & \text{high} - \text{speed lane} \\ 0.5 & \text{low} - \text{speed lane} \\ 200 & \text{reaching destination} \\ 0 & \text{missing destination, irregularities, collisions} \end{cases}$$

Not all manoeuvres are available in every state throughout driving. Considering the driving scenario of this study, the left-lane-change manoeuvre is only available in the right lane, and the right-lane-change manoeuvre is only available in the left lane. Thus, we predesignate a list of available manoeuvres via the observations at each step. Taking an unavailable manoeuvre is considered an error. The self-driving car should filter its manoeuvre choices to make sure that only legal manoeuvres can be selected.

4 Results

The architecture of all the policy networks and value networks is the same except their output layers. For each network, the input layer is composed of the states followed by 2 fully-connected layers with 256 hidden units for each layer. We use exponential linear units (ELUs) for hidden layers, while the output layers of all the networks are fully-connected linear layers. The output of the value network is a scalar $V(s; w)$. On the other hand, the outputs of the policy network are a set of action preferences $h(a|s; \theta)$, one for each action. The action with the highest preference in each state is given the highest probability of being selected. Then, the numerical preferences are translated to softmax outputs which could be used to represent the selection probability of each action:

$$\pi(a|s;\theta) = \frac{\exp(h(a|s;\theta))}{\sum_a(\exp(h(a|s;\theta)))}$$

The optimisation process runs 32 asynchronous parallel car-learners at a fixed frequency of 40 Hz. During the training, we run backpropagation after $n = 10$ forward steps by explicitly computing 10-step returns. The algorithm uses a discount of $\gamma = 0.99$ and entropy regularisation with a weight $\beta = 0.01$ for all the policies. The method performs updates after every 32 actions $(I_{\text{update}} = 32)$ and shared RMSProp is used for optimisation, with an RMSProp decay factor of 0.99. We learn the NN parameters with a learning rate of 10^{-4} for all learning processes. For each episode, the maximum step of each car-learner is $t_{\max} = 5000$.

4.1 Training performance

We compared the training performance of the algorithm with 32 asynchronous parallel car-learners and that with only one car-learner. Fig. 5 shows the average value and 95% confidence interval of normalised reward for ten different training runs. The training results of driving-in-lane manoeuvre are shown in Fig. 5*a*. As shown in this figure, both the learning speed, training stability and policy performances of APRL are much higher than the single-learner RL. On the other hand, it is easy to see that the SP-Net converges much faster than the AP-Net, which are nearly stable after approximately 30 epochs and 50 epochs, respectively. Besides, the lateral reward reaches a more stable plateau than the longitudinal reward. These noticeable differences are mainly

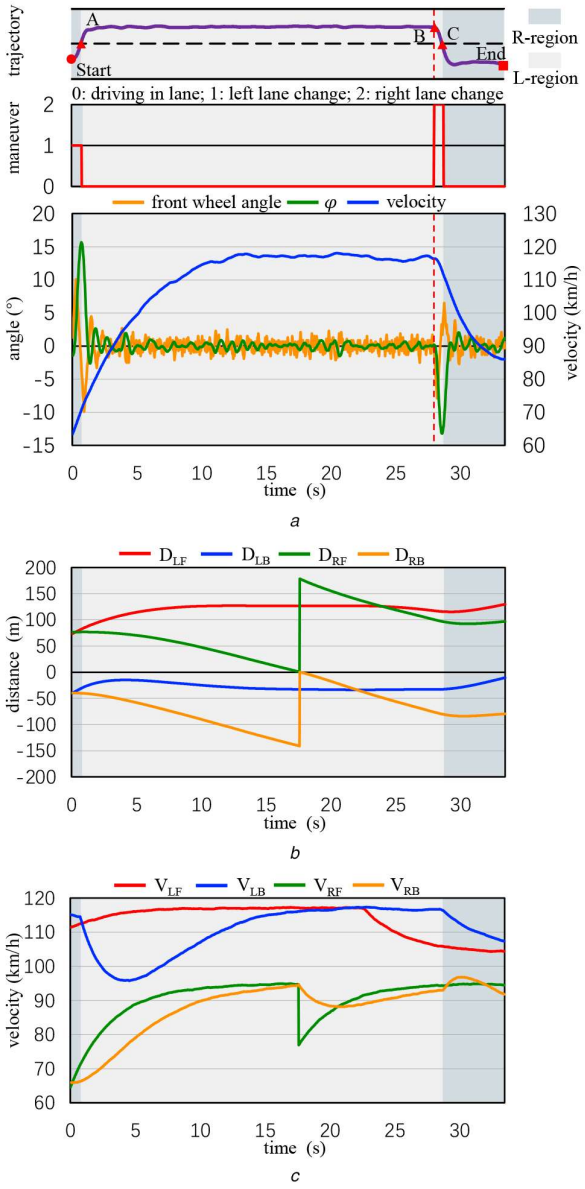

Fig. 6 *Simulation results display*

(a) High-level manoeuvre selection and low-level motion control, ***(b)*** Relative distance from other cars, ***(c)*** Speed of other cars (*R*-region and *L*-region indicate that the ego vehicle is driving in the right lane and the left lane, respectively.)

caused by two reasons. First, the delay between the acceleration increment and the resulting longitudinal rewards is too long. For example, the car usually needs to take hundreds of time steps to improve its speed to cause a rear-end collision. Nevertheless, the car may receive a penalty for crossing the road lane marking within only 20 steps. Second, the self-driving car is initialised with random velocity and position at the beginning of each episode. This would affect the longitudinal reward received by the car, because it is related to the velocity and speed limit.

The training results of right-lane-change manoeuvre are shown in Fig. 5*b*. For the APRL algorithm, both the lateral and longitudinal control modules stabilised after about 60 epochs. The fluctuation of the two curves around the peak after convergence is caused by random initialisation. The training results of left-lane-change manoeuvre are shown in Fig. 5*c*, which are similar to results of the right-lane-change manoeuvre. On the other hand, the single-learner RL fails to learn suitable policies in these two cases because the state dimension is relatively high. Therefore, lane-change manoeuvre requires more exploration and is more complicated than driving-in-lane manoeuvre, which is too hard for single car-learner.

Fig. 5*d* shows the training performance curve for the master policy. The results of APRL show that the master policy converges much more quickly than the manoeuvre policy, which reaches a plateau after around 10 epochs. The fluctuation is caused by the random initialisation and the number of times the self-driving car reaches the destination in each epoch. In comparison, the learning speed and learning performance of single-agent RL are very poor.

4.2 Simulation verification

We conduct some simulations to assess the performance of the learned master policy and manoeuvre policies. The simulation platform used in this paper, including the traffic flow module and the vehicle model module, is directly built-in Python. The states of all vehicles are randomly initialised for each simulation. The self-driving car is initialised on the low-speed lane with a random velocity. The destination is randomly placed between 800 and 1000 m ahead of the car in the same lane. Random continuous traffic is presented in both lanes to imitate real traffic situations. Fig. 6*a* displays the decision making progress of both the master policy and the manoeuvre policies in a particular driving task. The position and velocity of other cars are shown in Figs. 6*b* and *c*. Note that the sudden change in the curves in these two subgraphs (take the green line as an example) represents the change in relative position between the self-driving car and other cars. In particular, when the ego vehicle is driving in the left lane, D_{LF} and V_{LF} are the following distance and the speed of the preceding vehicle, respectively. Similarly, D_{RF} and V_{LF} have similar meanings when the ego vehicle is driving in the right lane. The car corresponding to this curve also changes at this time.

As shown in Fig. 6*a*, the self-driving car changes to the high-speed lane (from point Start to point A) at the beginning of the simulation and then keeps driving in this lane for a while. In the meantime, the velocity is increased in this phase from around 65 to 118 km/h. The self-driving car switches to low-speed lane (from point B to point C) when approaching the destination (point End), and the velocity eventually decreases to around 86 km/h. Interestingly enough, the car tends to drive near the lane centreline based on its manoeuvre policy, but the reward function does not explicitly instruct it to do that. This is because that driving near the centreline has the highest state value with respect to the SV-Net of the driving-in-lane manoeuvre. Driving near the road lane marking will easily cause the lane departure due to the state noise and high speed. This demonstrates that our method is able to learn a rationale value network to evaluate the driving state. The master policy, together with the three manoeuvre policies, are able to realise smooth and safe decision making on the highway.

We also directly trained non-hierarchical longitudinal and lateral driving policies using reward functions similar to those mentioned in Section 3.3.4. Both networks take all the states in Table 1 as input, and the NN architecture is the same as the manoeuvre NNs mentioned above. Fig. 7*a* shows the boxplots of the average reward and driving time for 20 different simulations of both hierarchical and non-hierarchical methods. Fig. 7*b* plots the typical decision trajectories for these two algorithms. It is clear that policy of non-hierarchical method has not learned to increase the average driving speed by changing to the high-speed lane, resulting in lower average speed and longer driving time (about 25% slower). Obviously, the policy found by the non-hierarchical method is the local optimal solution. This is because during the training process, the self-driving often collides with other vehicles when changing to the other lane, so it thinks it is best to drive in the current lane. This is why we chose the hierarchical architecture for self-driving decision-making.

4.3 Sensitivity to state noise

As mentioned above, we add sensing noise to each state before being observed by the host car during training. Denoting the max noise of each state in Table 1 as $N_{\max}$, we assume that the sensing noise obeys uniform distribution $\mathcal{U}(-N_{\max}, N_{\max})$. In a real

application, the sensor accuracy is usually determined by sensor configuration and sensing algorithms [35–37]. Therefore, the sensor noise of certain states of a real car may be greater than N_{max}. We assume that the noise of all states in practical application is M times the noise used in training, so the real sensing noise obeys $\mathcal{U}(-M*N_{max}, M*N_{max})$. To assess the sensitivity of the H-RL algorithm to state noise, we fix the parameters of the previously trained master policy and manoeuvre policies and assess these policies for different M.

Fig. 8 shows the average value and 95% confidence interval of normalised reward for ten different trained policies. It is clear that the policy is less affected by noise when $M \leq 7$. It is usually easy for today's sensing technology to meet this requirement [35–39]. To a certain extent, this shows that trained policies have the ability to transfer to real applications. Of course, it would be better if we add the same sensing noise like the real car during the training.

5 Conclusion

In this paper, we provide an H-RL method for decision making of self-driving cars, which does not rely on a large amount of labelled driving data. This method comprehensively considers the high-level manoeuvre selection and the low-level motion control in both the lateral and longitudinal directions. We first decompose the driving tasks into three manoeuvres, i.e. driving in lane, right lane change and left lane change, and learn the sub-policy for each manoeuvre. The driving-in-lane manoeuvre is a combination of many behaviours including lane-keeping, car-following and free driving. Then, a master policy is learned to choose the manoeuvre policy to be executed at the current state. The motion of the vehicle is controlled jointly by the lateral and longitudinal actuators, and the two types of actuators are relatively independent. All policies including master policy and manoeuvre policies, are represented by fully-connected neural networks and trained using the proposed APRL algorithm, which builds a mapping from the sensory inputs to driving decisions. We apply this method to a highway driving scenario, and the state and reward function of each policy are designed separately, which demonstrates that it can realise smooth

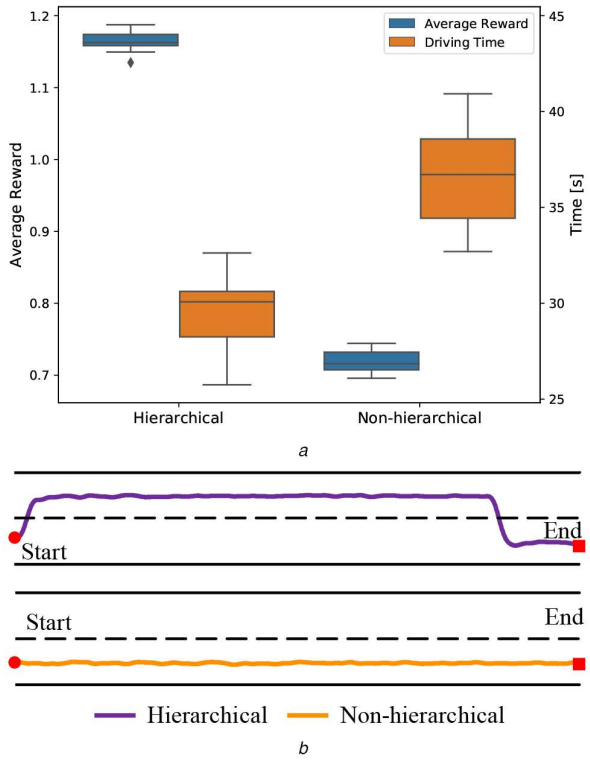

Fig. 7 *Performance comparison*
(a) Average reward and time consumption per simulation, ***(b)*** Driving trajectory

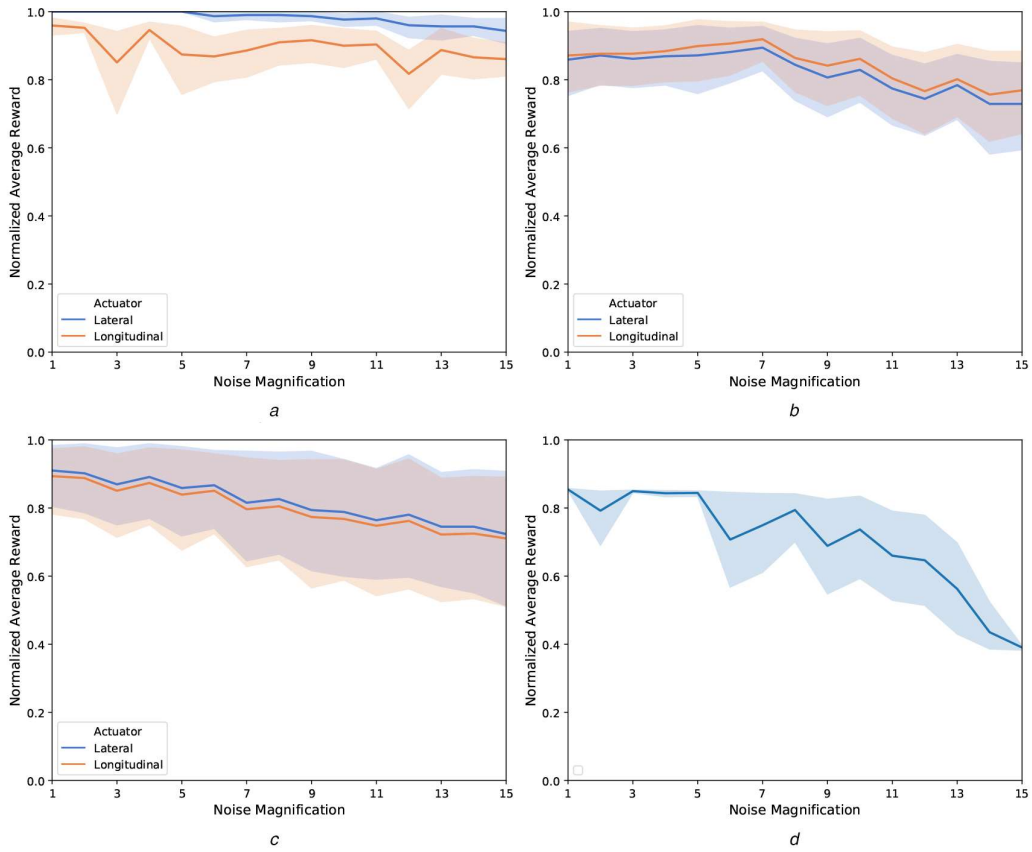

Fig. 8 *Policy performance for different noise magnification M*
(a) Driving-in-lane manoeuvre, ***(b)*** Right-lane-change manoeuvre, ***(c)*** Left-lane-change manoeuvre, ***(d)*** Manoeuvre selection

and safe decision making for self-driving cars. Compared with the non-hierarchical method, the driving time spent per simulation is reduced by ~25%. In the future, we will continue to validate and improve our algorithms on highways with complex traffic, intersections and urban environment.

6 Acknowledgments

This study was supported by the International Sci&Tech Cooperation Program of China under grant no. 2019YFE0100200 and NSF China with U1664263. The authors also like to acknowledge Renjie Li and Long Xin for their valuable suggestions for this paper.

7 References

[1] Paden, B., Čáp, M., Yong, S.Z., *et al.*: 'A survey of motion planning and control techniques for self-driving urban vehicles', *IEEE Trans. Intell. Veh.*, 2016, **1**, (1), pp. 33–55

[2] Bojarski, M., Yeres, P., Choromanska, A., *et al.*: 'Explaining how a deep neural network trained with end-to-end learning steers a car', arXiv preprint arXiv:170407911, 2017

[3] Katrakazas, C., Quddus, M., Chen, W.H., *et al.*: 'Real-time motion planning methods for autonomous on-road driving: state-of-the-art and future research directions', *Transp. Res. C, Emerg. Technol.*, 2015, **60**, pp. 416–442

[4] Montemerlo, M., Becker, J., Bhat, S., *et al.*: 'Junior: the Stanford entry in the urban challenge', *J. Field Robot.*, 2008, **25**, (9), pp. 569–597

[5] Furda, A., Vlacic, L.: 'Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making', *IEEE Intell. Transp. Syst. Mag.*, 2011, **3**, (1), pp. 4–17

[6] Glaser, S., Vanholme, B., Mammar, S., *et al.*: 'Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction', *IEEE Trans. Intell. Transp. Syst.*, 2010, **11**, (3), pp. 589–606

[7] Kala, R., Warwick, K.: 'Motion planning of autonomous vehicles in a non-autonomous vehicle environment without speed lanes', *Eng. Appl. Artif. Intell.*, 2013, **26**, (5), pp. 1588–1601

[8] Duan, J., Li, R., Hou, L., *et al.*: 'Driver braking behavior analysis to improve autonomous emergency braking systems in typical Chinese vehicle-bicycle conflicts', *Accident Anal. Prev.*, 2017, **108**, pp. 74–82

[9] Hou, L., Duan, J., Wang, W., *et al.*: 'Drivers braking behaviors in different motion patterns of vehicle-bicycle conflicts', *J. Adv. Transp.*, 2019, **2019**, pp. 1–17, doi: 10.1155/2019/4023970

[10] Li, G., Yang, Y., Qu, X.: 'Deep learning approaches on pedestrian detection in hazy weather', *IEEE Trans. Ind. Electron.*, 2019, pp. 1–1, doi: 10.1109/TIE.2019.2945295

[11] LeCun, Y., Bengio, Y., Hinton, G.: 'Deep learning', *Nature*, 2015, **521**, (7553), pp. 436–444

[12] Pomerleau, D.A.: 'Alvinn: an autonomous land vehicle in a neural network'. Advances in Neural Information Processing Systems, Denver, CO, USA, 1989, pp. 305–313

[13] LeCun, Y., Cosatto, E., Ben, J., *et al.*: 'Dave: Autonomous off-road vehicle control using end-to-end learning', Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/yann/research/dave/index.html, 2004

[14] Chen, C., Seff, A., Kornhauser, A., *et al.*: 'Deepdriving: learning affordance for direct perception in autonomous driving'. Proc. of the IEEE Int. Conf. on Computer Vision, Santiago, Chile, 2015, pp. 2722–2730

[15] Sutton, R.S., Barto, A.G.: '*Reinforcement learning: an introduction*' (MIT Press, Cambridge, MA, USA, 2018)

[16] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Human-level control through deep reinforcement learning', *Nature*, 2015, **518**, (7540), pp. 529–533

[17] Silver, D., Huang, A., Maddison, C.J., *et al.*: 'Mastering the game of go with deep neural networks and tree search', *Nature*, 2016, **529**, (7587), pp. 484–489

[18] Silver, D., Schrittwieser, J., Simonyan, K., *et al.*: 'Mastering the game of go without human knowledge', *Nature*, 2017, **550**, (7676), p. 354

[19] Lillicrap, T.P., Hunt, J.J., Pritzel, A., *et al.*: 'Continuous control with deep reinforcement learning', arXiv preprint arXiv:150902971, 2015

[20] Mnih, V., Badia, A.P., Mirza, M., *et al.*: 'Asynchronous methods for deep reinforcement learning'. Int. Conf. on Machine Learning, New York, NY, USA, 2016, pp. 1928–1937

[21] Daniel, C., Van Hoof, H., Peters, J., *et al.*: 'Probabilistic inference for determining options in reinforcement learning', *Mach. Learn.*, 2016, **104**, (2–3), pp. 337–357

[22] Wei, J., Snider, J.M., Gu, T., *et al.*: 'A behavioral planning framework for autonomous driving'. IEEE Intelligent Vehicles Symp. (IV), Dearborn, MI, USA, 2014, pp. 458–464

[23] Sutton, R.S., McAllester, D.A., Singh, S.P., *et al.*: 'Policy gradient methods for reinforcement learning with function approximation'. Advances in Neural Information Processing Systems, Denver, CO, USA, 2000, pp. 1057–1063

[24] Degris, T., Pilarski, P.M., Sutton, R.S.: 'Model-free reinforcement learning with continuous action in practice'. American Control Conf. (ACC), Montreal, QC, Canada, 2012, pp. 2177–2182

[25] Bhatnagar, S., Sutton, R., Ghavamzadeh, M., *et al.*: 'Natural actor-critic algorithms', *Automatica*, 2009, **45**, (11), pp. 2471–2482

[26] Sutton, R.S., Precup, D., Singh, S.: 'Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning', *Artif. Intell.*, 1999, **112**, (1–2), pp. 181–211

[27] Gopalan, N., Littman, M.L., MacGlashan, J., *et al.*: 'Planning with abstract Markov decision processes'. Twenty-Seventh Int. Conf. on Automated Planning and Scheduling (ICAPS), Pittsburgh, PA, USA, 2017

[28] Liao, Y., Li, S.E., Wang, W., *et al.*: 'Detection of driver cognitive distraction: a comparison study of stop-controlled intersection and speed-limited highway', *IEEE Trans. Intell. Transp. Syst.*, 2016, **17**, (6), pp. 1628–1637

[29] Toledo, T., Koutsopoulos, H.N., Ben Akiva, M.: 'Integrated driving behavior modeling', *Transp. Res. C, Emerg. Technol.*, 2007, **15**, (2), pp. 96–112

[30] Li, R., Li, Y., Li, S.E., *et al.*: 'Driver-automation indirect shared control of highly automated vehicles with intention-aware authority transition'. Intelligent Vehicles Symp. (IV), Los Angeles, CA, USA, 2017, pp. 26–32

[31] Attia, R., Orjuela, R., Basset, M.: 'Coupled longitudinal and lateral control strategy improving lateral stability for autonomous vehicle'. American Control Conf. (ACC), Montreal, QC, Canada, 2012, pp. 6509–6514

[32] Katriniok, A., Maschuw, J. P., Christen, F., *et al.*: 'Optimal vehicle dynamics control for combined longitudinal and lateral autonomous vehicle guidance'. European Control Conf. (ECC), Zurich, Switzerland, 2013, pp. 974–979

[33] Gao, Y., Gray, A., Frasch, J. V., *et al.*: 'Semi-autonomous vehicle control for road departure and obstacle avoidance', *IFAC Control Transp. Syst.*, 2012, **2**, pp. 1–6

[34] Li, G., Li, S.E., Cheng, B., *et al.*: 'Estimation of driving style in naturalistic highway traffic using maneuver transition probabilities', *Transp. Res. C, Emerg. Technol.*, 2017, **74**, pp. 113–125

[35] de Ponte Müller, F.: 'Survey on ranging sensors and cooperative techniques for relative positioning of vehicles', *Sensors*, 2017, **17**, (2), pp. 1–27

[36] Jeng, S.L., Chieng, W.H., Lu, H.P.: 'Estimating speed using a side-looking single-radar vehicle detector', *IEEE Trans. Intell. Transp. Syst.*, 2013, **15**, (2), pp. 607–614

[37] Park, K.Y., Hwang, S.Y.: 'Robust range estimation with a monocular camera for vision-based forward collision warning system', *Scientific World J.*, 2014, **2014**, pp. 1–9

[38] Guan, H., Li, J., Cao, S., *et al.*: 'Use of mobile lidar in road information inventory: a review', *Int. J. Image Data Fusion*, 2016, **7**, (3), pp. 219–242

[39] Cao, Z., Yang, D., Jiang, K., *et al.*: 'A geometry-driven car-following distance estimation algorithm robust to road slopes', *Transp. Res. C, Emerg. Technol.*, 2019, **102**, pp. 274–288