Driving Maneuvers Prediction based Autonomous Driving Control by Deep Monte Carlo Tree Search

Jienan Chen, *Member, IEEE,* Cong Zhang, Jinting Luo, Junfei Xie, *Member, IEEE,* Yan Wan, *Senior Member, IEEE*

***Abstract*—Autonomous driving has attracted significant attention in recent years. With the booming of artificial intelligence (AI), deep learning technologies have been applied to autonomous driving to help vehicles better perceive the environment. Besides the perceiving environment, predictive driving is another prominent smooth control and safe driving skill for human drivers. In this work, we develop a deep Monte Carlo Tree Search (deep-MCTS) control method for vision-based autonomous driving. Compared with existing deep learning-based autonomous driving control methods, our method can predict driving maneuvers to help improve the stability and performance of driving control. Two deep neural networks (DNNs) are employed for predicting action-state transformation and obtaining action-selection probabilities, respectively. The deep-MCTS utilizes the predicted information of the two DNNs and reconstructs multiple possible trajectories to predict driving maneuvers. An optimal trajectory is selected by the deep-MCTS based on both current road conditions and predicted driving maneuvers. The proposed method achieves high control stability by avoiding sharp turns and driving deviations. We implement our algorithm in the Udacity and Torcs self-driving environments. The testing results show that our algorithm achieves a significant improvement in training efficiency, the stability of steering control, and stability of driving trajectory compared to existing methods.**

***Index Terms*—Autonomous driving, Artificial intelligence, Deep Monte Carlo Tree Search, Deep neural network.**

I. Introduction

In recent years, autonomous driving has drawn significant attention from both academia and industries, such as Google, Baidu, Uber, and Nvidia [1]–[5]. Traditional autonomous driving technologies employ various sensors, such as radar, Lidar, odometry, computer vision, sonar, GPS, and inertial measurement units, to perceive the surroundings [6]–[8]. However, equipping these sensors on vehicles is costly and also influences the vehicles' aerodynamics. The high cost of these sensors also limits their wide applications to civilian vehicles. Moreover, most traditional autonomous driving techniques perceive the environment by estimating the 3-dimensional (3D) physical surface of the vehicles' surroundings and are not intelligent enough to truly understand the environment [9]–[11].

Human drivers can solely rely on the vision information to control a vehicle in complex scenarios. Recently, more researches focus on vision-only based autonomous driving [12]–[16] than traditional Lidar based autonomous driving, where the camera is the only sensor for the information collection. The motivation of the development of vision-only based autonomous driving is to imitate the human-like thinking process on perceiving the environment and driving behavior. Besides, with the advancement of computer vision [17], [18], deep learning becomes a promising solution for the design of vision-only based autonomous driving systems [19], addressing some drawbacks of traditional autonomous driving methodologies. Currently, most existing vision-based autonomous driving controllers are achieved through Imitating Learning (IL) [12], [13]. In particular, the vision information captured by cameras on the vehicle and the corresponding control commands sampled from the human expert's driving process is first used to train a classifier or regressor. In the course of driving, this classifier or regressor and the captured vision information are then used to imitate and predict the expert's control commands. A major issue of such IL-based autonomous driving is error accumulation. If control errors are generated during the driving process, the observation bias can be accumulated, causing a totally different future observation. Hence, the IL-based end-to-end control methods require large amount of training data that cover the whole observation space, which is very costly in terms of data collection. To solve this issue, a Long Short-Term Memory (LSTM) with a Mixture Density Network (MDN) architecture to correct the accumulated errors was proposed in [20]. However, it is limited to the imitation task scenario.

Recently, the deep reinforcement learning (DRL), a powerful Artificial Intelligence (AI) technique for solving decision-making problems [21]–[23], has also been employed in autonomous driving [24]–[26]. A typical DRL-based autonomous driving controller consists of two basic modules, a perception module and a control module. The perception module extracts useful features from the input images to locate the vehicle on the track. The control module generates control signals to keep the vehicle following a desired trajectory. The DRL algorithms can be applied in both the perception and control modules. Li et al. [14] introduce a multi-task learning (MTL) and actor-critical network based autonomous driving method. However, we believe that there is still large room for improvement in terms of stability of steering control and

J. Chen, C. Zhang, and J. Luo are with the National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China (Corresponding author: Jienan Chen, Jesson.chen@outlook.com).

J. Xie is with Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, USA, 92182.

Y. Wan is with Electrical Engineering, University of Texas at Arrington, Arlington, TX, 79163

Algorithm I General MCTS approach

Input: the current state s_0; Output: the best action a

1: **function** MCTS Search(s_0)
3: Create root node v_0 with state s_0
5: **while** within max iteration **do**
6: $v_l \leftarrow$TreePolicy(v_0)
7: $\Delta \leftarrow$DefaultPolicy($s(v_l)$)
8: Backup(v_l, Δ)
9: return a(BestChild(v_0))

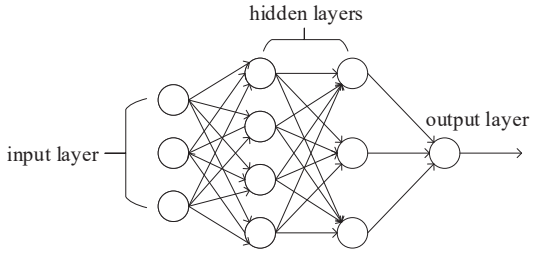


Fig. 1. The layer structure of DNN.

stability of driving trajectory.

In this paper, we introduce a DRL based deep Monte Carlo Tree Search (deep-MCTS) control method for autonomous driving. Different from existing DRL-based autonomous driving controllers, the deep-MCTS can predict driving maneuvers, a critical skill for human to ensure safe driving and smooth control. In particular, two deep neural networks (DNNs) are employed to predict action-state transformation and obtain action-selection probabilities. Then, the proposed deep-MCTS utilizes the predicted information from the two networks to reconstruct multiple possible trajectories of the vehicle. The optimal control action is finally selected based on the current road condition and the best predicted trajectories. Extensive experimental results demonstrate the high stability of the proposed control method to avoid the sharp turns and driving deviations. Compared to existing methods, the testing results show that the proposed deep-MCTS algorithm achieves 50.0%, 66.30% and 59.06% improvements in training efficiency, stability of steering control and stability of driving trajectories.

II. Related work

In recent years, many researchers have attempted to apply deep learning techniques for autonomous driving. IL and DRL are the two major deep learning approaches that have been explored.

The IL-based autonomous driving controllers work by imitating human's driving experience. In [13], an end-to-end supervised model for simulated self-driving vehicles is introduced, which takes images captured by a simulated F1 car. The method proposed in [15] employs a convolutional neural network (CNN) to map raw image pixels from a single front-facing camera directly to steering commands. This CNN model is trained using steering angles only but can automatically learn representations of the necessary internal processing steps such as road scene segmentation and decision making. A PilotNet architecture is developed in [27] for determining the elements in the road image that has the main impact on the steering decision. Paper [28] explores a variety of techniques including 3D convolutional neural networks, recurrent neural networks using LSTM, ResNets, etc. to predict the steering angle. However, these techniques have a complex network architecture and require powerful computing equipment to satisfy the real-time demand of autonomous driving. A novel fully-convolutional network and long short-term memory (FCN-LSTM) architecture for autonomous driving is introduced in [29], which is trained using a large-scale crowd-sourced vehicle action data. An object-centric model is developed in [30], which outperforms object-agnostic methods in the scenes involving other vehicles and pedestrians. However, the performance largely depends on the object detection. A framework proposed in [31] can greatly help the decision making in autonomous driving which adopts a state-of-the-art simulator for evaluating end-to-end Bayesian controllers. Paper [32] introduces a new IL method based on the learning-from-intervention dataset aggregation algorithm which is suitable for pedestrian-rich environments. However, this method needs human's interventions and will cost extensive human resources. To achieve autonomous driving with a simple deep neural network, paper [33] presents an end-to-end deep neural network with low complexity, which can be applied to embedded devices. The IL-based methods have achieved competitive performance by imitating human's driving experience. However, to cover a wide range of driving scenarios, the IL-based methods generally need large amount of data, where the data collection can be very costly.

Different from IL-based methods that are trained offline, the DRL-based methods automatically learn the environment online and do not require expensive data collections. In [34], the inverse reinforcement learning is implemented in a highway driving simulator to avoid collisions. Paper [35] employs a deep Q-network to derive the steering and throttle control outputs from raw sensory inputs. A DRL-based method to accomplish the lane keeping task in autonomous driving is proposed in [36]. To improve the training efficiency and driving robustness, an asynchronous actor-critic framework is developed in [37]. To handle complex driving environments, the deep deterministic policy gradient (DDPG) algorithm is adopted for autonomous driving in [16], which generates continuous control commands and achieves high control accuracy. Based on DRL, paper [38] develops a time-efficient navigation policy and realizes autonomous driving among pedestrians at the human's walking speed. Although DRL-based autonomous driving has been explored extensively, existing approaches can't predict future driving maneuvers, which is critical for the performance of driving control.

III. Background

Before we introduce the deep-MCTS control method, we first briefly describe the DNN, MCTS and the vehicular simulation environment in this section as the background for better understanding of the proposed approach.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

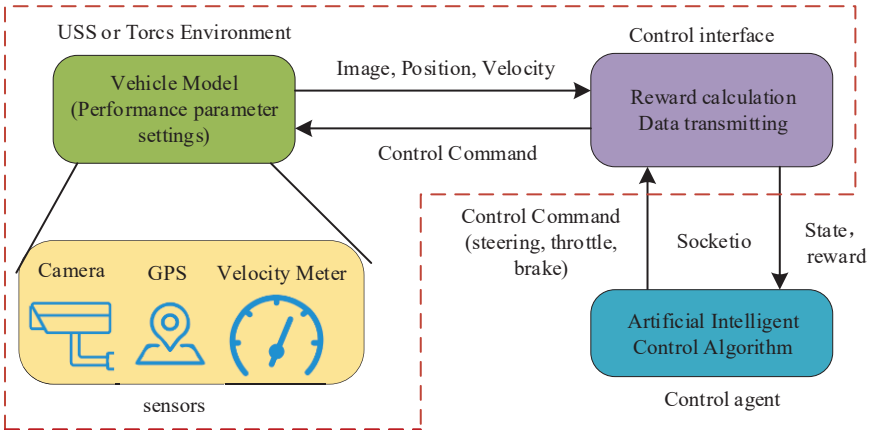


Fig. 3. The USS or Torcs environment (the inside of red dotted line) framework which contains three parts: vehicle model, sensors and control interface.

TABLE I
THE ACTIONS OF DIFFERENT CONTROL COMMANDS.

action	steering angle a_t										
value	$-\frac{5\pi}{36}$	$-\frac{4\pi}{36}$	$-\frac{3\pi}{36}$	$-\frac{2\pi}{36}$	$-\frac{\pi}{36}$	0	$\frac{\pi}{36}$	$\frac{2\pi}{36}$	$\frac{3\pi}{36}$	$\frac{4\pi}{36}$	$\frac{5\pi}{36}$

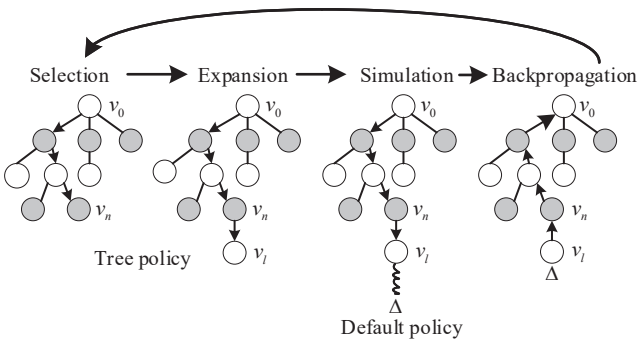


Fig. 2. The searching process of MCTS.

A. Deep Neural Network

DNN is a machine learning method that has found broad applications, such as image recognition, natural language processing, and object detection [39]–[42]. A DNN model has multiple layers with each layer consisting of multiple neurons as illustrated in Fig. 1. The transition between two adjacent layers is captured by

$$\mathbf{X}_{i+1} = f_{act}(\mathbf{W}_i\mathbf{X}_i + \mathbf{b}_i) \tag{1}$$

where i is the index of the layer, $\mathbf{X}_i$ is the activation value of the neurons in the ith layer, $\mathbf{W}_i$ is the weight matrix, $\mathbf{b}_i$ is the bias vector and f_{act} is the activation function. By tuning the network structure and the values of the weight matrices, DNN can well capture the relationship, either linear or non-linear, between the input and output. In this paper, we employ two DNNs to predict action-state transformation and obtain action-selection probabilities.

B. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm for decision-making problems [43]–[45] most notably employed in game play such as Alpha Go [46]. The core idea of MCTS is to find optimal decisions by performing random sampling of the action space iteratively and building a search tree according to the results. In the search tree, each node denotes a state of the decision domain and the directed links to its child nodes denote the actions that result in the subsequent states.

As illustrated in Fig. 2, at each iteration, the MCTS performs four steps: *selection*, *expansion*, *simulation*, and *backpropagation*. In particular, starting from the root node v_0 of the search tree, the algorithm first selects the best child nodes recursively according to the visited count and average reward until a terminate or expandable node v_n with state s_n is reached. If node v_n is an expandable node, the search tree is expanded by selecting an unvisited action a, which leads to a state transition from s_n to s_l. A new leaf node v_l with state s_l is then attached to node v_n. The algorithm then runs the simulation according to the default policy from node v_l to calculate the reward Δ. The simplest default policy is the uniformly random sampling. Finally, in the backpropagation step, the reward Δ is propagated back to the root node and is used to update the statistics of the nodes along the path. Specifically, the visited count of each node along the path is incremented and the average reward is re-calculated based on the reward Δ. The procedures of the MCTS algorithm are summarized in Algorithm 1.

In recent years, some advanced MCTS algorithms have been developed. A full expansion-based MCTS algorithm without the simulation step is proposed in [47] to accelerate the searching speed. Inspired by this result, we employ deep-MCTS based on asynchronous policy and the value MCTS algorithm (APV-MCTS) [47] which contains three steps by merging the expansion and simulation steps.

C. Simulation Environment

The Udacity Self-driving Simulator (USS) and Torcs [48]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

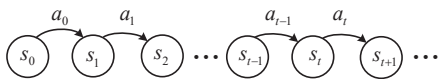


Fig. 4. Illustration of the state transition process in MDP.

Algorithm II Deep MCTS based autonomous driving Control algorithm

Input: image x_t, Output: control command a_t

1: **function** Deep-MCTS(x_t)
2: Load networks f_{VS}, f_{AV}
3: $s_t \leftarrow$Image Preprocessing Module (x_t)
4: $a_t, \boldsymbol{\pi} \leftarrow$ MCTS Search(s_t, f_{VS}, f_{AV})
5: return a_t

are two open-source platforms for autonomous driving. After specifying the basic platform configurations for our experimental tests, the two simulators both mainly consist of three parts: sensors, control interface and vehicle model as shown in Fig. 3. The control interface communicates with the external control agent through the tool, ***socketio*** [49]. ***Socketio*** is a standard communication library, which enables real-time, bi-directional communication. It has been widely employed in various web applications. When data (including GPS position, velocity and RGB images) are collected by the sensors, the vehicle model sends these data to the control interface, which then uses these data to estimate the vehicle's state. Reinforcement learning is adopted here for vehicle control. The information is then transmitted to the external control agent for decision making. After the control command is made by the control agent, the command is then transmitted to the vehicle model through the control interface. The vehicle model finally executes the command.

Algorithm III MCTS Searching process

Input: a copy of current vehicle's state s_t
Output: the best control command ∂_t

1: **function** MCTS Search (s_t, f_{VS}, f_{AV})
2: Create a new root node n_0 with state $\varsigma(n_0) = s_t$
3: **while** within max iteration **do**
4: $n_L \leftarrow$ Select(n)
8: $v =$Expand(n_L, f_{VS}, f_{AV})
9: Backup(n_L, r)
10: Calculate Q of root node by equation (8)
11: Calculate $\boldsymbol{\pi}$ by equation (13)
12: Select the control action $a(n')$ by $\boldsymbol{\pi}$
13: Map $a(n')$ to a_t
14: return $a_t, \boldsymbol{\pi}, Q$
1: **function** Select (n)
2: **while** n is not leaf node **do**
3: $n \leftarrow$Bestchild(n)
1: **function** Expand(n, f_{VS}, f_{AV})
2: $\mathbf{p}, v = f_{AV}(\varsigma(n); w_{AV})$
3: Fully expand node n
4: Initialize statistics of each child node n':
$\varsigma(n') = f_{VS}(\varsigma(n), a|\mathbf{w}_{VS})$
$R(n') = 0, N(n') = 0, a(n') = a, P(n') = p_{n'}$
5: return v
1: **function** Bestchild(n)
2: $n'_{best} = \underset{n' \in children\ of\ n}{\arg\max} (Q(n') + U(n'))$
3: return n'_{best}
1: **function** Backup(n, r)
2: **while** n is not null **do**
3: $N(n) \leftarrow N(n) + 1$
4: $R(n) \leftarrow R(n) + v$
5: $n \leftarrow$parent of n

IV. Deep-MCTS based Autonomous Driving

In this section, we describe the deep-MCTS algorithm to achieve autonomous driving control. This algorithm takes the vision information as the input, models the control process as a Markov Decision Process (MDP) [50]–[54], and adopts the MCTS search procedures to generate the control commands for the autonomous vehicle.

A. Vision-based Autonomous Driving

The input to the USS is a 3-dimensional (3D) RGB image x_t which represents the 3D image captured at time step t. Then we compress the image into a gray-scale image as the vehicle's states s_t. If we consider the autonomous driving controller as a module that generates the control commands based solely on the vision inputs, its mathematical formulation is given as follows

$$a_t = f_A(s_t|\mathbf{w}_A), \tag{2}$$

where a_t is the control action at time t, $\mathbf{w}_A$ is the weight vector and $f_A(\cdot)$ maps the image to the control action.

Here, the control action a_t refers to the steering angle, which is discretized according to TABLE I. The total number of possible control actions is 11.

B. Deep Reinforcement Learning

To design the controller that maps the vehicle's state s_t to a control action a_t, i.e., $f_A(\cdot)$ in Equation (2), we adopt the DRL, which formulates a control problem as an MDP illustrated in Fig. 4. In particular, at each time step t, the DRL selects an action a_t, and then picks a new state s_{t+1} according to a state transition probability $P(s_{t+1}|s_t, a_t)$. A function $\pi : S \rightarrow A$ specifies the action to take given the current state, where S and A represent the state and control spaces, respectively. To determine the control policy, a reward function, $r_t = r(s_t, a_t)$, is introduced, which evaluates the immediate reward of taking action a_t when in state s_t. The optimal control policy π^* is then found by maximizing a total expected reward as follows

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi}:S\rightarrow A}{\arg\max}\, Q^{\pi}(s,a) = \underset{\boldsymbol{\pi}:S\rightarrow A}{\arg\max}\, \mathrm{E}_{s,a\sim\pi,r}[R], \tag{3}$$

where

$$R = \sum_{k=t}^{T} \gamma^{k-t} r_k. \tag{4}$$

T is the terminal time. $\gamma \in (0, 1)$ is a discount factor that reduces the impact of future rewards. The reward function r_t can take various forms, such as the metric that evaluates the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

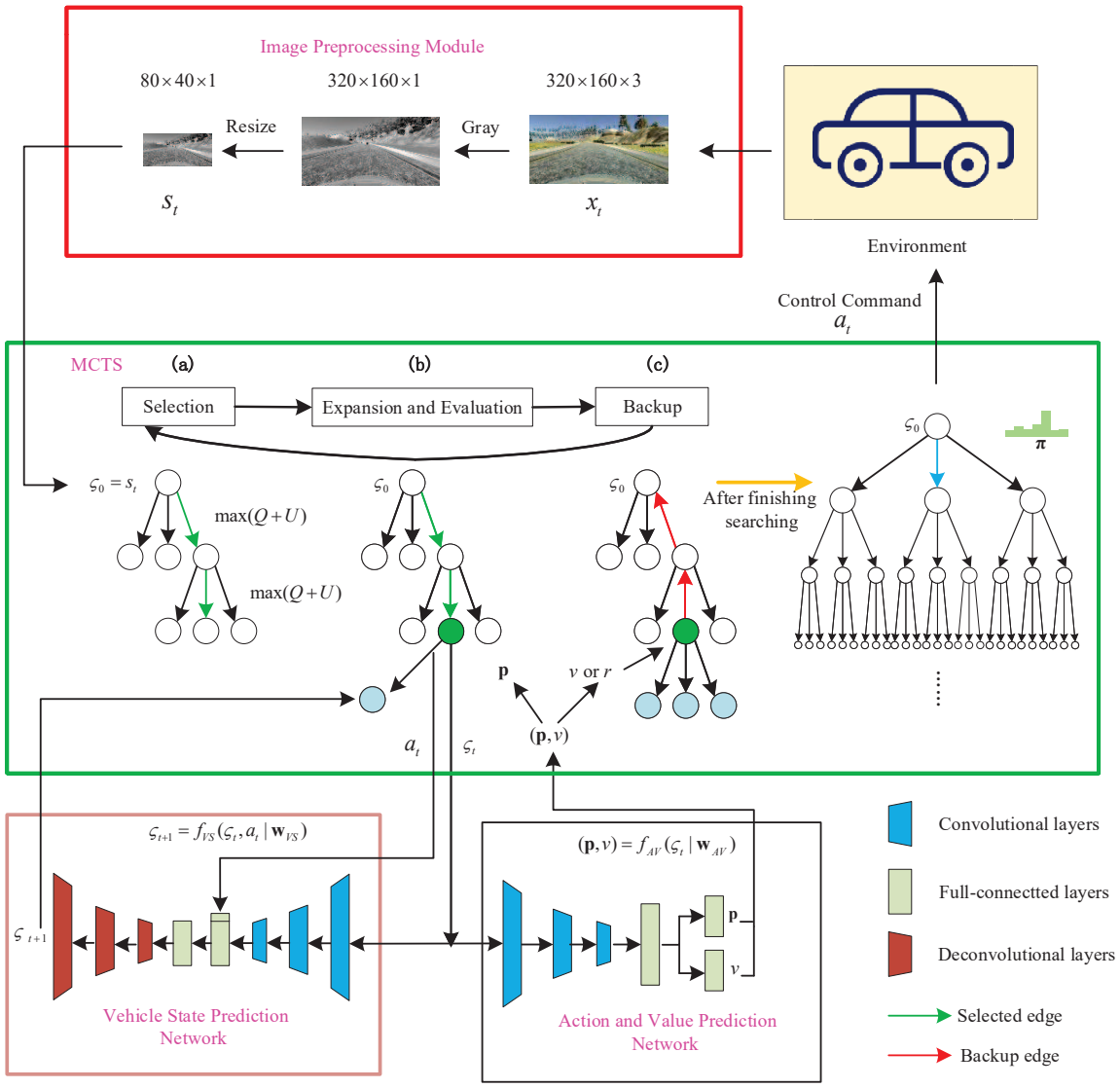


Fig. 5. The whole framework of deep-MCTS based autonomous driving control method. The whole framework mainly contains the Image Preprocessing Module, MCTS, Vehicle State Prediction network, Action and Value Prediction network and Environment.

deviation of the vehicle from the lines being tracked or the one that evaluates the possibility of the vehicle to collide into an obstacle.

The action value function is represented by a neural network $Q(s, a; \theta)$ where the parameters θ can be learned by iteratively minimizing a sequence of loss functions, with the ith loss function defined as

$$L_i(\theta_i) = \mathbb{E}\left(r + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2 \quad (5)$$

Here, s_{t+1} is the next state after state s.

C. Deep-MCTS based Autonomous Driving Control Framework

In this section, we briefly describe the deep-MCTS based autonomous driving control framework shown in Fig. 5, which consists of the Image Preprocessing Module, MCTS, Vehicle State Prediction (VSP) network $f_{VS}(\cdot)$ [55], [56], Action and Value Prediction (AVP) network $f_{AV}(\cdot)$ and the environment.

In this framework, the simulator sends the RGB image x_t captured by the vehicle's camera to the image preprocessing module, which generates the vehicle's state s_t by compressing the image x_t and then sends s_t to the MCTS. Here, s_t is a gray-scale image. Then MCTS performs the main searching process to derive the control action a_t for the vehicle to take. During the searching process, the VSP network realizes the gray-scale image to gray-scale image prediction, which predicts the future state of the vehicle based on a given control action a_t, and the AVP network predicts the action probabilities and associated reward values based on the current state. These predicted values are used in the expansion and evaluation steps in MCTS. Finally, MCTS generates the control action, and the simulator updates the vehicle's state based on this action. The detailed procedures are summarized in Algorithm II. Here, $\boldsymbol{\pi}$ is the searching probability vector of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

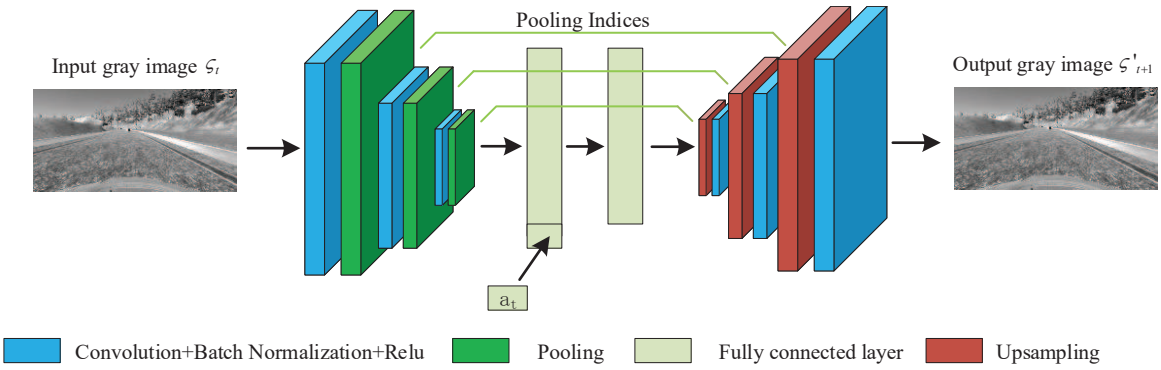


Fig. 6. The architecture of the vehicle state prediction network which consists of three parts: three convolution and pooling layers, two fully connected layers, three upsampling and deconvolution layers.

each node in the searching tree and Q is the mean reward of each node in the searching tree.

D. MCTS Searching Process

Given the current state of the vehicle s_t, the MCTS performs an iterative searching process to derive the control policy π. In particular, the MCTS starts from creating a tree by setting the state of the root node n_0 in the tree to be $\varsigma(n_0) = s_t$. Note that the state of each node n in the tree, denoted as $\varsigma(n)$, is a virtual state that does not impact the state of the vehicle. The MCTS then runs the VSP and AVP networks to expand the tree, where each node in the tree stores the following values

$$\{\varsigma(n), R(n), N(n), a(n), \mathbf{p}(n)\}$$

$R(n)$ is the cumulative reward of node n, $N(n)$ is the visited count, i.e., number of times node n has been visited, $a(n)$ is the control action that causes the transition to node n from its parent node, and $\mathbf{p}(n)$ is the prior selection probability of node n.

Here, the maximum search depth is denoted as M. The detailed searching process is summarized as follows:

1) *Selection* (Fig. 5(a)): Starting from the root node, the best child of the current node n is selected based on the following equation:

$$n'_{best} = \arg\max_{n'}(U(n') + Q(n')) \tag{6}$$

where n' is the child of node n. $U(n')$ is the probability upper confidence bound [57] of child node n' defined as

$$U(n') = c_{puct}\mathbf{p}(n')\frac{\sqrt{\sum_b N(b)}}{N(n')} \tag{7}$$

where c_{puct} is a temperature parameter. The sum $\sum_b N(b)$ represents the total visited count of all the nodes in the level same with the child node n' (including child node n'). $Q(n')$ is the mean reward value of node n' defined as

$$Q(n') = \frac{R(n')}{N(n')} \tag{8}$$

This selection process continues until the leaf node n_L is reached.

2) *Expansion and evaluation* (Fig. 5(b)): If the leaf node n_L is not a terminate node, the search tree is expanded by taking the following actions. Firstly, the AVP network is employed to calculate the prior child node selection probability $\mathbf{p}$ and the current value v by

$$(\mathbf{p}, v) = f_{AV}(\varsigma(n_L)|\mathbf{w}_{AV}), \tag{9}$$

which will be assigned to the children of node n_L. Next, the VSP network is used to predict the next states by

$$\varsigma(n'_L) = f_{VS}(\varsigma(n_L), a|\mathbf{w}_{VS}), \tag{10}$$

where a is the control action that causes the state transition from $\varsigma(n_L)$ to $\varsigma(n'_L)$. At last, each child n'_L of the leaf node n_L is initialized as

$$R(n'_L) = 0, N(n'_L) = 0, a(n'_L) = a, \mathbf{p}(n'_L) = p_{n'} \tag{11}$$

where $p_{n'}$ is the prior selection probability of n'_L and is an element of vector $\mathbf{p}(n'_L)$.

3) *Backup* (Fig. 5(c)): In this step, the values and visited count is propagated back to the root node to update each node's statistics by

$$\begin{aligned} R(n) &= R(n) + v \\ N(n) &= N(n) + 1 \end{aligned} \tag{12}$$

The above three steps are performed iteratively until the max iteration time is reached. The searching probability $\boldsymbol{\pi}(n')$ for each child n' of the root node is then calculated by

$$\boldsymbol{\pi}(n') = \frac{(N(n'))^{\frac{1}{\tau}}}{\sum_b (N(b))^{\frac{1}{\tau}}} \tag{13}$$

where τ is a temperature parameter that controls the level of exploration. All selection probabilities of the child nodes form a selection probability vector $\boldsymbol{\pi}$. The corresponding control action $a(n')$ is selected according to this probability vector $\boldsymbol{\pi}$. Finally, the control action $a(n')$ will be mapped to the control command a_t to be returned. The procedures of the MCTS searching process are summarized in Algorithm III.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

V. Neural Networks and Training Processes

In this section, we provide more details about the two deep neural networks used in the proposed deep-MCTS framework: the VSP network and the AVP network. The training process for each network is also discussed.

The image preprocessing module first receives the original RGB image with the size of $320\times160\times3$, and convert the RGB image to gray-scale one. In order to reduce state space and raise operation efficiency, we compress the gray-scale image to the size of $80 \times 40 \times 1$.

A. VSP Network

1) Network Function: Given the current state ς_t and the control action a_t, the VSP network f_{VS} is employed to predict the next state ς'_{t+1} through the following equation

$$\varsigma'_{t+1} = f_{VS}(\varsigma_t, a_t|\mathbf{w}_{VS}) \tag{14}$$

where $\mathbf{w}_{VS}$ is the parameter of the VSP network to be configured.

2) Network Architecture: As shown in Fig. 6, the VSP network consists of three convolution and pooling layers, two fully connected layers, three upsampling and deconvolution layers. The input is the gray-scale images with a size of $80 \times 40 \times 1$ and the action. The first convolution layer has 16 kernels with a size of $8 \times 8 \times 1$. The second convolution layer has 16 kernels with a size of $4 \times 4 \times 16$. The third convolution layer has 16 kernels with a size of $3 \times 3 \times 16$. Each convolutional layer is followed by a batch normalization layer, Relu and a 2×2 max-pooling. The number of neural units in the first fully connected layer is 801, which combines the action with the extracted feature. The number of neural units in the second fully connected layer is 800. Then, there are four upsampling and deconvolution layers. Each upsampling layer upsamples its input feature map using corresponding pool indices and is followed by a deconvolution layer. The first deconvolution layer has 16 kernels with a size of $3 \times 3 \times 16$. The second convolution layer has 16 kernels with a size of $4 \times 4 \times 16$. The third convolution layer has 1 kernel with a size of $8 \times 8 \times 16$. The last deconvolution layer predict the next state ς'_{t+1}.

3) Network Training: Suppose s_t and a_t are available to train the VSP network. Because the VSP network is used to predict the state of the next moment ς'_{t+1}, the corresponding label is the next real state s_{t+1}. The loss function is given by

$$\mathcal{L}_{VS} = \frac{1}{B_1}\sum_{t=0}^{B_1}(\| s_{t+1} - \varsigma'_{t+1}\|_2) \tag{15}$$

where B_1 is the batch size.

B. AVP Network

1) Network Function: The AVP network is employed to predict the action selection probabilities $\mathbf{p_t}$ and the value v_t of current state ς_t by

$$(\mathbf{p}_t, v_t) = f_{AV}(\varsigma_t|\mathbf{w}_{AV}) \tag{16}$$

where $\mathbf{w}_{AV}$ is the parameter of the AVP network to be configured.

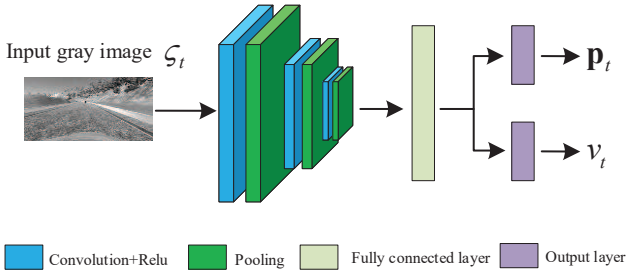


Fig. 7. The architecture of action and value prediction network, which contains 4 fully connected layers.

Algorithm IV Deep-MCTS training process

```
function Train Networks()
  Initialize simulator environment Env
  Initialize two networks f_VS, f_AV
  Initialize two data buffers D_1, D_2
  while within max episode do
    x_t ←get image from Env
    s_t ← Image Preprocessing Module(x_t)
    a_t, π ←MCTS Search (s_t, f_VS, f_AV)
    r ← Env executes a_t and returns reward
    Store {s_t, a_t} into D_1
    Store {s_t, π, r} into D_2
    Sample data from D_1 to train f_VS
    Sample data from D_2 to train f_AV
```

2) Network Architecture: As shown in Fig. 7, the AVP network consists of three convolution and pooling layers, and two fully connected layers. The input is the gray-scale images with a size of $80 \times 40 \times 1$. The first convolution layer has 32 kernels with a size of $8 \times 8 \times 1$. The second convolution layer has 64 kernels with a size of $4 \times 4 \times 32$. The third convolution layer has 64 kernels with a size of $3 \times 3 \times 64$. Each convolutional layer is followed by a batch normalization layer, a Relu activation layer and a 2×2 max-pooling. The number of neural units in the first fully connected layer is 128. The last fully connected layer contains two parts, the action prediction sub-layer with 11 neural units and the value evaluation sub-layer with a single neural unit.

3) Network Training: The data used for training the AVP network include $\{\mathbf{p}_t, v_t, v_{t+1}\}$, corresponding ground-truth action selection probabilities and value of current state $\{\boldsymbol{\pi}_t, r+\gamma v_{t+1}\}$. The loss function is given by

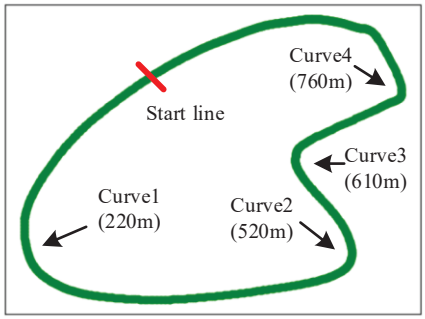
$$\mathcal{L}_{AV} = \frac{1}{B_2}\sum_{t=1}^{B_2}[-\boldsymbol{\pi}_t^T \log \mathbf{p}_t + (r + \gamma v_{t+1} - v_t)^2] \tag{17}$$

where B_2 is the batch size and r is the reward returned by the environment.

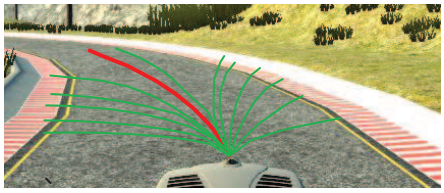
C. Training Process for the Deep-MCTS

To train the proposed deep-MCTS of two deep neural networks, the training data are collected through the following procedure (see Algorithm IV). First, the simulator is initialized

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology



(a) The race road with four sharp curves in the simulator USS.



(b) The predicted driving maneuver trajectories of proposed deep-MCTS where the red line is the best trajectory in the simulator USS.

Fig. 8. The race road and the predicted driving maneuver trajectories.

TABLE II
THE HYPERPARAMETERS OF DEEP-MCTS.

Symbol	Description	value
τ	The temperature parameter in equation (13)	1.0
c_{puct}	The temperature parameter in equation (7)	0.1
B_1	The training batch size of the VSP network	20
B_2	The training batch size of the AVP network	20
γ	The discounting rate in equation (17)	0.9

and two data buffers $\mathcal{D}_1$, $\mathcal{D}_2$ are created, with the first one used for storing the training data for the VSP networks and the second one used for storing the training data for the AVP network. We then run multiple episodes to collect the data. In particular, in each episode, the MCTS produces $a_t, \boldsymbol{\pi}$ based on the current state s_t, and the simulator executes the control command a_t to generate the next state s_{t+1} and get the reward r from the environment. After that, (s_t, a_t, s_{t+1}) is stored into buffer $\mathcal{D}_1$ and $\{s_t, \boldsymbol{\pi}, r\}$ is stored into buffer $\mathcal{D}_2$. Finally, the VSP network f_{VS} is trained with data from $\mathcal{D}_1$, and the AVP network f_{AV} is trained with data from $\mathcal{D}_2$. Consequently, for the VSP network, the inputs include s_t, a_t and the corresponding label is s_{t+1}. For the AVP network, the input is s_t and the corresponding label includes $\boldsymbol{\pi}$ and r.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we conduct experimental studies to evaluate the performance of the proposed method by comparing it with existing RL-based and IL-based autonomous driving methods. The experimental results are obtained by running simulations using the USS and the Torcs with the hyperparameters of the deep-MCTS configured according to TABLE II. The learning rate for the VSP network is set to 0.00001 and the learning rate for the AVP network is set to 0.0001. The iteration number and search depth M of MTCS in training is set to 1000 and

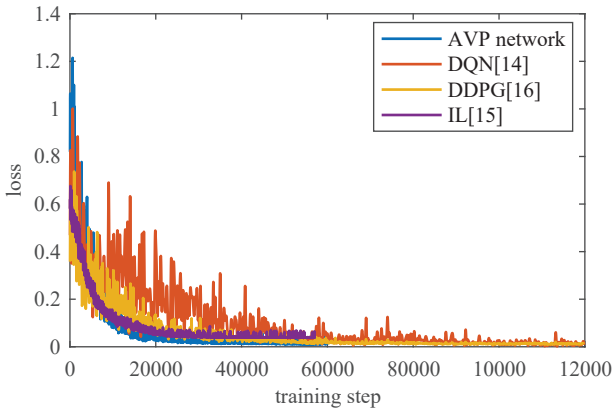


Fig. 9. The training loss curves of AVP network, DQN and IL.

TABLE III
THE MCES AND MDCS OF DEEP-MCTS, DQN, DDPG AND IL METHOD IN THE SIMULATOR USS.

method	deep-MCTS	DQN	DDPG	IL
MCE	**0.0365**	0.1083	0.0564	0.0403
MDC	**0.2505**	0.6118	0.5026	0.3557

8, respectively and the iteration number and search depth of MTCS in testing is set to 100 and 4, respectively. The reward returned by the simulator is defined as

$$r = \begin{cases} -1, & \text{if collision occurs} \\ 0.1, & \text{otherwise} \end{cases} \tag{18}$$

These hyperparameters are all obtained by the trading off between accuracy and time complexity through trial and error in their possible values after grid searching in the given range.

A. Performance Evaluation Metrics

The performance of the proposed approach is evaluated from two aspects: the stability of driving control and the stability of driving trajectory. To measure the stability of driving control, we adopt the mean continuity error (MCE) [58] metric defined as

$$MCE = \sqrt{\frac{1}{D-1}\sum_{t=1}^{D-1}(\partial_{t+1} - \partial_t)^2}, \tag{19}$$

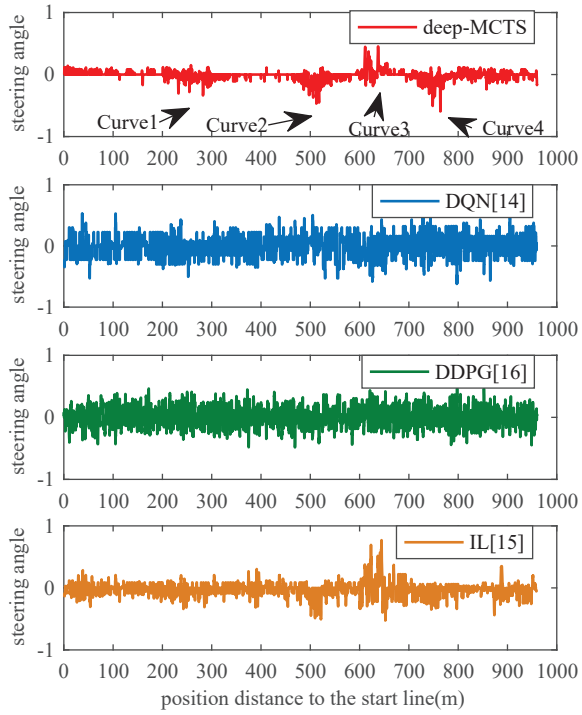
where D is the total number of control commands during testing. Here, ∂_t is the control command at time t. A smaller MCE indicates smoother changes in control values and thus is a more stable controller. To evaluate the stability of driving trajectory, the mean of the deviation to track center (MDC) [14] is adopted, which is defined as

$$MDC = \frac{1}{D}\sum_{t=1}^{D} d_t, \tag{20}$$

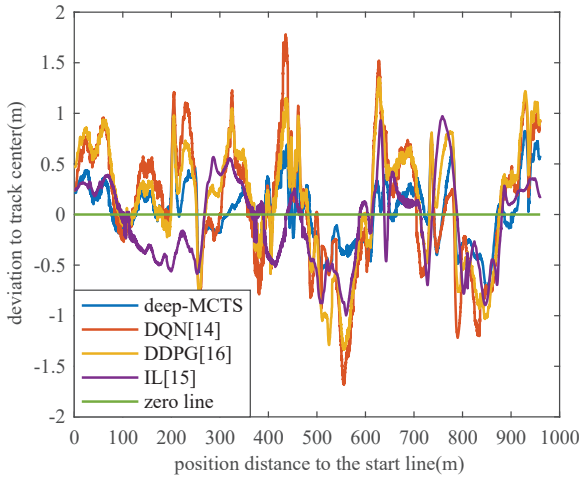
where d_t is the deviation of the vehicle to the track center at time step t. A smaller MDC indicates more stable driving maneuver.

We select DQN [14], DDPG [16] and IL [15] as the benchmarks for comparison. a) DQN, in which input and output

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology



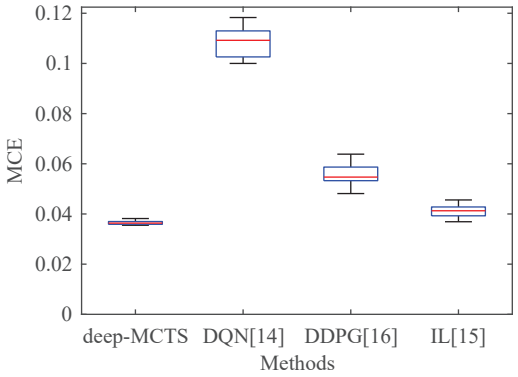
(a) The steering controls of deep-MCTS, DQN, DDPG and IL method in the simulator USS.



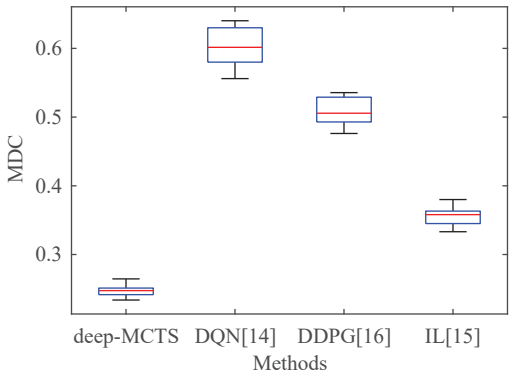
(b) The distances to center of deep-MCTS, DQN, DDPG and IL method in the simulator USS.

Fig. 10. The experimental results of deep-MCTS, DQN, DDPG and IL method in the simulator USS. The deep-MCTS method has the best performance.

are successive images and the control action, respectively. Its reward is calculated by the distance to track center of the current state. b) DDPG, in which the input, output and reward configurations are the same as DQN. It has a much more complicated network architecture than DQN and integrates the advantages of DQN and policy gradient [59]. c) IL, which is a supervised learning method by imitating the human's driving experiences. Its input and output are a single image and the control action, respectively.



(a) The MCEs of deep-MCTS, DQN, DDPG and IL method in the simulator USS.



(b) The MDCs of deep-MCTS, DQN, DDPG and IL method in the simulator USS.

Fig. 11. The MCEs and MDCs of deep-MCTS, DQN, DDPG and IL method in the simulator USS.

B. Experimental Results on the simulator USS without obstacles

In this subsection, we compare the proposed deep-MCTS autonomous driving method with the DQN-based method in [14], DDPG-based method in [16] and IL-based method in [15] proposed by Nvidia, all of which are also evaluated using the same race road in the simulator USS. The maximum speed of the vehicle is limited to 40km/h. All the four methods can successively control the vehicle to run along the race road with images as the only input. However, their performances in training efficiency, MCE and MDC are quite different and the details will are described as following:

1) The training loss: The AVP network, DQN and IL all output the action selection policy, but they differ significantly in the convergence speed. As shown in Fig. 9, the training losses of AVP network, DQN, DDPG and IL converge at the training step 40000, 80000, 70000 and 30000, respectively. Hence, the AVP network improves 50.0% compared to DQN, 42.9% compared to DDPG in training efficiency and only loses 33.3% in training efficiency compared to IL. In addition, during the training process, the loss of DQN fluctuates over a much larger range than those of the AVP network and IL, indicating a worse convergence performance.

2) The stability of driving control: As shown in Fig. 10(a), the control values generated by DQN demonstrate a large variance, compared with the DDPG, deep-MCTS and IL. In

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

Track 1 Track 2 Track 3

Fig. 12. The testing tracks in the simulator Torcs. From left to right, the curves of the tracks will increase gradually.

TABLE IV
THE MCES AND MDCS OF DEEP-MCTS IN THE SIMULATOR TORCS.

track	Track 1	Track 2	Track 3
MCE	0.0412	0.0465	0.0507
MDC	0.3125	0.3652	0.4023

addition, the MCEs of deep-MCTS, DQN, DDPG and IL are 0.0365, 0.1083, 0.0564 and 0.0403 respectively as shown in TABLE III. Hence, the deep-MCTS improves by 66.30% in the stability of driving control compared to DQN, 35.28% compared to DDPG and 9.43% compared to IL.

3) The stability of driving trajectory: As shown in Fig. 10(b), when the vehicle enters the sharp curves, its deviation to the track center increases. Note that the optimization goal of a controller is to keep the vehicle close to the track center, leading to smaller MDC values. The MDCs of the deep-MCTS, DQN, DDPG and IL are 0.2505, 0.6118, 0.5026 and 0.3557 respectively as shown in TABLE III. Hence, the deep-MCTS improves by 59.06% in the stability of driving trajectory compared to DQN, 50.16% compared to DDPG and 29.58% compared to IL.

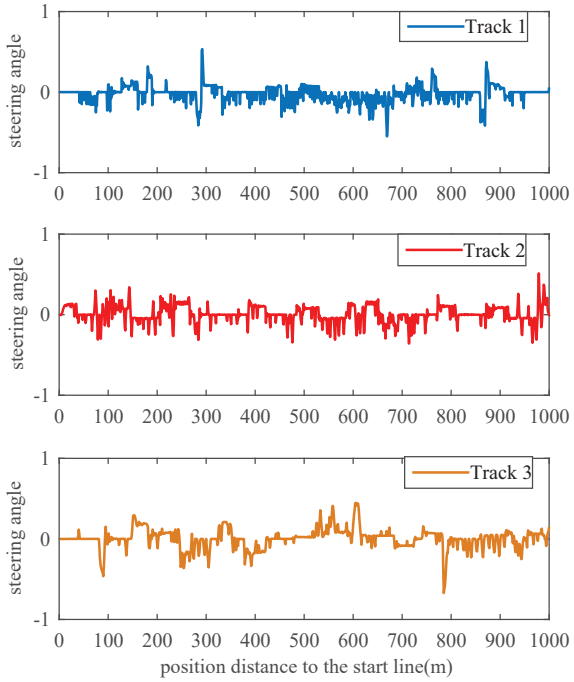
Besides, we also perform the single way ANOVA [60] for the four methods in terms of MCE and MDC. The MCEs and MDCs of the four methods are shown in Fig. 11. In particular, the single way ANOVA finds that the differences among the MCEs of the four methods are significantly different, where

$$[F(3, 36) = 413.81, p < 0.001, eta\_Sqr = 0.97].$$

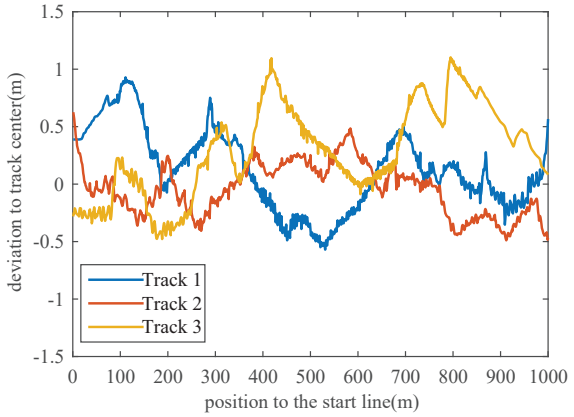
Similarly, the single way ANOVA indicates that the MDCs of the four methods are also significantly different, where

$$[F(3, 36) = 669.68, p < 0.001, eta\_Sqr = 0.98].$$

Moreover, a set of unequal variance t tests show that the deep-MCTS method has significantly smaller MCE and MDC $[p < 0.001]$ than the other three methods.



(a) The steering controls of deep-MCTS in the three tracks of the simulator Torcs.



(b) The distances to track center of deep-MCTS in the three tracks of the simulator Torcs.

Fig. 13. The experimental results of deep-MCTS, DQN, DDPG and IL method in the three tracks of the simulator Torcs. The deep-MCTS method has the best performance.

C. Experimental Results on the simulator Torcs without obstacles

To verify that the proposed method can be applied to various scenarios, we also use the simulator Torcs to better illustrate the performance of the proposed algorithm. There are totally three different tracks evaluated in this study as shown in Fig. 12. From left to right, the curves of the tracks increase gradually. All the three tracks have shadows and different ambient lights. Besides, the maximum speed of the vehicle is limited to 60km/h.

The steering angle and deviation to track center curves are shown in Fig. 13. The deep-MCTS method can successively control the vehicle to run along the track center with driver-view images as input in all three tracks and it achieves similar performances in the three tracks. The MCEs of the deep-MCTS method in the three tracks are 0.0412, 0.0465 and 0.0507 respectively as shown in TABLE IV. The MDCs of the deep-MCTS method in the three tracks are 0.3125, 0.3652, 0.4023 respectively as shown in TABLE IV. We can conclude that as the curve of the track increases, the performance of the deep-MCTS method degrades slightly.

Hence, the experimental results prove that our proposed method can solve more complex autonomous driving problems with shadows, different ambient lights and high vehicle speed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2991584, IEEE Transactions on Vehicular Technology

Fig. 14. The obstacles on the road of the simulator USS.

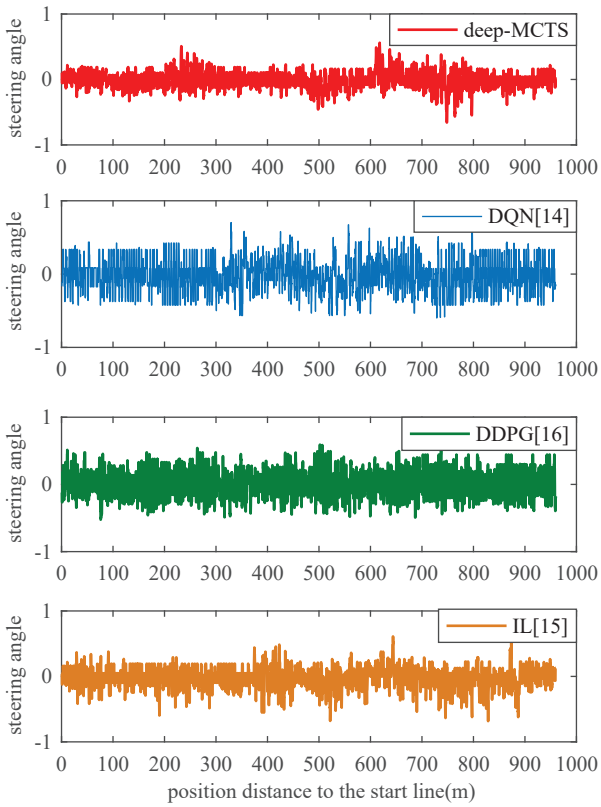


Fig. 15. The steering controls of deep-MCTS, DQN, DDPG and IL method in the simulator USS with obstacles. The deep-MCTS method has the best performance.

D. Experimental Results on the simulator USS with obstacles

In this study, we add dozens of obstacles on the race road in the simulator USS as shown in Fig. 14. The maximum speed of the vehicle is also limited to 40km/h. We again compare the proposed deep-MCTS autonomous driving method with the DQN-based method, DDPG-based method and IL-based method. Because of obstacles, the vehicle can't always run along the track center. Hence, we only use MCE to evaluate the performance of the four methods shown in Fig. 15. The MCEs of deep-MCTS, DQN, DDPG and IL are 0.0673, 0.1581, 0.1120 and 0.0769 respectively. Hence, the deep-MCTS improves by 57.43% in the stability of driving control compared to DQN, 39.91% compared to DDPG and 12.48% compared to IL.

VII. Conclusion

In this paper, we propose a reinforcement learning based deep-MCTS algorithm for vision-based autonomous driving control. Different from traditional autonomous driving algorithms, the driver-view images captured by the camera onboard of the autonomous vehicle is the only input and the deep-MCTS algorithm can learn how to control the vehicle without any human knowledge. The deep-MCTS algorithm can predict the driving maneuvers by performing virtual driving simulations, which improve the stability of steering control and the stability of driving trajectory. Compared to the existing methods, the deep-MCTS algorithm shows 50.0%, 66.30% and 59.06% improvement in training efficiency, stability of steering control and stability of driving trajectory, respectively. In the future work, we will consider migrating our proposed method from the virtual to the real life.

References

[1] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 163–168.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[3] M. Maurer, J. C. Gerdes, B. Lenz, H. Winner *et al.*, "Autonomous driving," *Berlin, Germany: Springer Berlin Heidelberg*, vol. 10, pp. 978–3, 2016.

[4] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1013–1020.

[5] R. P. D. Vivacqua, M. Bertozzi, P. Cerri, F. N. Martins, and R. F. Vassallo, "Self-localization based on visual lane marking maps: An accurate low-cost approach for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 582–597, 2018.

[6] X. Yue, B. Wu, S. A. Seshia, K. Keutzer, and A. L. Sangiovanni-Vincentelli, "A lidar point cloud generator: from a virtual world to autonomous driving," in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ACM, 2018, pp. 458–464.

[7] R. W. Wolcott and R. M. Eustice, "Robust lidar localization using multiresolution gaussian mixture maps for autonomous driving," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 292–319, 2017.

[8] K. Banerjee, D. Notz, J. Windelen, S. Gavarraju, and M. He, "Online camera lidar fusion and object detection on hybrid data for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1632–1638.

[9] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[10] J. Dou, J. Fang, T. Li, and J. Xue, "Boosting cnn-based pedestrian detection via 3d lidar fusion in autonomous driving," in *International Conference on Image and Graphics*. Springer, 2017, pp. 3–13.

[11] P. Li, T. Qin *et al.*, "Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 646–661.

[12] L. Li, K. Ota, and M. Dong, "Humanlike driving: empirical decision-making system for autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 8, pp. 6814–6823, 2018.

[13] R. Mahdavian and R. D. Martinez, "Ignition: An end-to-end supervised model for training simulated self-driving vehicles," *arXiv preprint arXiv:1806.11349*, 2018.

[14] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement learning and deep learning based lateral control for autonomous driving," *arXiv preprint arXiv:1810.12778*, 2018.

[15] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[16] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *arXiv preprint arXiv:1811.11329*, 2018.

[17] O. K. Oyedotun and A. Khashman, "Deep learning in vision-based static hand gesture recognition," *Neural Computing and Applications*, vol. 28, no. 12, pp. 3941–3951, 2017.

[18] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: a brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[19] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. M. Lopez, "Vision-based offline-online perception paradigm for autonomous driving," in *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE, 2015, pp. 231–238.

[20] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "Learning real manipulation tasks from virtual demonstrations using lstm," *arXiv preprint arXiv:1603.03833*, 2016.

[21] M. J. Frank and E. D. Claus, "Anatomy of a decision: striato-orbitofrontal interactions in reinforcement learning, decision making, and reversal." *Psychological review*, vol. 113, no. 2, p. 300, 2006.

[22] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7011–7024, 2019.

[23] J. Chen, Z. Wei, S. Li, and B. Cao, "Artificial intelligence aided joint bit rate selection and radio resource allocation for adaptive video streaming over f-rans," in press, doi: 10.1109/MWC.001.1900351.

[24] A. Taya, T. Nishio, M. Morikura, and K. Yamamoto, "Deep-reinforcement-learning-based distributed vehicle position controls for coverage expansion in mmwave v2x," *arXiv preprint arXiv:1810.11211*, 2018.

[25] A. Yu, R. Palefsky-Smith, and R. Bedi, "Deep reinforcement learning for simulated autonomous vehicle control," *Course Project Reports: Winter*, pp. 1–7, 2016.

[26] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 285–292.

[27] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[28] S. Du, H. Guo, and A. Simpson, "Self-driving car steering angle prediction based on image recognition," *arXiv preprint arXiv:1912.05440*, 2019.

[29] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.

[30] D. Wang, C. Devin, Q.-Z. Cai, F. Yu, and T. Darrell, "Deep object-centric policies for autonomous driving," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8853–8859.

[31] R. Michelmore, M. Wicker, L. Laurenti, L. Cardelli, Y. Gal, and M. Kwiatkowska, "Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control," *arXiv preprint arXiv:1909.09884*, 2019.

[32] J. Bi, T. Xiao, Q. Sun, and C. Xu, "Navigation by imitation in a pedestrian-rich environment," *arXiv preprint arXiv:1811.00506*, 2018.

[33] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, vol. 19, no. 9, p. 2064, 2019.

[34] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers, "Learning to drive using inverse reinforcement learning and deep q-networks," *arXiv preprint arXiv:1612.03653*, 2016.

[35] M. Vitelli and A. Nayebi, "Carma: A deep reinforcement learning approach to autonomous driving," 2016.

[36] S. Mazumder, B. Liu, S. Wang, Y. Zhu, L. Liu, and J. Li, "Action permissibility in deep reinforcement learning and application to autonomous driving," 2018.

[37] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2070–2075.

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

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[41] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[42] W. Wang, J. Shen, and L. Shao, "Video salient object detection via fully convolutional networks," *IEEE Transactions on Image Processing*, vol. 27, no. 1, pp. 38–49, 2018.

[43] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[44] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[45] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[47] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[48] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *Computer Science*, 2013.

[49] R. Rai, *Socket. IO Real-time Web Application Development*. Packt Publishing Ltd, 2013.

[50] R. A. Howard, "Dynamic programming and markov processes." 1960.

[51] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[52] D. J. White, "A survey of applications of markov decision processes," *Journal of the operational research society*, vol. 44, no. 11, pp. 1073–1096, 1993.

[53] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating mcts modifications in general video game playing," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 107–113.

[54] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing." in *IJCAI*, 2016, pp. 2514–2520.

[55] L. Du, W. Zixuan, L. Wang, Z. Zhao, F. Su, B. Zhuang, and B. Nikolaos V, "Adaptive visual interaction based multi-target future state prediction for autonomous driving vehicles," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2019.

[56] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *arXiv preprint arXiv:1911.08265*, 2019.

[57] C. D. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, 2011.

[58] Y. Chen, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan, "Learning on-road visual control for self-driving vehicles with auxiliary tasks," *arXiv preprint arXiv:1812.07760*, 2018.

[59] J. Peters and J. A. Bagnell, "Policy gradient methods," *Encyclopedia of Machine Learning*, vol. 5, no. 11, pp. 774–776, 2010.

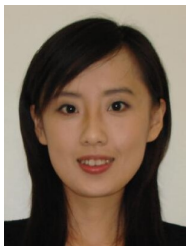
[60] G. Prabhakar, A. Ramakrishnan, M. Madan, L. R. D. Murthy, V. K. Sharma, S. Deshmukh, and P. Biswas, "Interactive gaze and finger controlled hud for cars," *Journal on Multimodal User Interfaces*, Nov 2019. [Online]. Available: https://doi.org/10.1007/s12193-019-00316-9

Jienan Chen (S'10, M'14) received the B.S. and Ph.D degrees in communication systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China in 2007 and 2014, respectively. He also worked in the School of Electrical and Computer Engineering, University of Minnesota, Minneapolis, as a visiting scholar in 2012 and 2014 and then a postdoctoral scholar at the University of North Texas. His current research interests include VLSI circuit designs, low-power circuit designs, and stochastic computation-based system designs. He is currently an Associate Professor with the National Key Laboratory of Science and Technology on Communications at the UESTC. He also served as symposium chair for Globalsip and TPC member for Globecom and ICC. His research interests are on machine learning-based signal processing, artificial intelligence for networking and circuit-system design.

Cong Zhang received B.E. degree in the School of Information and Communication Engineering from University of Electronic Science and Technology of China (UESTC), in 2018. Now he is pursuing the M.Sc. degree in the National Key Laboratory of Science and Technology on Communications at the UESTC. His current research interests include deep learning, deep reinforcement learning and self-driving.

Jin Ting received B.E. degree in the School of Information Science and Technology from Dalian Maritime University, in 2019. Now he is pursuing the M.Sc. degree in the National Key Laboratory of Science and Technology on Communications at the University of Electronic Science and Technology of China. His current research interests include computer vision, deep reinforcement learning and self-driving.

Junfei Xie (S'13, M'16) received the B.S. degree in electrical engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science and engineering from the University of North Texas (UNT), Denton, TX, USA, in 2013 and 2016, respectively. She was an Assistant Professor with the Department of Computing Sciences, Texas A&M University-Corpus Christi (TAMUCC). She is currently an Assistant Professor with the Electrical and Computer Engineering Department, San Diego State University. Her current research interests include large-scale dynamic system design and control, managing and mining spatiotemporal data, unmanned aerial systems, distributed computing, airborne computing, complex information systems, air traffic flow management, and so on.



Yan Wan (S'08, M'09, SM'17) is currently an Associate Professor with the Electrical Engineering Department at the University of Texas, Arlington. She received the Ph.D. degree in electrical engineering from Washington State University in 2008 and then did postdoctoral training at the University of California, Santa Barbara. Her research interests lie in the modeling and control of large-scale dynamical networks, cyber-physical system, stochastic networks, learning control, networking, uncertainty analysis, algebraic graph theory, and their applications to UAV networking, UAV traffic management, epidemic spread, complex information networks, and air traffic management. Dr. Wans research has led to over 170 publications and successful technology transfer outcomes. She has received prestigious awards, including the NSF CAREER Award, RTCA William E. Jackson Award, U.S. Ignite and GENI demonstration awards, IEEE WCNC and ICCA Best Paper Award, and Tech Titan of the Future-University Level Award.