Vehicle Control with Prediction Model Based Monte-Carlo Tree Search

Timothy Ha[1], Kyunghoon Cho[1], Geonho Cha[1], Kyungjae Lee[1], and Songhwai Oh[1]

Abstract— **In this paper, we propose a model-based Monte-Carlo Tree Search (model-based MCTS) algorithm for the vehicle planning and control problem. While driving vehicles, we need to predict the future states of other vehicles to avoid collisions. However, because the movements of the vehicles are determined by the intentions of the human drivers, we need to use a prediction model which captures the intention of the human behavior. In our model-based MCTS algorithm, we introduce a neural-network-based prediction model which predicts the behaviors of the human drivers. Unlike conventional MCTS algorithms, our method estimates the rewards and Q-values based on intention-considering future states, not from the pre-defined deterministic models or self-play methods. For the evaluation, we use environments where the other vehicles follow the trajectories of pre-collected driver datasets. Our method shows novel results in the collision avoidance and success rate of the driving, compared to other reinforcement learning and imitation learning algorithms.**

I. Introduction

With the advances of the reinforcement learning, many neural-network-based algorithms have shown successful results in robot planning problems [1], [2]. Also, recent reinforcement learning algorithms have focused on the autonomous driving problems. However, in autonomous driving, the safety issues become more important because it can lead to a serious accident when a collision occurs.

One of the most necessary things for safe driving is to consider the movement of nearby vehicles. When we drive a car, we must keep an eye on the nearby vehicles, predict the future trajectories, and avoid collisions through them. This is not a simple problem because a human-driven car can move in a variety of ways. The vehicles can accelerate or decelerate, or turn left or right, depending on the intention of the other drivers. Therefore, it is needed to consider the intention of the human drivers when we predict the future trajectories of the other vehicles.

For the vehicle planning and control problem, the autonomous vehicle also needs to consider the intention of the human drivers to predict the futures. In this work, we propose a model-based Monte-Carlo Tree Search (model-based MCTS) algorithm, which considers the future trajectories of the adjacent human drivers. In conventional MCTS methods, they assume that the next state of the agent can be calculated from a pre-defined deterministic model, or sampled by a self-play method. However, when it is applied to autonomous driving problem, this method can be inaccurate and cannot reflect the intentions of actual vehicle movements. In our method, we incorporate a prediction model to take into account the intentions and actual movements of adjacent human-driven vehicles.

For the prediction model, we train our network with a highway driving datasets [3], in which the vehicles change their lanes and speeds. Because the vehicles do not move constantly, the network needs to capture the intention of the vehicle for the prediction. We show that our prediction model can successfully predict the future states and trajectories of the vehicles, and also is effective for planning algorithms.

Along with this prediction model, we use MCTS algorithm for the entire planning process. Our prediction model is used to calculate the next states and rewards of each nodes. For the MCTS search, we leverage a deep-neural-network-based reinforcement learning algorithm [4] which have been proven as effective in other environments. We get a policy and value network by training a reinforcement learning algorithm. Then we combine this pre-trained networks with the MCTS algorithm [5], [6]. Our prediction model is used in the extension steps of the MCTS algorithm, where the future states are estimated.

In experiments, we show that our method is effective in the planning problem of the autonomous vehicle, and that it avoids collisions by considering the future state of the other cars. Our method shows the state-of-the-art performance in the success rate of the driving in the environment.

The remainder of this paper is organized as follows: we explain the entire process of the predicting and training method in Section IV. The details of our algorithm is explained in Section V. All the results of our experiments are shown in Section VI, and we conclude the paper in Section VII.

II. Related Work

There are many works about predicting the future trajectories of the vehicle. Wiest et al. [7] suggests a Gaussian mixture model as a probabilistic modeling to predict future trajectories of vehicles. The works of Tomar et al. [8], Ding et al. [9], and Kim et al. [10] predict the future trajectories of vehicles using a neural network model.

Monte-Carlo Tree Search (MCTS) [5] is a planning algorithm which simulates the future states. MCTS is widely used in a number of problems, such as chess [11], Go [6], and curling [12]. However, for the multi-agent cases, the simulation processes of previous works have mainly

[1]The authors are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul, Republic of Korea (e-mail: {timothy.ha, kyunghoon.cho, geonho.cha, kyungjae.lee}@rllab.snu.ac.kr, songhwai@snu.ac.kr). This work was supported by the ICT R&D program of MSIT/IITP (No. 2019-0-01309, Development of AI Technology for Guidance of a Mobile Robot to its Goal with Uncertain Maps in Indoor/Outdoor Environments), and by Samsung Electronics Co., Ltd. (No. 0115-20190007).

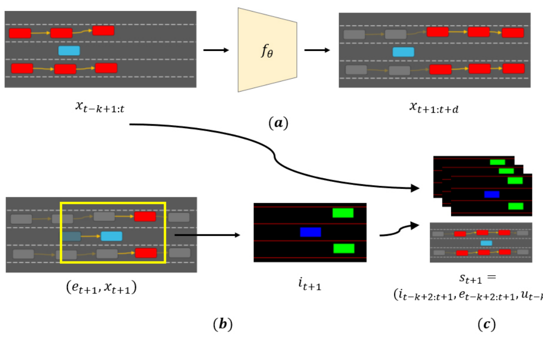

Fig. 1: The overall process of the prediction model T_θ. (a) The previous vehicle states $x_{t-k+1:t}$ is entered into the network f_θ, and the results become the predicted states $x_{t+1:t+d}$. (b) With the next ego state e_{t+1} and a prediction result of x_{t+1}, the next bird-eye view image i_{t+1} is constructed. Here, e_{t+1} is calculated from a unicycle model with ego state e_t and input a_t. (c) The next state $s_{t+1} = (i_{t-k+2:t+1}, e_{t-k+2:t+1}, x_{t-k+2:t+1})$ comes out from the results of i_{t+1}, e_{t+1}, x_{t+1}, and the previous inputs $i_{t-k+2:t}$, $e_{t-k+2:t}$, $x_{t-k+2:t}$.

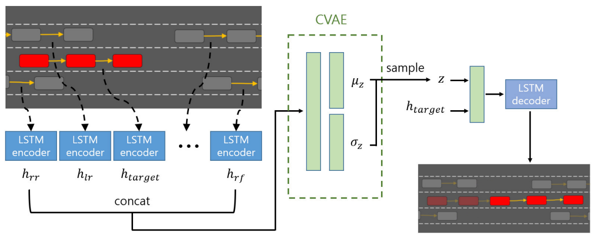

Fig. 2: The network architectures for the prediction network f_θ. To predict one vehicle trajectory, the network f_θ considers the trajectories of neighboring vehicles. Each neighbor trajectories and the target vehicle trajectory are encoded by the same LSTM encoder, and then concatenated. The concatenated hidden vectors are entered into CVAE architecture, and a random vector z is sampled from the CVAE. This random vector z and the hidden encoding h_{target} run through a MLP layer, and are used as the hidden state for LSTM decoder. LSTM decoder outputs the predicted trajectory for the target vehicle. The network f_θ processes all of each trajectories found in $x_{t-k+1:t}$ one by one, and returns the final output $x_{t+1:t+k}$.

performed through self-play, where the actions of the other agents are determined by the same policy with one of the ego agent. The uses of prediction about the future states of the other agent have not been fully studied yet. Stern et al. [13] predict the strength of the players using Bayesian inference, but do not predict the future state itself.

There are many planning and control algorithms which are based on prediction models and search methods. Brechtel et al. [14] assume a probabilistic model with a Dynamic Bayesian Networks (DBN), and make decisions by sampling the future states and rewards. However, their prediction model infers the future states using pre-defined and pre-structured probabilistic model, while our work uses a prediction model which is trained with an actual human driving dataset. Cho et al. [15] suggest prediction model which considers the interaction between the vehicles, and control the vehicle with model predictive control formulation and mixed integer quadratic programming. In this work, we make use of the same prediction model as [15]. However, we control the vehicle using MCTS based planning algorithm which leverages a pre-trained policy network, not using simple quadratic programming method.

III. Notations and Environments

The target of our algorithm is the multi-agent driving environment where there are driving vehicles near the ego vehicle. In our problem setting, s_t is the state, a_t is the action, and r_t is the reward. The state s_t consists of three components: an image i_t which represents the bird-eye view of the ego vehicle, a vector e_t which represents the position and velocity of the ego-car, and a set of vectors x_t which represents the position and velocity of neighborhood vehicles. The image i_t has three channels, and each channel represents the lane marking, the neighborhood vehicles, and the ego vehicle. Here, the neighborhood vehicles and the ego vehicle are represented as rectangular shapes. The state $s_t = (i_{t-k+1:t}, e_{t-k+1:t}, x_{t-k+1:t})$ consists of k previous histories of the images, and vectors.

In the environment we use, the ego vehicle is controlled with a two-dimensional input vector, where each component represents the acceleration or braking of the car u_{accel} and the changes of the steering angle u_{steer}. From the control input, the next position and the velocity of the ego vehicle are determined by using a unicycle model. In this work, we discretize the action space for each the acceleration or braking component, and the steering component, which means that each action a_t we perform is one of the predefinded control vectors (u_{accel}, u_{steer}).

The agent of the ego vehicle gets a reward r_t when it takes action a_t in state s_t. The reward r_t is determined by a reward function $r_t = r(s_t, a_t, s_{t+1})$. Here, we define the reward function using the cost functions which are used in [16]. Each cost we use depends on the the next state s_{t+1}. $C_{prox}(s_{t+1})$ is a cost representing the proximity between the ego vehicle and the neighboring vehicles. $C_{lane}(s_{t+1})$ is a cost representing the area where the ego vehicle overlaps the lane marking. C_{prox} and C_{lane} are calculated from the i_{t+1} which is a component of the state s_{t+1}. $C_{success}(s_{t+1})$ is a indicator function which becomes 1 when the ego vehicle successfully reaches to the goal and 0 when it does not. Similarly, $C_{failure}(s_{t+1})$ is a indicator function which becomes 1 when the ego vehicle collides and 0 when it does not. The total reward function r is determined by a weighted sum of C_{prox}, C_{lane}, $C_{success}$, and $C_{failure}$.

IV. Prediction Model for Planning

In this section, we explain the prediction model which is used for our model-based MCTS method. Our prediction model T_θ predicts the next states s_{t+1} by taking the current state s_t and the action a_t as inputs. The overall process of the T_θ is as follows: predict x_{t+1}, calculate e_{t+1}, construct i_{t+1}, and compose s_{t+1} from the prediction results.

First, to get predicted states x_{t+1}, a neural-network-based model f_θ predicts the future vehicle states $x_{t+1:t+d}$ of length d from the input sequences of $x_{t-k+1:t}$. In this work, our model f_θ needs to take into account the intentions and behaviors of the human drivers. For this purpose, we use a Multi-Agent Joint Trajectory Prediction Model [15] which is trained to consider the intention of the other vehicles.

Multi-Agent Joint Trajectory Prediction Model [15] predicts a future trajectory of each vehicle. All trajectories of the vehicles are predicted one by one. The overall flow of the network is explained in Figure 2. The network inputs for one prediction are the previous trajectory of one vehicle, and those of the adjacent vehicles. The number of vehicles it accounts can be up to six. Here, the model f_θ consists of the encoder LSTM and the decoder LSTM [17]. The encoder LSTM encodes a previous trajectory of one target vehicle shown in $x_{t-k+1:t}$. Note that the set $x_{t-k+1:t}$ contains the trajectories of all vehicles in the environment. To predict the future trajectory of one vehicle, this encoder LSTM also encodes the trajectories of the six neighborhood of the target vehicle. All the hidden state vectors are entered into conditional variational autoencoder (CVAE) structure [18], and the output of the CVAE becomes the initial hidden state of the decoder LSTM. The final rollout results of the decoder LSTM become the predicted future tajaectory of the target vehicle. By processing all the vehicle trajectories in $x_{t-k+1:t}$, the model f_θ the set of the future trajectories $x_{t+1:t+d}$.

After the future states $x_{t+1:t+d}$ are predicted with f_θ, our model brings x_{t+1} and incorporate it with the future ego state e_{t+1}. Here, e_{t+1} is calculated from a given dynamic model $g(e_t, a_t)$. We use a unicycle model for g as described in Section III. After the calculation of e_{t+1}, our model combines e_{t+1} and x_{t+1}, and construct the image i_{t+1} from them. From e_{t+1} and x_{t+1}, our model can determine the position of the ego and the other vehicles. The image of the ego and the other vehicles are drawn on the image i_{t+1} as rectangular shapes, and the positions and tilts of them are determined by the position and heading information in e_{t+1} and x_{t+1}. The lane marking image is also constructed regarding the predicted position and heading of the ego vehicle.

As a final output, our model composes the next state s_{t+1} by $s_{t+1} = (i_{t-k+2:t+1}, e_{t-k+2:t+1}, x_{t-k+2:t+1})$ where $i_{t-k+2:t}$, $e_{t-k+2:t}$, $x_{t-k+2:t}$ are from the state s_t, and i_{t+1}, e_{t+1}, x_{t+1} are from the prediction results. Our prediction model $T_\theta(s_t)$ returns this predicted next state s_{t+1}. The s_{t+1} is used in the expansion step of our model-based MCTS, which is described in Section V-B. As like s_{t+1}, the state s_{t+l+1} also can be predicted from the previously predicted state s_{t+l}. Once the vehicle states $x_{t+1:t+d}$ are predicted, we can bring x_{t+l+1} from this prediction $x_{t+1:t+d}$. Also, we can calculate e_{t+l} and construct i_{t+l} from x_{t+l+1} and s_{t+l}. Using these s_{t+l}, x_{t+l+1}, e_{t+l+1}, and i_{t+l+1}, our model $T_\theta(s_{t+l})$ can predict the next future state s_{t+l+1}. The overall process of the prediction model is described in Figure 1.

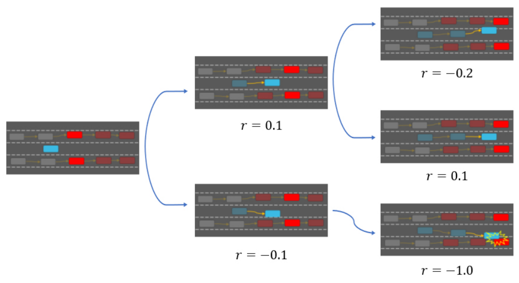

Fig. 3: A simple illustration of the tree which is gotten from our model-based MCTS algorithm. From the predicted states $x_{t+1:t+d}$, our algorithm searches and expands the tree based on the estimated rewards and values.

V. Model-Based MCTS

The MCTS algorithm estimates the Q-value $Q(s,a) = E[\sum_t \gamma^t r_t]$ by simulating each actions with given transition model and reward function. In MCTS methods, each state s is represented as a node, and each action a which can be performed in state s is represented as an edge connected to the node s. A MCTS tree has a directed tree structure with a root node which represents the current state s_0. Each edge in the tree contains statistic information about rewards $r(s,a)$, Q-values $Q(s,a)$, and visit counts $N(s,a)$. These statistic information is incrementally updated while the tree is searched and expanded.

Generally, a search process of MCTS consists of four steps: a selection step, an expansion step, an evaluation step, and a back-propagation step. The MCTS algorithm repeats the above steps for M times to get the final Q-values of the tree. After that, the final action a is chosen according to the final selection method. In our model-based MCTS algorithm, the intention-considering prediction model T_θ is used to estimate the next states of the nodes, and then they are used to estimate the corresponding rewards, and Q-values.

The overall algorithm of the proposed method is described in Algorithm 1, and the illustration of our algorithm including the prediction model and MCTS method is visualized in Figure 3. In the following subsections, the details of our algorithm are explained in the order of each MCTS step.

A. Selection Step

First, in the selection step, the algorithm selects one of the edges according to some selection methods. The selection step occurs repeatedly until it finally reaches to one of the edges which is not visited before. One of the most popular methods for the selection is Upper Confidence Bound Tree (UCT) [5], [19], which selects the action that maximizes the following equation (1).

$$\arg\max_a Q(s,a) + cU(s,a) \tag{1}$$

$$\text{where } U(s,a) = \sqrt{\frac{\log(1+\sum_a N(s,a))}{N(s,a)+1}}$$

$$c \text{ is a constant}$$

Algorithm 1 Model-based MCTS

```
1: Input : $s_0$
2: for $m = 1, ..., M$ do
3:     $t \leftarrow 0$                                   ▷ Selection step
4:     Select $a_0$ with the selection method
5:     while $N(s_t, a_t) \neq 0$ do
6:         $t \leftarrow t + 1$
7:         Select $a_t$ with the selection method
8:     $s_{t+1} \leftarrow T_\theta(s_t, a_t)$            ▷ Expansion step
9:     for each $a$ do
10:        $Q(s_{t+1}, a), N(s_{t+1}, a) \leftarrow 0, 0$
11:    if $s_{t+1}$ is terminal then                      ▷ Evaluation step
12:        $R \leftarrow 0$
13:    else
14:        $R \leftarrow V_\psi(s_{t+1})$
15:    for $\tau = t, ..., 0$ do                          ▷ Back-propagation step
16:        Set $(s, a, r) = (s_\tau, a_\tau, r_\tau)$
17:        $R \leftarrow r + \gamma R$
18:        $Q(s, a) \leftarrow Q(s, a) + \frac{1}{N(s,a)+1}(R - Q(s, a))$
19:        $N(s, a) \leftarrow N(s, a) + 1$
20: Select $a_0$ with the final selection method
21: return $a_0$
```

The first term makes the selection method to choose the action with a higher Q-value, while the second term causes exploration by leading to choose the action which is not fully searched. In this paper, we use the following modified UCT equation [6], which leverages a pre-trained policy π_ϕ.

$$\arg\max_a Q(s, a) + c\pi_\phi(s, a)U(s, a) \tag{2}$$

Here, the policy π_ϕ represents the output of the policy network. The pre-trained policy π_ϕ encourages the algorithm to search the tree to the promising directions. To train the policy π_ϕ, we use Proximal Policy Optimization (PPO) [4] algorithm in a discretized action space. Simultaneously, we also train a value network V_ψ during the PPO policy training, which is used in Section V-C.

B. *Expansion Step*

In the expansion step, a new leaf node s_{t+1} is expanded from the edge a_t which is selected for the selection step. In conventional MCTS methods [5], [11], they use a pre-defined transition model or self-play method to sample the next state s_{t+1}. In our method, we use the prediction model T_θ which considers the intention of the actual human behavior. The training process of T_θ is described in Section IV.

Our pre-trained model $T_\theta(s_t, a_t)$ samples a next state s_{t+1}. After sampling, the algorithm calculates the corresponding reward $r_t = r(s_t, a_t, s_{t+1})$ based on the given reward function r as described in Section III. The predicted state s_{t+1} and corresponding reward $r(s_t, a_t, s_{t+1})$ are recorded in the new leaf node, and used for the evaluation and the back-propagation steps.

C. *Evaluation Step*

In the evaluation step, the value of the expanded node is evaluated with a pre-trained value network V_ψ. With the value of V_ψ and reward r, we can estimate the Q-value $Q(s_t, a_t)$ as $r_t + \gamma V_\psi(s_{t+1})$. In this paper, we train this value network V_ψ using PPO algorithm [4], and fine-tune the network using policy roll-out.

By the expansion step, we can get a predicted state s_{t+1}. From this s_{t+1}, we initialize the value and the visit count of the leaf node using the value network V_ψ.

$$Q(s_t, a_t) = r_t + \gamma V_\psi(s_{t+1})$$
$$N(s_t, a_t) = 1$$

For the network V_ψ, we use a value network trained by PPO [4] as described in V-A. However, we do not directly use this PPO value network because of high errors and variances of the trained network. The network V_ψ is fine-tuned with trajectories $\{s_t, a_t, s_{t+1}, r_t\}_{1:T}$, which is sampled in the environment using the PPO policy π_ϕ. From the sampled trajectories, we can get the exact value of each states,

$$V(s_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{3}$$

The value network $V_\psi(s)$ is trained by the pairs of states and corresponding values to minimize the mean squared error between the result of $V_\psi(s_t)$ and $V(s_t)$.

D. *Back-propagation Step*

Lastly, in back-propagation step, the values of the ancestor nodes of the new leaf node are updated using the values and rewards evaluated in the evaluation step. The back-propagation step occurs from the leaf node to the root node, while updates the node statistics $N(s, a)$ and $Q(s, a)$ of each node. The ancestor of the leaf node is updated based on the evaluated value $V_\psi(s_{t+1})$. Starting from the leaf node s_{t+1}, we sequentially apply the following update rule for each ancestor node $s_t, ..., s_0$.

$$Q(s_\tau, a_\tau) \leftarrow Q(s_\tau, a_\tau) + \frac{1}{N(s_\tau, a_\tau) + 1}(R_\tau - Q(s_\tau, a_\tau))$$
$$\text{where } R_\tau = \sum_{t'=\tau}^{t} \gamma^{t'-\tau} r_{t'} + \gamma^{t-\tau} V_\psi(s_{t+1})$$

E. *Final Selection Method*

After M iterations of the MCTS searches, we can select one of the actions a_0 among the edges of the root node s_0. To select the action a_0, we use the tree statistics updated by MCTS searches. In most MCTS algorithms, the final action is selected with the highest value of the Q-value Q, the visit count N, or a combination of them $Q - U$. In this paper, we adopt the last method for our final selection method. In addition, we use a heuristic method to calculate $Q - U$ without directly using the final Q-values. The equation for

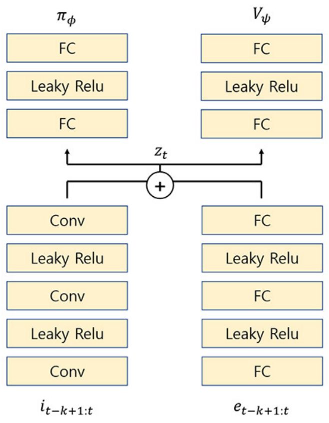

Fig. 4: The network architectures for the policy network π_ϕ and the value network V_ψ. Each $i_{t-k+1:t}$ and $x_{t-k+1:t}$ run through the convoultional layers and the fully-connected layers, and are summed into a hidden vector z_t. A policy distribution and value are calculated from this z_t.

the final selection method we use is described as below:

$$\arg\max_{a_0} Q(s_0, a_0) - U(s_0, a_0) \tag{4}$$

$$\text{where } Q(s,a) = \bar{r} + \frac{1}{\sum_{a'} N(s',a')} \sum_{a'} Q(s',a')N(s',a')$$

$$\bar{r} = r(s',a') - mean_{s,a \in tree} r(s,a),$$

$$U(s,a) = \sqrt{\frac{\log(1+\sum_a N(s,a))}{N(s,a)+1}}$$

VI. Experiments

In this section, we first explain the dataset and the environment settings we used, and then the experimental results.

A. Environment Settings

We trained the prediction network f_θ using NGSIM dataset [3]. NGSIM dataset is collected by taking videos from highway traffic camera, and each vehicle is tracked and tagged with their positions, velocity, and sizes. In this work, we used a viewpoint transformed and refined version of the dataset [16]. Among the NGSIM dataset [3], we tested the cases of both US-101 and I-80 data. After training the network, we used a driving environment which is similar to [16] for the training and testing the algorithm. In this environment, the other vehicles follow the trajectories of the dataset [3] while the ego vehicle is the only vehicle controlled. The data which are used in training the network f_θ and testing our algorithm are completely divided.

B. Networks and Parameters

We used a Multi-Agent Joint Trajectory Prediction Model [15] for the prediction model T_θ. For the prediction, we use the length of state history $k = 20$ and the length of the prediction by the network $d = 20$. The encoder and the decoder LSTM of T_θ has the hidden size of 256.

We also trained the policy π_ϕ and the value V_ψ networks using PPO algorithm [4]. The network architecture we use is similar to the policy network implemented in [16]. The image $i_{t-k+1:t}$ are entered into a three-layer convolutional network which have feature sizes of 64, 128, 256. The ego state vector $u_{t-k+1:t}$ are entered into a two-layer fully-connected network where each layer has 256 hidden size. The outputs of the fully-connected network are projected with a single linear layer whose output dimension has the same as that of the convolutional network. The network outputs for the image and the ego state vector are than summed in a hidden vector z_t. Finally, the policy and the value networks takes this z_t as an input and return the final outputs of π_ϕ and V_ψ. Each policy and value networks consists of two-layer fully-connected network. The network π_ϕ outputs a log probability of categorical distribution with 35 dimension, and the action a can be sampled from it. The entire network architecture is visualized in Figure 4. For the implementation, we used Pytorch library [20] and OpenAi Baselines [21].

For the reward function r_t, we used the following costs:

$$r_t = 1 - C_{prox} - 0.2C_{lane} + C_{success} - C_{failure}$$

The costs C_{prox} and C_{lane} are the same as the costs in [16].

As described in Section V-C, we fine-tuned the value network v_ψ. Because the magnitude of the value network is higher than that of the reward function, we reduced values for the balancing. We used $\gamma = 0.8$ for Equation (3) and Section V-D to reduce the magnitude of the output of V_ψ.

For MCTS algorithm, we used $c = 100.0$ for Equation (2), and $M = 200$ for the repeating time. As a heuristic, we applied breath-first-search to the root node, which means the selection method first selects not-visited action before all the action from the root node are searched at least once. This heuristic is only applied to the root node.

C. Main Experiments

We compared our model-based MCTS algorithm with other reinforcement learning, and imitation learning algorithms. We tested PPO method [4] we trained, and MPUR [16] method which is an imitation learning algorithm and dealing with the same environment we used. Unlike our method, MPUR algorithm returns an action with continous action space. For the comparision, we also trained a PPO algorithm in continous action space with the same network parameters. The results of the experiments are shown in Table I. The method 'no action' means the ego vehicle is controlled to maintain the constant velocity. We showed that our model-based MCTS algorithm exceeds the other methods in each metric. Altough our method leverages the PPO algorithm, it shows a better performance because it can consider the future trajectories during the planning.

To show the effects of the trained model T_θ, we tested MCTS without using T_θ. In ours with constant velocity model, each adjacent vehicles are assumed to maintain the velocity of the last state x_t. As shown in Table I, the MCTS algorithm shows the better results when it uses the trained model T_θ. Because the constant velocity model cannot consider the intentions of the vehicle, the prediction results should not be accurate. In Figure 5, we showed an

TABLE I: The results of the experiments which are conducted in NGSIM US101 and I80 datasets.

Methods	US101		I80	
	mean distance	mean success (%)	mean distance	mean success (%)
No action	1025.6	17.5	619.6	16.2
PPO [4]	2029.1	78.9	1101.0	72.3
PPO [4] (discrete action)	2031.0	81.6	1085.2	68.7
MPUR [16]	1970.9	81.6	1061.0	68.2
Ours (prediction model)	**2106.3**	**87.4**	**1140.2**	**79.4**
Ours (constant velocity model)	2049.7	85.4	1088.7	72.2

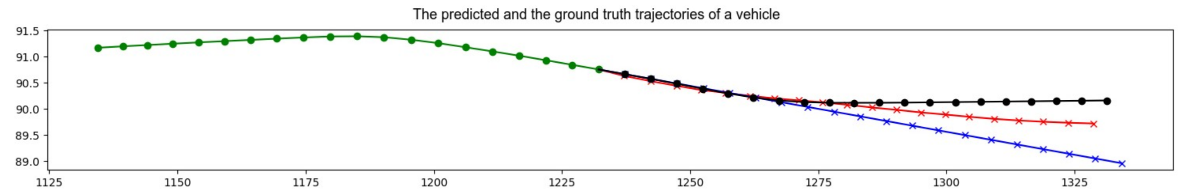

Fig. 5: An example of the predicted trajectory. The horizontal and the vertical axes represent the x and the y coordinate each. We plot the position of a vehicle in the environment with prediction results. The green line means the previous 20 time-step positions of a vehicle. The black line means the ground truth future positions. The red line means the predicted results from our network model f_θ. For the comparison, we also plot the results of the constant velocity model in which the vehicle is assumed to maintain the velocity of the last state.

example of the predicted trajectories of the constant velocity model and T_θ. By the results, we can say that our model T_θ can make more accurate predictions, and it consequently influenced the performance of the planning.

VII. CONCLUSIONS

In this work, we showed that our model-based MCTS method has the-state-of-the-art performances in the highway driving problem. We used a pre-trained prediction model for MCTS expansion steps, which shows smaller prediction errors. By considering the intention of the drivers, the proposed method improved the performance of model-based MCTS.

REFERENCES

[1] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[2] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[3] J. Halkias and J. Colyar, "Ngsim interstate 80 freeway dataset," US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA, 2006.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[6] 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.

[7] J. Wiest, M. Höffken, U. Kreßel, and K. Dietmayer, "Probabilistic trajectory prediction with gaussian mixture models," in *2012 IEEE Intelligent Vehicles Symposium*. IEEE, 2012, pp. 141–146.

[8] R. S. Tomar, S. Verma, and G. S. Tomar, "Prediction of lane change trajectories through neural network," in *2010 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2010, pp. 249–253.

[9] C. Ding, W. Wang, X. Wang, and M. Baumann, "A neural network model for driver's lane-changing trajectory prediction in urban traffic flow," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[10] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 399–404.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[12] T. Yee, V. Lisỳ, M. H. Bowling, and S. Kambhampati, "Monte carlo tree search in continuous action spaces with execution uncertainty." in *IJCAI*, 2016, pp. 690–697.

[13] D. Stern, R. Herbrich, and T. Graepel, "Bayesian pattern ranking for move prediction in the game of go," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 873–880.

[14] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic mdp-behavior planning for cars," 10 2011.

[15] K. Cho, T. Ha, G. Lee, and S. Oh, "Deep predictive autonomous driving using multi-agent joint trajectory prediction and traffic rules," in *2019 IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[16] M. Henaff, A. Canziani, and Y. LeCun, "Model-predictive policy learning with uncertainty regularization for driving in dense traffic," *arXiv preprint arXiv:1901.02705*, 2019.

[17] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[18] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in neural information processing systems*, 2015, pp. 3483–3491.

[19] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[21] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.