2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636442

KB-Tree: Learnable and Continuous Monte-Carlo Tree Search for Autonomous Driving Planning

Lanxin Lei[1*], Ruiming Luo[2*], Renjie Zheng[1*], Jingke Wang[1],
JianWei Zhang[1], Cong Qiu[1], Liulong Ma[1], Liyang Jin[1], Ping Zhang[1], Junbo Chen[1†]

***Abstract*— In this paper, we present a novel learnable and continuous Monte-Carlo Tree Search method, named as KB-Tree, for motion planning in autonomous driving. The proposed method utilizes an asymptotical PUCB based on Kernel Regression (KR-AUCB) as a novel UCB variant, to improve the exploitation and exploration performance. In addition, we further optimize the sampling in continuous space by adapting Bayesian Optimization (BO) in the selection process of MCTS. Moreover, we use a customized Graph Neural Network (GNN) as our feature extractor to improve the learning performance. To the best of our knowledge, we are the first to apply the continuous MCTS method in autonomous driving. To validate our method, we conduct extensive experiments under several weakly and strongly interactive scenarios. The results show that our proposed method performs well in all tasks, and outperforms the learning-based continuous MCTS method and the state-of-the-art Reinforcement Learning (RL) baseline.**

I. INTRODUCTION

As a potential technology to significantly improve traffic efficiency and reduce the human-driving effort, autonomous driving has attracted increasing attention from academia and industry in recent years [1]. In the architecture of the autonomous driving system, motion planning is a crucial component to achieve reliable autonomy [2]. The motion planning module typically takes the perception and localization as input, to make decisions and generate collision-free trajectories in real-time. A good motion planner for autonomous driving is required to generate accurate and smooth trajectories while ensuring real-time requirements. Additionally, the safety guarantees and the interpretability of the planning algorithm are crucial.

Traditional planners are considered interpretable owing to their descriptive nature and achieve good performance in many driving scenarios. There are mainly two categories of traditional planners used in autonomous driving: optimization-based planners and sampling-based planners. Optimization-based methods often suffer from efficiency issues when planning in continuous space. Sampling-based methods discretize continuous space to increase computational efficiency, but the discretization process may cause sub-optimal results [3]. In addition, the traditional planners are designed base on rules, thus they are unable to generalize or handle interactive scenarios well.

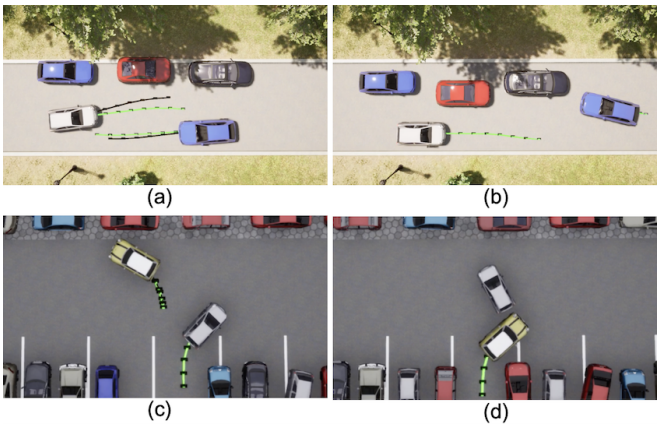

Fig. 1: Motion planning for weakly and strongly interactive scenarios. a): Cooperative Meeting; b): Competitive Meeting; c): Cooperative Parking; d): Competitive Parking;

In recent years, a deluge of learning-based planners has been proposed for autonomous driving. Learning-based planning methods typically train neural network models that directly map the raw perception data (e.g. point cloud, RGB image) or the output of the perception module to the trajectory or control output [4]. However, using neural networks as a black-box to generate motion plans as an end-to-end manner is not interpretable, and thus often safety-prune.

To combine the advantages of learning-based and traditional planners, AlphaGo [5] first utilizes learning-based models in the selection and simulation processes of Monte-Carlo Tree Search (MCTS) to achieve super-human performance in the game of Go, in discrete action spaces. KR-DL-UCT [6] proposes a learnable and continuous MCTS, which combines continuous MCTS [7]–[11] with RL models. It is validated to perform well in simulated curling, yet it follows Progressive Widening [12], [13] to incrementally add a new node (i.e., a new node is expanded only when existing nodes are visited a sufficient number of times), which requires a manually designed threshold.

To address the aforementioned problems, we propose KB-Tree, a novel learnable and continuous MCTS method for motion planning in autonomous driving. The proposed method utilizes Kernel Regression (KR-AUCB), Bayesian Optimization (BO) [14], and Graph Neural Network (GNN) with MCTS to achieve good overall performance. We use BO to replace Progressive Widening in expanding the process to avoid a human-designed threshold, which not only increases scalability but also utilizes more information such as value

* These authors contributed equally to this work.
† Corresponding author, junbo.chenjb@taobao.com
[1] Department of Autonomous Driving Lab, Alibaba DAMO Academy, Hangzhou, China. [2] College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China

distributions of existing nodes to enhance search accuracy. Besides, we propose KR-AUCB to adaptively balance the exploration and exploitation in the selection process. Furthermore, our learning-based models use a customized GNN as a feature extractor, and a mixture density network (MDN) [15] as a policy model.

The main contributions are summarized as follows:

- To our best knowledge, we are the first to apply the learnable and continuous MCTS to motion planning of autonomous driving, to achieve accurate, safe, and interpretable motion plans in real-time.
- We propose KR-AUCB and BO with MCTS to improve selection, expansion, and simulation processes and thus improve search accuracy and efficiency.
- We conduct extensive experimental studies in weakly and strongly interactive scenarios. Experimental results show that our method outperforms the learnable and continuous MCTS baseline KR-DL-UCT as well as RL baseline PPO [16].

II. RELATED WORK

A. Monte-Carlo Tree Search with Reinforcement Learning

Combining MCTS and RL has enjoyed huge success in games. AlphaGo [5], which uses MCTS with two neural networks policy and value evaluation based on human expert data, first achieves superhuman performance in board game Go. AlphaGo Zero [17] utilizing a unified structure for the policy and value networks, obtains a significant improvement in performance. Further, AlphaZero [18] generalizes this approach into a single algorithm to master the games of chess, shogi, and Go. Built upon AlphaZero's powerful search-based policy iteration algorithm, MuZero [19] incorporates a learned model into the training process to internally infer environment dynamics, and reaches the state-of-the-art level in three classic board games as well as 57 Atari games.

B. Continuous Monte-Carlo Tree Search

Existing MCTS methods deal with continuous actions mainly in three ways.

1) Progressive Widening: Progressive Widening (PW) [12], [13] maintains a finite list of available nodes to be searched and incrementally adds children to the list according to the visitation counts. Work has been done to improve action selection in PW. The Kernel Regression UCT algorithm (KR-UCT) [7] introduces information sharing between action nodes of MCTS tree through smooth kernel regression on value estimates. Based on this work, KR-DL-UCT [6] is equipped with a policy-value network to help node expansion, and Lee et al. suggests using MCTS for a global search of coarsely discretized actions while applying a finer local search to more promising actions via value gradients [9].

2) Hierarchical Partitioning: This type of method considers each node of a tree search as the search space for a budgeted-black-box function optimization (BBFO) algorithm, and then hierarchical-partitioning-based optimization [20], [21] is used to find the approximate global optimal. Hierarchical Optimistic Optimization (HOO) [22] builds a cover tree and recursively divides the action space into smaller candidate ranges at each depth. HOOT [23] replaces the UCB1 action selection rule by HOO in UCT to deal with continuous actions in discrete state environments. HOLOP [24] takes an alternative approach of leveraging HOO by representing the entire planning problem as a continuous bandit problem, where actions correspond to plans. Another method uses Voronoi partitioning to dynamically cluster large space and search by sampling from the Voronoi cells [25].

3) Bayesian Optimization: Progressive action selection at each branching step can be modeled using a Bayesian Optimization framework [26]. The Continuous Belief Tree Search algorithm (CBTS) [27] selects actions using a Bayesian optimization process and adds actions up to a fixed limit within a fully observable MCTS solver. BOMCP [28] uses a Gaussian Process to model a belief over the action-value function and selects the action that will maximize the expected improvement in the optimal action value.

III. PRELIMINARIES

A. Reinforcement Learning

We consider the standard formulation of RL, which can be formalized as Markov Decision Processes (MDP) [29]. A MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathtt{Pr}(\mathcal{S})$ is the transition probabilities characterizing the dynamics of the world, $r(s, a) \in \mathbb{R}$ is the immediate reward for taking action a in state s, and $\gamma \in (0, 1]$ is the discount factor. The discounted cumulative return is defined as $G_t := \sum_{i=t}^{T-1} \gamma^{i-t} r(s_t, a_t)$, where t is the time step. The goal of RL is to maximize the expected return $\mathbb{E}_{s_0 \sim S_0}[G_0 \mid s_0]$, where S_0 is the distribution of initial states.

B. Monte-Carlo Tree Search

MCTS [30] is a generic online planning algorithm that combines random sampling and tree search. Starting from the root node, it iteratively performs finite-horizon lookahead search consisting of four steps: selection, expansion, simulation and backpropagation. Upper Confidence Bounds Applied to Trees (UCT) [31] is a common variant of MCTS that adopts UCB1 [32] as the action selection criteria:

$$\arg\min_a \quad \bar{v}_a + c\sqrt{\frac{\log \sum_b n_b}{1 + n_a}} \tag{1}$$

where $\bar{v}_a$ is the expected value of selecting action a, n_b is the number of visits for each expanded child b (including a), and c is a constant to adjust the exploration-exploitation trade-off. UCT is guaranteed to choose the optimal action at the root node $a_0^* = \pi_0^*(s_0)$ as the number of simulations goes to infinity in finite MDPs. To better utilize prior knowledge, PUCT uses PUCB [33] to select action:

$$\arg\min_a \quad \bar{v}_a + cP(a)\frac{\sqrt{\sum_b n_b}}{1 + n_a} \tag{2}$$

where $P(a)$ is the prior probability of selecting action a.

C. Gaussian Mixture Model

A Gaussian Mixture Model (GMM) [34] , also called a mixture of Gaussians (MoG), is a weighted sum of multivariate Gaussian probability density functions:

$$\boldsymbol{p}(\mathrm{x};\boldsymbol{\Theta}) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(\mathrm{x};\mu_i,\boldsymbol{\Sigma}_i) \tag{3}$$

where x is a multidimensional random variable, $\mathcal{N}(\mathrm{x};\mu,\boldsymbol{\Sigma})$ is the multidimensional Guassian function with mean vector μ and covariance matrix $\boldsymbol{\Sigma}$, α is the mixing parameter, $\boldsymbol{\Theta} = \{\{\alpha_1,\mu_1,\boldsymbol{\Sigma}_1\},...,\{\alpha_K,\mu_K,\boldsymbol{\Sigma}_K\}\}$ is the overall set of parameters, and K is the number of Guassians being mixed.

IV. METHODOLOGY

A. Problem Formulation

Since KB-Tree is an RL-based algorithm, it is necessary to define the state, action, and reward before explaining the algorithm. In autonomous driving, the agent is the ego-vehicle controlled by the algorithm. The state for an agent includes the vehicle geometry, moving trajectory, and road scene (e.g., sideline, obstacle). Typically the state is a compact representation of perception data (see next section).

The vehicle receives velocity v and steering ω commands, which are continuous values in the range of [-2, 2] m/s and [-23, 23] degree respectively in our case.

We design a reward function that combines dense reward and sparse reward. The dense reward of each step is defined as the decrement of pose error:

$$r_t = k_1(d_{t-1}^{pos} - d_t^{pos}) + k_2(d_{t-1}^{ang} - d_t^{ang}) + k_3(d_{t-1}^{\kappa} - d_t^{\kappa}) \tag{4}$$

where d^{pos} is the Euclidean distance between current position and target position, d^{ang} is the angle distance of the current orientation to the target orientation, and d^{κ} is the angle difference of the front wheel. k_1, k_2, k_3 are tunable coefficients. For the sparse reward, we design different rewards for different termination states which are added at the end of an episode:

$$r_T = \begin{cases} 10 - 6 \times d^{pos} - \frac{12}{\pi} \times d^{ang} & reach\ goal \\ -15 & collision \\ 0 & timeout \end{cases} \tag{5}$$

B. Feature Representation by Graph Neural Networks

In recent years, convolutional neural networks (CNN) are widely used as deep learning-based navigation models [35], [36]. The surrounding environment and obstacles are rendered on the map with the ego-vehicle as the center, and the features are extracted by convolution. However, the MCTS-based methods repetitively perform simulation and inference, so the map rendering process required by CNN becomes a huge overhead. In addition, redundant information generated in the traditional map rendering process further increases the burden of CNN. Therefore, we use a GNN for feature extraction and inference, which significantly improves the efficiency of the feature extraction process.

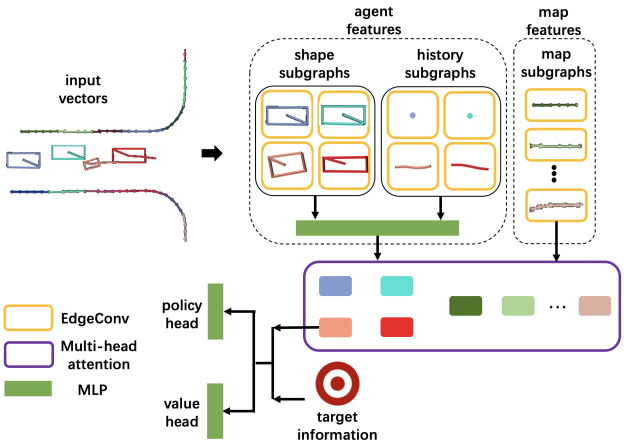

Fig. 2: The architecture of the our policy-value network. We first vectorize the agent information and environment information as shown in the upper left corner of the figure. A hierarchical graph neural network is introduced with edgeconv-based subgraph and multi-head self-attention based global graph.

1) feature representation: We model three kinds of information: vehicle shape, vehicle history trajectory, and road scene. For vehicle shape information, we use vectors that successively connect the coordinates of the four corners and the center of the vehicle, and all the coordinates are unified in the ego-vehicle coordinate system. For vehicle history trajectory information, we encode it with the time sequence vectors, which consist of the position and heading information in the last five timesteps. For road information, firstly, we discretize the road boundary and divide it into different subgraphs with 5m intervals. In each subgraph, we connect the road boundary point to point to construct the road information vector.

2) network architecture: Since the encoded vectors have a large number of clear hierarchy, inspired by VectorNet [37], hierarchical graph neural network is used as our model, which consists of subgraph layer and global graph layer.

As shown in Fig. 2, the nodes v_i in each subgraph first go through a fully-connected graph convolution layer, which can be written as

$$v_i{}^{out} = \varphi_{rel}(g_{enc}(v_i{}^{in}), \varphi_{rel}(\{g_{enc}(v_j{}^{in})\})) \tag{6}$$

After that, we use max-pooling to aggregate the information from different nodes in the same subgraph and output a fixed-length vector to the global graph. A multi-head self-attention module is used in the global graph, which can be written as

$$\left\{p_i{}^{(l+1)}\right\} = GNN(\left\{p_i{}^{(l)}\right\}, A) \tag{7}$$

where $p_i{}^{(l)}$ is the sub-graph output of layer l and A is the adjacency matrix, which is fully-connected. Finally, the outputs of the final layer of the ego-vehicle node are concatenated with the target information and go through an MLP to output the stochastic policy and value estimation.

C. Mixture Density Network Training

In our policy-value network, the policy head is constructed as a mixture density network (MDN) [15] and outputs

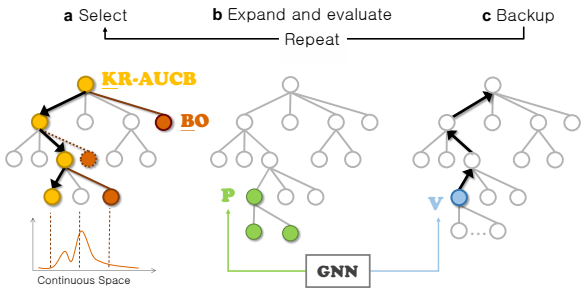

Fig. 3: An overview of KB-Tree. **a**. Each traverse of the tree selects the children with maximum KR-AUCB while expanding the visited nodes by BO in continuous space. Yellow circles indicate nodes selected by KR-AUCB. Orange circles with solid lines indicate nodes are accepted to be expanded and those with dashed lines indicate nodes are rejected. **b**. The leaf node is initially expanded to several children by the policy model (green arrow), and it is evaluated by the value model (blue arrow). All these models are based on a GNN Network. **c**. The value of each node throughout this traverse is updated from the leaf node by backpropagation.

parameters of a GMM. Note that GMM induces a multi-model distribution, intuitively it can better model the action distribution in the context of autonomous driving.

The policy-value network f_θ is trained to maximize the similarity of the action probabilities p to the tree search probabilities, and to minimize the error between the estimated value v and the value drawn from a simulation. Consequently, the loss function consists of two objective terms, accompanied with an $L2$ weight regularization to prevent over-fitting:

$$L_\theta = c_p L_\theta^p + c_v L_\theta^v + c_r \left\|\theta\right\|^2 \tag{8}$$

L_θ^p represents the cross-entropy error:

$$L_\theta^p := -\sum_a \boldsymbol{\pi}_a \log \mathrm{p}_a^\theta \tag{9}$$

where $\boldsymbol{\pi}_a$ is the action probability in MCTS:

$$\boldsymbol{\pi}_a = \frac{n_a}{\sum_b n_b} \tag{10}$$

and p_a^θ is calculated by GMM:

$$\mathrm{p}_a^\theta = \sum_{i=1}^{K} \alpha_i \mathcal{N}(a; \mu_i, \boldsymbol{\Sigma}_i) \tag{11}$$

where α_i, μ_i and $\boldsymbol{\Sigma}_i$ are obtained from MDN. L_θ^v represents the mean-squared error:

$$L_\theta^v := \sum_i (\mathrm{v}_i^\theta - G_i)^2 \tag{12}$$

where v^θ is the estimated value of current state, G is the cumulative return calculated with rewards provided by the environment. The parameters c_p, c_v, c_r are weights of each loss component.

D. Monte-Carlo Tree Search in Continuous Action Spaces

To efficiently search the optimal action in continuous space, we propose an asymptotical PUCB based on Kernel Regression (KR-AUCB) and adapt it to KR-DL-UCT [6]

Algorithm 1: KB-Tree

```
1  v̄ ← 0
2  p^θ ← policy network
3  v^θ ← value network
4  s ← the current state
5  A ← A set of visited actions of s
6  D ← 0 // Search depth
7  T ← false // Terminal status
8  while D <maxDepth and T ≠ true do
9      if no children nodes then
           /* Expansion process                    */
10         A ← ∪_{i=1}^{k} {a_init^i}   (a_init^i ~ p^θ(·|s))
11         v̄ ← v^θ(s)
12     else
           /* Selection process                    */
13         a* ← BayesianOptimization(A)
14         A ← A ∪ a*
15         a ← KR-AUCB
16         if a ≠ a* then
17             A ← A \ a*
18         end
19         s', r', T', A' ← TakeAction(s, a)
20         s, T, A ← s', T', A'
21         v̄ ← v̄ + γ^D r'
22         D ← D + 1
23     end
24 end
   /* Simulation process                          */
25 if T ≠ true then
26     v̄ ← v̄ + γ^D v^θ(s)
27 end
   /* Backpropagation process                     */
28 Update v̄
```

which optimizes the selection and expansion steps with Kernel Regression and Bayesian Optimization. We name our new method KB-Tree as shown in Fig. 3. The pseudocode of the KB-Tree is presented in **Algorithm 1** which is described in detail below.

The KB-Tree algorithm is iteratively applied on the root of the search tree within a pre-defined number of iterations max_iter. Each iteration follows the lookahead search procedure of the vanilla MCTS until the depth of the tree reaches a threshold $maxDepth$ or a terminal state is met (line 8). The keys of KB-Tree are KR-AUCB that determines the node selection (line 15) and the Bayesian Optimization involved in producing candidates for expansion (line 13). These two techniques are respectively described in detail in the following two subsections.

1) Asymptotical PUCB based on Kernel Regression: We propose a Kernel Regression-based asymptotical PUCB (KR-AUCB) in the selection stage of MCTS, which initially prefers unexplored actions, then asymptotically prefers actions with high prior probability or low visit count, and finally prefers actions with high value.

Both initially expanded actions and progressively added actions form the dataset for Kernel Regression at each node. The expected value and the count of visitation for each node in the origin PUCB formula (Eq. (2)) are estimated using the information of already branched sibling nodes by Kernel

Regression and kernel density respectively:

$$\mathbb{E}(\bar{v}_{\hat{a}} \mid \hat{a}) = \frac{\sum_a K(\hat{a}, a)\bar{v}_a n_a}{\sum_a K(\hat{a}, a) n_a} \tag{13}$$

$$W(\hat{a}) = \sum_a K(\hat{a}, a) n_a \tag{14}$$

where $\hat{a}$ is the selected action, $a \in \mathcal{A}_t$ is existing sibling action, and K is the Gaussian probability density:

$$K(\hat{a}, a) = \frac{\exp\left(-\frac{1}{2}(\hat{a}-a)^T \mathbf{\Sigma}^{-1}(\hat{a}-a)\right)}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \tag{15}$$

$\mathbf{\Sigma}$ is the covariance matrix and n is the dimension of action.

To avoid premature convergence to a local optimum, we propose a better prior policy P_{asym} to asymptotically control exploration decay:

$$P_{asym} = \lambda P_{prior} + (1-\lambda) P_{uniform} \tag{16}$$

where $P_{uniform} = \frac{1}{|\mathcal{A}_t|}$ is the uniform distribution from action space $\mathcal{A}_t$, $P_{prior} = \mathrm{p}^{\boldsymbol{\theta}}$ is the prior action probability distribution from policy network, $\lambda \in [0, 1]$ is a tunable parameter that increases from 0 to λ_{target} over training time.

The formula of the KR-AUCB is:

$$\underset{\hat{a} \in \mathcal{A}_t}{\arg\max} \quad \mathbb{E}[\bar{v}_{\hat{a}} | \hat{a}] + c P_{asym}(\hat{a}) \frac{\sqrt{\sum_a W(a)}}{W(\hat{a})} \tag{17}$$

where c is the exploration parameter.

2) Bayesian Optimization: When the current node has no children, k actions are sampled from policy $\mathrm{p}^{\boldsymbol{\theta}}$ as initial expansion (line 9-11). Oherwise, we use Bayesian Optimization (BO) to choose candidate for node expansion, and the prior distribution of the expected value is modeled by Gaussian Process (GP). During action expansion at belief node b, the action a^* sampled as candidate is decided by previously branched actions $Ch(b)$:

$$a^* \leftarrow \underset{a \in \mathcal{A} \setminus Ch(b)}{\arg\max} \; A(a) \tag{18}$$

$A(a)$ is the acquisition function in BO that guides the sampling to areas where the probability of finding the optimum increases. Here the acquisition function is defined as:

$$A(\hat{a}) = \mu + \omega_1 \sigma + \omega_2 d(\hat{a}) \tag{19}$$

where μ and σ are the GP posterior mean and standard deviation of the candidate $\hat{a}$, and $d(\hat{a})$ is the expected distance between $\hat{a}$ and other branched actions $a \in Ch(b)$:

$$d(\hat{a}) = \frac{1}{|Ch(b)|} \sqrt{\sum_i (\hat{a} - a_i)^2} \tag{20}$$

The first two terms of Eq. (19) can be view as the upper boundary of the expected value of choosing $\hat{a}$, and the last term penalizes the candidate for being too close to existing actions. ω_1 and ω_2 are tunable coefficients for controlling the exploration-exploitation trade-off. Fig. 4 visualizes the concept of choosing an action with BO.

Once the candidate is sampled, it is added into the visited action set $\mathcal{A}$ (line 14). Then the action selected to be taken is determined by KR-AUCB (line 15). If the result is the not the candidate a^*, we remove a^* from $\mathcal{A}$ (line 16-18). Next, the selected action a is executed and yields a new state at the next depth (line 19-22). After an iteration stops, the expected value of the leaf node $\bar{v}$ is evaluated by value network $\mathrm{v}^{\boldsymbol{\theta}}$. In the end, the value of each node throughout the traverse is updated in a backpropagation way (line 28).

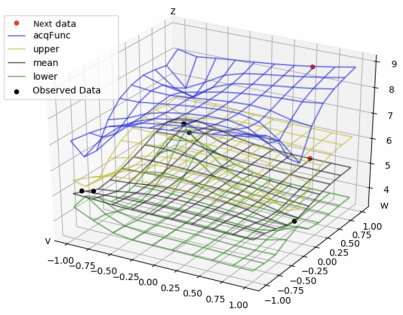

Fig. 4: Illustration of the Bayesian Optimization based on GP. Black points are observed data which share the same parent. The yellow layer, black layer and green layer respectively represent the upper confidence, the mean value and the lower confidence of the observed data induced by GP. The red point is the maximum of the acquisition function (blue layer) that is sampled as the new children to be branched.

V. EXPERIMENTS

A. Autonomous Driving Simulator

We evaluate the proposed method in a customized autonomous driving simulator. It is light-weight to meet the spatial-temporal restriction in MCTS type methods. Furthermore, it can reconstruct road scenes by importing HD-map scanned in the real environment so as to cover diverse scenarios that appear in the physical world.

B. Experimental Setup

The discount factor γ is set to 0.9. The buffer size and batch size of the policy-value network are 5500 and 64. The maximal depth of the search tree is set to 10. We set exploration parameter c to 0.3 and exploration decay parameter λ_{target} to 0.7.

First of all, we compare our proposed KB-Tree method with its discrete version and a baseline KR-DL-UCT in a parking task. Then we compare KB-Tree with a commonly-used RL baseline algorithm, Proximal Policy Optimization (PPO) [16] in a navigation task into a complex real-world scene. At last, we test KB-Tree in a dynamic environment with two tasks: parking and meeting. Videos of all our experiments are provided online [1].

To improve training efficiency, all programs are trained with 20 processors in parallel.

C. Simulation Evaluation

1) Comparison with Discrete Version: To validate the advantage of the proposed improvements aiming to scale the planning in the continuous space of our new method, we compare KB-Tree to its discrete version in a parking task

[1] https://sites.google.com/view/kb-tree/

(the task scene is shown in Fig. 5). The discrete version here means all components designed for continuity in MCTS are removed and the action space is discrete (the steering angle is discretized into 7 quantities and the speed is discretized into 3 quantities), which is similar to AlphaGo Zero.

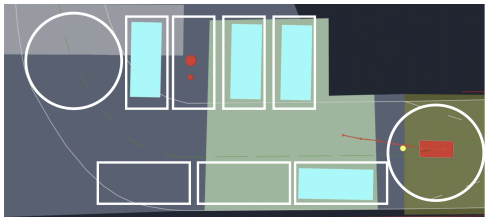

Fig. 5: A case of parking task. The red block represents the ego-vehicle. The big red dot represents the ending point. The big circle represents the starting area. White rectangles indicate the locations of the static vehicles (blue blocks). The yellow dot and little red dot represent the current and the target orientation of the ego-vehicle respectively. The starting point is randomly sampled within the white circle and the ending point is randomly sampled in the unoccupied white rectangles. Static vehicles are randomly located in white rectangles.

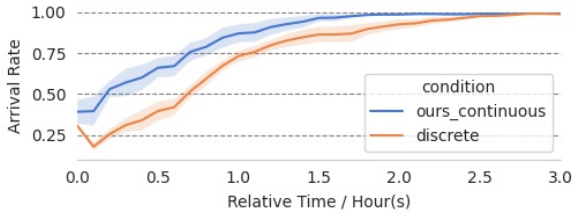

Fig. 6: Training curve of the continuous and discrete version of KB-Tree in parking task. The plot shows the mean and standard deviation over 3 runs with different random seeds.

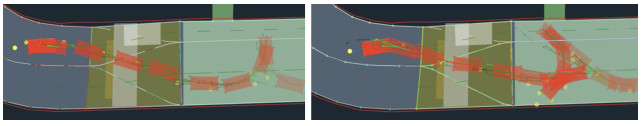

Fig. 7: Trajectories of KB-Tree performing in reaching task. The vehicle (red block) is required to move from the initial point (on the right) to the target point (big red dot on the left). Yellow dots in front of the vehicle and little red dots beside target point represent the current and the target orientation of the vehicle respectively. **Left**: Continuous KB-Tree. **Right**: Discrete KB-Tree.

The learning curve depicted in Fig. 6 shows that both versions of KB-Tree can converge to a 100% arrival rate, but the continuous version converges faster than the discrete version. Further, we test both versions of KB-Tree in a reaching task. As depicted in Fig. 7, when the initial orientation of the vehicle is not in accordance with the target orientation, discrete KB-Tree spends more effort in adjusting the direction than continuous KB-Tree. These two tests prove that the improvements proposed for adapting learning-based MCTS to continuous action space in KB-Tree are effective, and planning in continuous space results in better performance than in discrete space.

2) Comparison with Continuous MCTS Baseline: To evaluate the advantage of our proposed method over the existing continuous MCTS method, we implement KR-DL-UCT [6] as the baseline, and compare the performance of programs trained by KB-Tree and KR-DL-UCT in the same parking task. The learning curve depicted in Fig. 8 shows that our method outperforms KR-DL-UCT, which indicates that the effectiveness of the proposed asymptotical PUCB method and Bayesian Optimization.

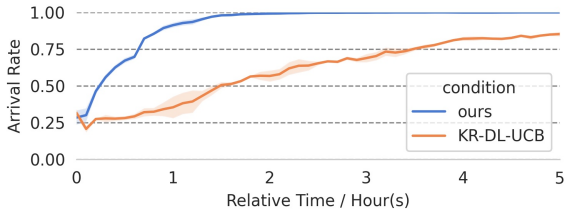

Fig. 8: Training curve of the KB-Tree and KR-DL-UCT in parking task. The plot shoxws the mean and standard deviation over 3 runs with different random seeds.

3) Comparison with RL Baseline: Real-world scenes involve various complicated circumstances and are much more challenging than hand-crafted scenes in a simulator for motion planning. To evaluate the performance of our algorithm in different scenarios, we apply KB-Tree in the real-world environment of Guangfu Temple (a scenic spot in Hangzhou). As shown in Fig. 9(a), the sidelines and obstacles are all in accordance with the imported HD-map environment, which contains relatively difficult situations (e.g., sharp turn, random obstacles, etc.). The vehicle navigates in this environment with goal points are consecutively provided. Quantitative metrics are calculated to evaluate the performance. We totally collect 55 cases containing six scenarios (lane following, U-Turn, turnaround, roadblock, side pass, and narrow passing), then we test each case 50 times and evaluate quantitative metrics (arrival rate, sideline collision rate, obstacle collision rate, and timeout rate).

From Table I we can see that the program trained by KB-Tree performs significantly better than PPO. Our method achieves a high arrival rate except for two relatively hard scenarios (Turnaround and Roadblock) which may cause a collision or creep. In most cases, the vehicle planned by KB-Tree can reach the goal normally, even in non-trivial scenarios like U-Turn and roadblock in which PPO severely suffers in the collision. The results prove that the involvement of MCTS can produce safer actions than pure RL planners, and KB-Tree can handle diverse and even difficult scenarios.

4) Performance in Interactive Scenario: Real driving scenario includes multiple vehicles. To convincingly validate the practical potential of KB-Tree, we test it in an interactive scenario where two controllable vehicles are involved. We design two tasks: parking and meeting. The scenes of these tasks are demonstrated in Fig. 10 and Fig. 11 respectively.

Table II shows that KB-Tree performs well in both tasks with a high arrival rate and very few collisions. Fig. 11 demonstrates four cases in the meeting task. In a weakly interactive scenario (meeting in the spacious road), two vehicles pass by normally (see two subfigures on the bottom of Fig. 11). In a strongly interactive scenario (narrow meeting), one vehicle moves backward to create space for the other

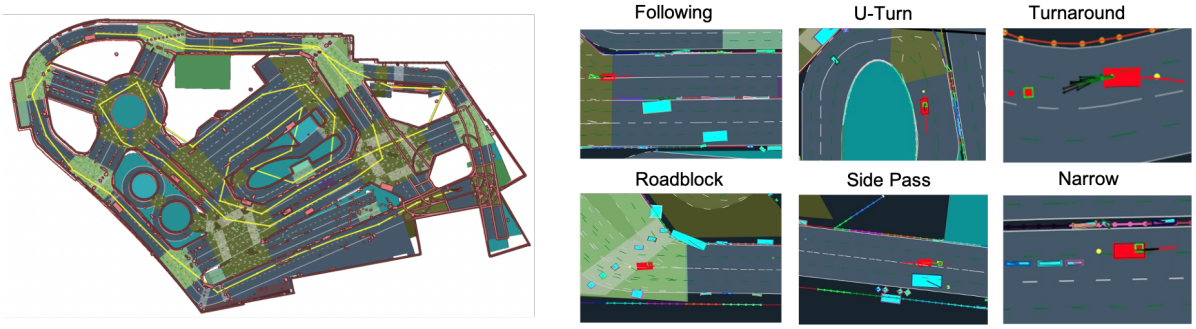

Fig. 9: (a) The HD-map of the Guangfu Temple. The yellow line indicates the navigation route of the vehicle. Red lines represent sidelines and pink polygons represent obstacles. (b) Six scenarios appeared in the Guangfu Temple scene.

TABLE I: Quantitative Metrics of KB-Tree in Navigation Task

Scenaio	Count	KB-Tree				PPO			
		Arrival (%)	Sideline Collision (%)	Obstacle Collision (%)	Timeout (%)	Arrival (%)	Sideline Collision (%)	Obstacle Collision (%)	Timeout (%)
Following	32	99.81	0.06	0.13	0.00	69.29	9.82	19.80	1.09
U-Turn	2	100.00	0.00	0.00	0.00	0.00	97.50	2.50	0.00
Turnaround	1	84.00	0.00	16.00	0.00	51.00	33.00	11.00	5.00
Roadblock	8	87.00	0.00	2.75	10.25	14.00	0.00	84.75	0.00
Side Pass	9	98.44	0.00	1.56	0.00	68.71	6.69	22.83	1.77
Narrow	3	100.00	0.00	0.00	0.00	53.27	13.53	33.20	0.00
All	55	97.45	0.04	0.73	1.78	57.42	11.69	29.69	1.20

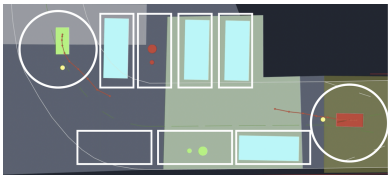

Fig. 10: A case of parking task in the interactive scenario. Red block and green block represent two vehicles which are required to move from starting points (randomly sampled in white circles) to target points (big dots in white rectangles). Blue blocks represent static vehicles randomly located in seven white rectangles. Yellow dots ahead of vehicle and little dots beside target points indicate the current and target orientation of the vehicle respectively.

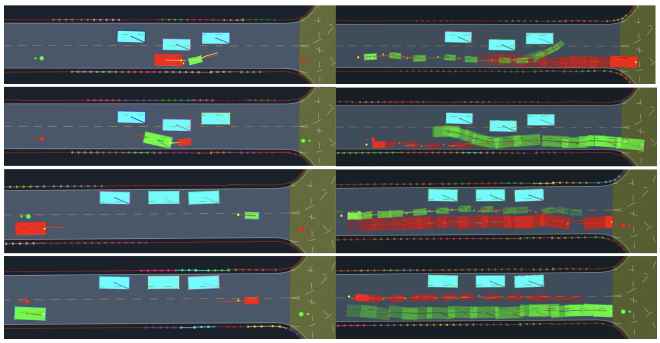

Fig. 11: Four cases of meeting task in the interactive scenario. Red block and green block represent two vehicles aiming to reach target points (big dots). Blue blocks indicate static vehicles. Yellow dots ahead of vehicle and little dots beside target points indicate the current and target orientation of the vehicle respectively. **Left**: Scenes of each case. **Right**: Trajectories of two vehicles.

TABLE II: Results of Tasks in Interactive Scenario

Task	Parking	Meeting
Arrival (%)	93.4	97.9
Static Obstacle Collision (%)	4.0	0.0
Dynamic Obstacle Collision (%)	1.5	0.1
Sideline Collision (%)	0.1	0.0
Timeout (%)	1.0	2.0

TABLE III: Comparison of GNN and ResNet18

	ResNet18	GNN
Feature construction (ms)	3.525	0.754
Inference (ms)	5.111	3.624
Number of parameters	11.2M	69.9K

vehicle passing (see two subfigures on top of Fig. 11). The results prove that vehicles with KB-Tree planners can perform collaborative behaviors in interactive scenarios.

D. GNN vs ResNet

We compare the resource expenditure of our GNN model and the widely-used ResNet18 model [38] as feature extractor in terms of consumed time in feature construction and inference and the number of parameters. Results in Table III show that GNN is dramatically faster than ResNet18 in feature construction and inference, and has much fewer parameters. This comparison explains why we use GNN.

VI. CONCLUSION

In this paper, a novel learning-based MCTS in continuous space, KB-Tree, is presented to deal with real-time planning in autonomous driving. KB-Tree improves the selection, simulation, and expansion processes with Bayesian Optimization, Kernel Regression, asymptotical PUCB, and RL models. As the first learnable and continuous MCTS method applied in autonomous driving planning, KB-Tree is validated in various weakly and strongly interactive scenarios and shows great potential in being used in practice. Experimental results show that our algorithm outperforms not only the learning-based continuous MCTS baseline but also the state-of-the-art RL baseline.

ACKNOWLEDGMENT

This work was supported by the funding of "Leading Innovation Team of the Zhejiang Province" (2018R01017).

REFERENCES

[1] Y. Huang and Y. Chen, "Autonomous driving with deep learning: A survey of state-of-art technologies," *arXiv preprint arXiv:2006.06091*, 2020.

[2] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.

[3] P. Cai, Y. Luo, A. Saxena, D. Hsu, and W. S. Lee, "Lets-drive: Driving in a crowd by learning from tree search," *arXiv preprint arXiv:1905.12197*, 2019.

[4] L. Tai, J. Zhang, M. Liu, J. Boedecker, and W. Burgard, "A survey of deep network solutions for learning control in robotics: From reinforcement to imitation," *arXiv preprint arXiv:1612.07139*, 2016.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of go with deep neural networks and tree search," *Nature*.

[6] K. Lee, S.-A. Kim, J. Choi, and S.-W. Lee, "Deep reinforcement learning in continuous action spaces: a case study in the game of simulated curling," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2937–2946.

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

[8] T. Anthony, R. Nishihara, P. Moritz, T. Salimans, and J. Schulman, "Policy gradient search: Online planning and expert iteration without search trees," 2019.

[9] J. Lee, W. Jeon, G. H. Kim, and K. E. Kim, "Monte-carlo tree search in continuous action spaces with value gradients," *Proceedings of the Aaai Conference on Artificial Intelligence*, vol. 34, no. 4, pp. 4561–4568, 2020.

[10] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Perez, "Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 6, pp. 9916–9924, 2020.

[11] W. Mao, K. Zhang, Q. Xie, and T. Başar, "Poly-hoot: Monte-carlo planning in continuous space mdps with non-asymptotic analysis," *arXiv preprint arXiv:2006.04672*, 2020.

[12] R. Coulom, "Computing elo ratings of move patterns in the game of go," *ICGA journal*, vol. 30, no. 4, 2007.

[13] J. B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for monte-carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. p.343–357, 2008.

[14] M. Pelikan, D. E. Goldberg, E. Cantú-Paz *et al.*, "Boa: The bayesian optimization algorithm," in *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1. Citeseer, 1999, pp. 525–532.

[15] C. M. Bishop, *Mixture density networks*. Aston University, 1994.

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

[17] David, Silver, Julian, Schrittwieser, Karen, Simonyan, Ioannis, Antonoglou, Aja, and H. and, "Mastering the game of go without human knowledge," *Nature*, 2017.

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, and T. a. Graepel, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362.

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

[20] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári, "X-armed bandits." *Journal of Machine Learning Research*, vol. 12, no. 5, 2011.

[21] R. Munos, "Optimistic optimization of deterministic functions without the knowledge of its smoothness," in *Advances in neural information processing systems*, 2011.

[22] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári, "Online optimization in x-armed bandits," in *Twenty-Second Annual Conference on Neural Information Processing Systems*, 2008.

[23] C. Mansley, A. Weinstein, and M. Littman, "Sample-based planning for continuous action markov decision processes," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21, no. 1, 2011.

[24] A. Weinstein and M. Littman, "Bandit-based planning and learning in continuous-action markov decision processes," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 22, no. 1, 2012.

[25] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Pérez, "Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 06, 2020, pp. 9916–9924.

[26] J. Mockus, "Bayesian heuristic approach to global optimization and examples," *Journal of Global Optimization*, vol. 22, no. 1-4, pp. 191–203, 2002.

[27] P. Morere, R. Marchant, and F. Ramos, "Bayesian optimisation for solving continuous state-action-observation pomdps," *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[28] J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer, "Bayesian optimized monte carlo planning," 2020.

[29] Laurence, A., and Baxter, "Markov decision processes: Discrete stochastic dynamic programming," *Technometrics*, 1995.

[30] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, no. 1, pp. 1–43, 2012.

[31] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, 2006.

[32] P. Auer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, 2002.

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

[34] D. A. Reynolds, "Gaussian mixture models." *Encyclopedia of biometrics*, vol. 741, pp. 659–663, 2009.

[35] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[36] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.

[37] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 525–11 533.

[38] 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.