2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Speeding Up Path Planning via Reinforcement Learning in MCTS for Automated Parking

Xinlong Zheng*, Xiaozhou Zhang*, Donghao Xu
DeepRoute.Ai
3125 Skyway Ct, Fremont, CA 94539, USA
{xinlongzheng, xzzhang}@alumni.upenn.edu, donghaoxu@deeproute.ai

***Abstract*— In this paper, we address a method that integrates reinforcement learning into the Monte Carlo tree search to boost online path planning under fully observable environments for automated parking tasks. Sampling-based planning methods under high-dimensional space can be computationally expensive and time-consuming. State evaluation methods are useful by leveraging the prior knowledge into the search steps, making the process faster in a real-time system. Given the fact that automated parking tasks are often executed under complex environments, a solid but lightweight heuristic guidance is challenging to compose in a traditional analytical way. To overcome this limitation, we propose a reinforcement learning pipeline with a Monte Carlo tree search under the path planning framework. By iteratively learning the value of a state and the best action among samples from its previous cycle's outcomes, we are able to model a value estimator and a policy generator for given states. By doing that, we build up a balancing mechanism between exploration and exploitation, speeding up the path planning process while maintaining its quality without using human expert driver data.**

I. INTRODUCTION

In general, an automated parking task is primarily separated into sub-modules such as perception, localization, planning, control, etc. The appearance of novel perception frameworks such as Bird's-Eye-View[1] and occupancy networks[2] brings clarity into sensing the environments in an online automated parking task. There are also well-studied methods of localization to achieve a high-precision result. The performance of an automated parking execution heavily relies on the outcomes from the planning and control module.

Given a fully observable environment, the goal of the planning module is to generate an optimal trajectory for autonomous vehicles. The trajectory consists of several points in the spatio-temporal space, combining both path and speed information. Common methodologies applied on a real-time planner partition the spatio-temporal domain into path planning and speed planning in order to reduce the complexity and computational load. The speed planning under parking scenarios can be relatively easy compared to on-road driving scenarios since the planning time horizon is limited, the search space is smaller, and there are fewer interactive objects involved.

*Equal Contribution

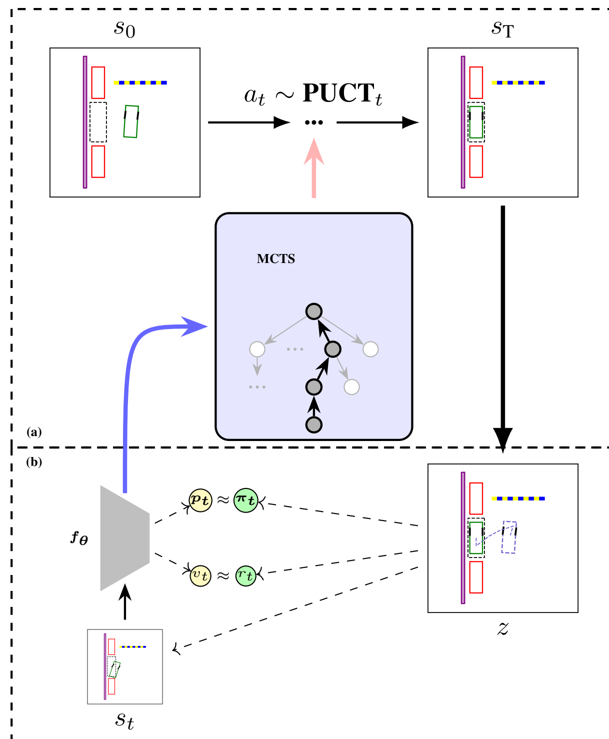


Fig. 1. Reinforcement learning integrated MCTS in path planning tasks. (a) The agent plans its move under the guidance of MCTS. s_0 is the start state and s_T is the destination state. a_t is the action takes at time t selected by PUCT[3]. (b) Neural network training against the previously produced results. z is the terminated tree that can be used to generate training input state s_t, training label π_t, and r_t. f_θ is the neural network projecting s_t to policy distribution p_t and value v_t.

Sampling-based approaches are the most common method employed on the path planning task in parking scenarios, and it can be followed by a numeric optimization after. The performance of such an algorithm relies heavily on its sampling density — the denser the sampling, the algorithm is more likely to converge towards a global optimal solution. Under the parking tasks, especially those complex ones, the vehicle would like to use most of the spare space with bi-directional maneuvers to reach the destination spot with minimal effort while ensuring a comfortable experience for clients. In order to get a satisfying result, a higher dimension of sampling space and a compact sampling strategy are needed, which makes the planning process time-consuming.

Many methods have been introduced to reduce the time consumption for such algorithms in a real-time system. Most of them aim to enhance the exploitation capability of the algorithm, by either trimming out infeasible samples or prioritizing the promising ones with a heuristic estimate. However, too much preference on the exploitation side would decrease the ability to explore and lead to sub-optimal solutions.

Monte Carlo tree search (MCTS)[4], as an orderly sampling-based method, shows its strength in solving long-horizon planning tasks like path planning. However, without the significant guidance of prior domain knowledge, the iterative tryouts from its cycled operations make the planning time too long to run in the real-time system. In our approach, we would like to leverage the capability of reinforcement learning to provide domain knowledge guidance for MCTS, boosting up its planning time and finding the equilibrium of searching process, without the help from human driver data, as what is shown in Fig. 1.

II. RELATED WORKS

The path planning problem in automated parking tasks has been studied for decades. It can be easily treated as a Euclidean geometry problem with constraints. By putting pieces of straight lines and basic curves together, one can construct a path by connecting a series of pre-designed points. The curves are mainly circle arcs that can be computed to meet the basic kinematic constraints like maximum curvature. [5] [6] have implemented these geometric methods in different types of parking scenarios. The computing time of the geometric method is very short, but it only works well for easy parking tasks, since there is not enough variation in the shape of the path.

The sampling-based approach is well-studied as well. In [7], implementation of one of the most popular random-based sampling algorithms is given: the Rapidly-exploring Random Tree (RRT)[8] for planning on dynamic vehicle model, and it has been used by [9] in automated parking situation. RRT has a simple logic and is easy to implement, but its randomness leads to a lack of stability and is not guaranteed to find an optimal solution.

Besides the random one, another popular orderly-based sampling algorithm is A*[10] and its variant. [11] comes up with a Hybrid-State A* Search which associates a continuous state with each sample, and uses it to solve a perpendicular parking task. The process of A* is expedited by the prior knowledge known as heuristic function[10]. In terms of automated parking, the study on its prior knowledge is mainly focused on the shape properties of the curves, such as the Reeds-Shepp (RS) curve[12], Bezier curve[13], etc. However, the curve's shape property becomes less useful when there are lots of obstacles in the parking environment.

Another way to formulate the corresponding prior knowledge considering both the curve's shape and obstacles is through supervised learning from human expert driver data like [14]. To achieve a well-rounded performance, a large amount of high-quality human data is required and these are often expensive. Prior knowledge can also be enhanced using reinforcement learning strategies. Since the advent of AlphaGo [15], researchers have been working to solve long-horizon planning tasks in the field of autonomous driving, such as those discussed in [16]. [17] managed to address autonomous parking tasks with MCTS, but their solution was limited to parallel parking scenarios.

III. PRELIMINARIES

A. Problem Formulation

Assume that the automated parking task is executed at a low constant speed in an observable and consistent environment. The path planning in such task can be considered as a Markov Decision Process (MDP) with certain $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, and $\mathcal{R}$. The goal of this process for autonomous vehicles is to reach the destination pose from a starting pose while minimizing the corresponding cost of doing that. The MDP can be formulated as follows.

Let $\mathcal{S}$ be the state space of the agent (autonomous vehicle). A state s_t can be defined as the pose of the vehicle as (x_t, y_t, ϕ_t) at time t, where x and y are the coordinates of the rear axis center in a 2D Cartesian space and ϕ is the heading of the vehicle.

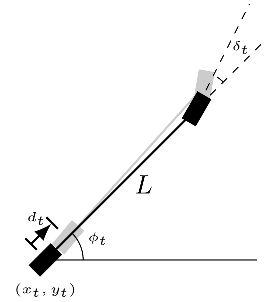


Fig. 2. Bicycle vehicle model

Let $\mathcal{A}$ be the action space of the agent and $\mathcal{T}$ as the transition function maps a state s_t to s_{t+1} with a given action a_t. We use the bicycle model as in Fig. 2 for vehicle kinematics under this low-speed parking scenario. By considering the speed as a constant as well, we can use the distance gap d_t to denote the gap between different vehicle poses. The motion equations can be written as follows:

$$\begin{aligned} x_{t+1} &= x_t + d_t \cos \phi_t \\ y_{t+1} &= y_t + d_t \sin \phi_t \\ \phi_{t+1} &= \phi_t + \frac{d_t \tan \delta_t}{L} \end{aligned} \tag{1}$$

Equation (1) reveals the composition of action space as an action $a_t = (d_t, \delta_t)$, where δ_t is the front wheel angle of the vehicle. (1) itself is the transition function, where L is the length of the wheel base, with the kinematic and collision constraint taken into account. In practice, the action space is discretised, which made the state space discrete as well.

Let $\mathcal{O}$ be the observation space. o_t is the observation at time t, which is sensed by the perception module formulated as the occupancy grids of different layers for interested environment objects of different categories such as parking slots, obstacles, road edges, speed bumps, etc. Since the environment is assumed as consistent during the whole process, the o_t is a constant at any time t as well.

Let $\mathcal{R}$ be the reward function which maps a corresponding cost by the sub-sequential states the process goes so far and a fixed scalar reward if the agent reaches the destination as

$$\begin{aligned} r &= \mathcal{R}(s_0, s_1, ..., s_t, o) + r_t \\ &= \mathcal{C}_{\text{safety}}(s_0, s1, ..., s_t, o) \\ &\quad + \mathcal{C}_{\text{comfort}}(s_0, s1, ..., s_t) \\ &\quad + \mathcal{C}_{\text{efficiency}}(s_0, s1, ..., s_t) + r_t \end{aligned} \tag{2}$$

If the agent at s_t reaches the destination pose, $r_t = 1$, otherwise $r_t = 0$. The cost function itself is related to safety, comfort, and efficiency of the path formed by traveling through the states. Paths getting closer to the obstacles within a certain threshold would result in more negative safety cost. Paths with redundant wheel or gear moves would result in more negative comfort and efficiency cost .

B. Monte Carlo Tree Search

Since the discovery of Upper Confidence bounds applied to Trees (UCT)[4] was introduced in the year 2006, the Monte Carlo tree search has become popular when it comes to finding an optimal solution by searching in high-dimension spaces. Especially after MCTS was used for AI to play long-horizon board games like chess and go[18][15], it was notably suitable for solving episodic planning tasks like an MDP problem.

The basic idea of MCTS just meets the requirements of the path planning task: it naturally balances between exploitation and exploration by building a search tree and iteratively updating its search strategy. The algorithm itself falls into a cycle consisting of the following procedures: *selection*, *expansion*, *simulation* and *backpropagation*.

During the *selection* phase, MCTS repeatedly chooses a child node to visit starting from the root node, until it finds a node that is not *expanded*. The *selection* is guided by certain criteria such as the *upper confidence bound* (UCB)[19] and the *predictor upper confidence bound* (PUCB)[3]. This criterion primarily aims to strike a balance between exploiting the child with known high rewards, termed exploitation, and exploring lesser-visited children, known as exploration.

If the selected node does not reach the terminal state, it enters the *epansion* phase and a *simulation* will be applied to the node. A *simulation* can be carried out with different methods such as random rollout, rule-based rollout, or other estimations to get a reward for the node.

The estimated reward will be *propagated back* from the *simulated* child all the way back to the root node. Interested properties of the nodes along the retrieved path will be updated respectively as well.

The whole cycle will be repeated until a terminal condition is met. Then the updated preference on the nodes from the search tree will be adapted for real-time execution from the root node with the same *selection-expansion-simulation-backpropagation* procedures.

IV. METHODOLOGY

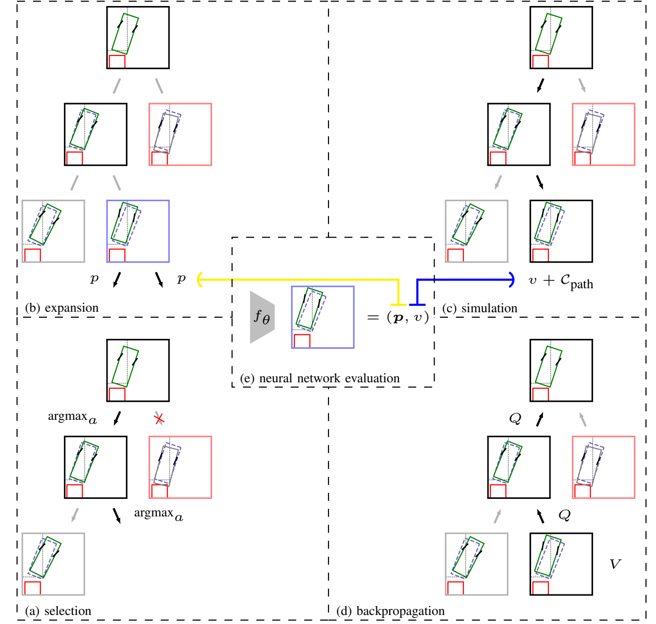


Fig. 3. Cycled steps of MCTS in path planning. (a) Iteratively selection tree traversal. (b) Node expansion with policy generator. (c) Simulation with value approximator and cost function $\mathcal{C}_{\text{path}}$. (d) All-way propagation back to root node. (e) Neural network evaluator.

With the words Monte Carlo in it, MCTS originally works with the idea of randomization within the section of *simulation*. From a *expanded* node, child nodes are visited stochastically until they reach a final state, and have an evaluation of such node by then. This method certainly works and will get to a converge when a large number of iteration is performed. But it takes a huge amount of time and brings uncertainty to the work, which both are not favored in a real-time system. We would like to use reinforcement learning to get an evaluation neural network as a fast rollout policy to boost up the searching process and reduce the randomness at the same time. Before doing that, we need to construct the Monte Carlo tree under the automated parking framework.

A. Monte Carlo Tree Search Design

A tree node n is the combination of a state s and its observation o. According to our assumption, with a constant observation o, a tree node n has a corresponding state s. A tree node n can exist in one of three possible status: *UNEXPLORED*, *EXPLORED*, and *TRIMMED*. A node n is *UNEXPLORED* when it is just spawned from the *expansion* of its parent, and it transitions to the *EXPLORED* status once it is *selected* and *expanded*. A node n is *TRIMMED*

if it is trimmed due to certain constraints like collision with obstacles, as the red node shown in Fig. 3, or all children of it are trimmed. A tree node n is considered reaching the destination if there is a simple Dubins path[20] can be generated connecting its corresponding pose and the corresponding pose of the destination node.

1) Selection: A *selection* will be performed iteratively until it encounters a *UNEXPLORED* node, as is shown in Fig. 3 (a). In order to fully leverage both the prior evaluation and probabilities, the selected action is guided under a designed PUCT algorithm[3],

$$\underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(n, a') + C_p P(n, a') \sqrt{\frac{N(n) + 1}{N(n, a') + 1}} \tag{3}$$

where $Q(n, a)$ is the expected return when action a is taken under node n, $P(n, a)$ is the policy distribution when action a is taken under node n, N is the visit number of a node or the action under a node, and C_p is an adjustable constant factor to adjust the exploitation preference.

2) Expansion: After *selecting* a node n, an immediate *expansion* follows. A uniform sampling strategy is performed within the action space $\mathcal{A}$, generating a list of front wheel angle δ and a pair of bidirectional travel distance d with the same scale, representing forward and reverse action. Child nodes are therefore spawned from s with the transition function $\mathcal{T}$ under the bicycle kinematic model. A prior probability vector $\boldsymbol{p}$ of each child will be outputted by the neural network estimator f_θ at the same time, as shown in Fig. 3 (b) and (e). If a child node is trimmed by violating certain constraints, its probability share will be split evenly to its living siblings.

3) Simulation: Once $expansion$ is over, we would like to find the value of the selected node n through *simulation*. As what is stated in the *Problem Formulation*, the value(reward) V of a node n can be separated into two parts: the cost it takes from root node to n, and a value scalar v denoting the possibility that n lies in the best path from the start node to the destination node, which is estimated by the same neural network f_θ with $\boldsymbol{p}$ at the same time, as what is shown in Fig. 3 (c) and (e),

$$V = \alpha_0 v + \alpha_1 \mathcal{C}_{\text{path}} \tag{4}$$

where α_0 and α_1 are constant factors which are used to normalize V to $[-1, 1]$, and $\mathcal{C}_{\text{path}}$ is the cost function.

4) Backpropagation: When the value of a node is obtained, we would like to update the interested properties of its ancestors recursively, as what is shown in Fig. 3 (d), known as *backpropagation*.

As shown in Alg. 1, $V(n)$ is the stored estimated value of a node n, and n_0 is the root node. We would like to keep the best estimation if a node is connected to the destination as the value propagated back to enhance the power of successful prior knowledge.

The MCTS algorithm is terminated if it meets one of the following conditions: (1) the tree has fully expanded; (2) it reaches a time limit or a visited node limit; (3) a certain number of paths is found.

Algorithm 1: Backpropagation

```
1 Function Backpropagation(n, V):
2     V(n) ← V
3     while n ≠ n_0 do
4         if n is connected to destination then
5             V ← Max(V, V(n))
6         n, a ← parent of n, corresponding action
7         Q(n, a) ← (N(n,a)Q(n,a) + V) / (N(n,a) + 1)
8         N(n, a) ← N(n, a) + 1
9         N(n) ← N(n) + 1
```

B. Reinforcement Learning Design

MCTS itself can be treated as a natural *policy improvement* operator. Given prior value estimations, it gradually updates its selection policy. After each MCTS is terminated, retrieving the information from the search tree will help us re-estimate the value, as *policy evaluation*. By iteratively executing *policy improvement* and *policy evaluation* process, we have a *policy iteration* framework where MCTS takes part in both processes.

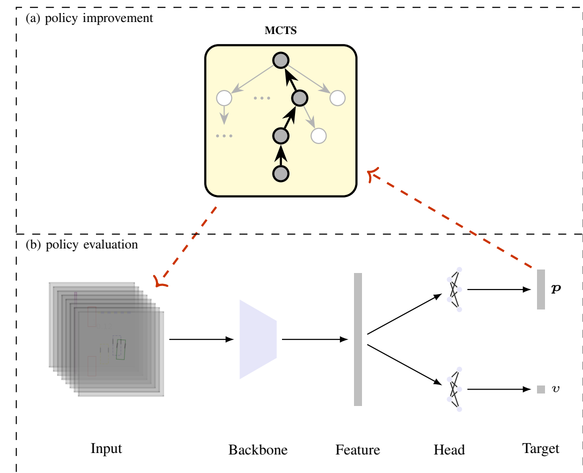


Fig. 4. Reinforcement learning pipeline. (a) Training data is retrieved once MCTS is terminated, as the *policy improvement* finished. (b) The architecture of the *evaluation* neural network is composed of a convolutional backbone and two separated MLP heads. The updated model parameters are used in the next MCTS iteration.

A neural network is used to project a given state to an estimated policy and value. It will be trained against the assembled labels from MCTS outcomes. The general architecture of this network is shown in Fig. 4. The input of the neural network is a stacked tensor consisting of the following layers: (1) the occupancy layers for observation space like different types of obstacles; (2) the occupancy layers for vehicle agent in the current state, parent state and destination state; (3) the numeric layers for other interested current and previous state's properties like gear and steering

wheel angle.

The input tensor is fed into a *backbone* consisting of several convolutional blocks with corresponding batch normalization layer and activation layer. A single dimension hidden *feature* layer is outputted by *backbone*, and goes to two separate multilayer perceptron *heads*, which project the hidden *feature* into the *policy* vector and the *value* scalar respectively.

Given a fair amount of time, feasible paths would be found if they exist. After the search process is terminated, we would have a tree with nodes that lie in feasible paths as good nodes and the out-lied ones as bad nodes. We use the Farthest Point Sampling[21] strategy to sample the same amount of both good and bad nodes for positive and negative data.

For a sampled node n, we would like to use

$$\pi(a|n) = \frac{N(n,a)^{\frac{1}{\tau}}}{\sum_b N(n,b)^{\frac{1}{\tau}}} \tag{5}$$

as the policy label, where τ is the temperature factor which is used to tune the confidence for the next *policy evaluation*. We would like to label the good node as $r = 1$ and the bad node as $r = 0$, representing the chance if there is a good path from the node to the destination.

The network parameter θ is then trained against a loss function that simply sums up the cross-entropy loss of policy *head* and the mean-square loss of value *head*, as

$$\underset{\theta}{\operatorname{argmin}} \; -\boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + (v - r)^2 \tag{6}$$

The network is trained after each round of MCTS, and its updated parameters are used for the next round of MCTS exploration, as what is shown in Fig. 1.

V. EXPERIMENTS

The training and inference experiments are evaluated on a computer with 32GB memory, an AMD Ryzen 7 3700x CPU, and an Nvidia GeForce RTX 3060 GPU.

Derived from real road test scenarios, we constructed thousands of simulated parking scenarios in different types, enumerating the critical features one could have like the size of the parking spots, obstacles' poses, agent's starting poses, etc. The generated scenarios are separated into training, test and validation datasets with a ratio of 70%:15%:15%.

The performance of the MCTS in the inference stage is evaluated based on the planning time it takes to generate a feasible path that meets the path generated by the Hybrid A* [11] algorithm in its quality. We recorded the behavior of Hybrid A* and MCTS at different training steps through the validation dataset, as what is shown in Fig. 5 (a). The discretization setup is 0.1 m for the position and 0.01 Rad for the orientation. The planning time MCTS takes to find a path drops quickly in the opening stages, and gradually converges to a stable condition. In the end, the median planning time required by MCTS through the dataset is only 7.2% compared to the time taken by Hybrid A*. The performance of trained MCTS and Hybrid A* under multiple discretizations is shown in Fig. 5 (b), which suggests that model can be easily scaled up to different setups.

The algorithm has been deployed on autonomous vehicles for daily operations. We selected three distinct parking scenarios from real road test data for analysis. The path planning is tailored to initiate from the parking spot towards the vehicle's current position to ensure rapid convergence.

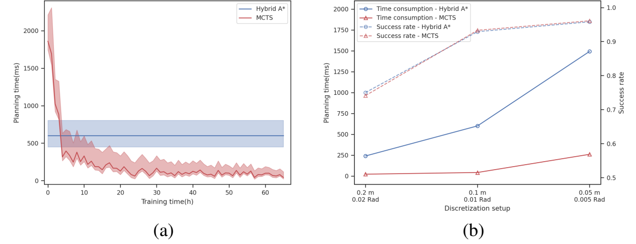
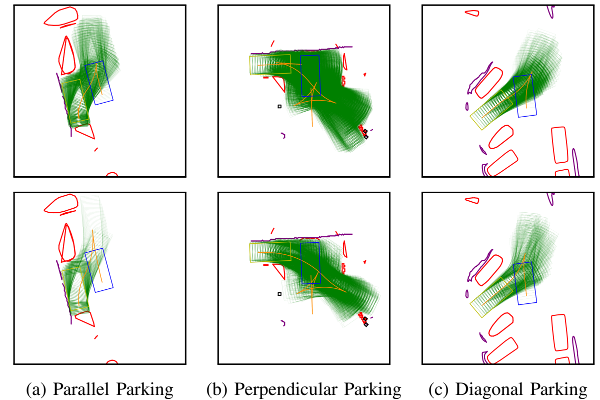


Fig. 5. (a) Training metric. The darker line is the median over the validation dataset, and the pale shaded area is formed by the 10th and 90th percentiles. The blue one is the planning time which Hybrid A* algorithm takes while the red one is the performance of MCTS at different training steps. (b) Planning time and success rate comparison between trained MCTS and Hybrid A* under different discretization setups over the validation dataset.



(a) Parallel Parking (b) Perpendicular Parking (c) Diagonal Parking

Fig. 6. Comparison between Hybrid A* (above) and MCTS (below) results. The red, purple, and black polygons are different types of obstacles. The agent needs to park from the blue box to the yellow box. The orange curves are the generated path while the pale green boxes are the visited search nodes respectively.

A. Parallel parking with narrow spot in length

As shown in Fig. 6 (a), Our agent would like to take a parallel parking into a spot that is close to a curb and narrow in length. The Hybrid A* makes a huge effort to manage to get out from the spot, leading to unnecessary gear changes within the spot. The MCTS however, uses its prior knowledge to navigate out quickly with clean moves, nudging the surrounding obstacles and makes to the destination confidently with no other tryouts.

B. Perpendicular parking under complex environment

Fig. 6 (b) records a rare situation where our agent would like to park vertically into a spot, while there is a dead end ahead and numerous obstacles around. Under such a scenario, no well-rounded heuristic function could be applied to Hybrid A*, reducing its exploitation preference nearly to the level of Dijkstra's. The search space gets almost to exhaustion making the planning time very long. MCTS well balances its exploitation and exploration tendency in this case, managed to generate a human-like path with less than 500ms.

C. Diagonal parking with enough space

When it comes to an easier task where a clear exploitation suggestion like the one in Fig. 6 (c), MCTS still has a solid performance compared to the result of Hybrid A*, generating similar solutions within the same amount of time.

VI. CONCLUSION

In this paper, we present a method that integrates reinforcement learning into the Monte Carlo tree search algorithm to expedite automated parking tasks. We formulate path planning as a Markov Decision Process with a bicycle model as the vehicle kinematics. Through the design of the Monte Carlo search tree structure and relative strategies in its iterative steps, we ensure adaptability within the path planning scheme. Furthermore, a neural network is used to generate the policy distribution and estimate the corresponding value of a state within the MCTS. We implement a reinforcement learning pipeline with MCTS serving as a policy improvement operator on our generated dataset, and achieved an excellent training result. The algorithm has been successfully deployed on real-time autonomous vehicles and greatly enhances the experience of the automated parking product.

REFERENCES

[1] J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer, 2020, pp. 194–210.

[2] Y. Wei, L. Zhao, W. Zheng, Z. Zhu, J. Zhou, and J. Lu, "Surroundocc: Multi-camera 3d occupancy prediction for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 21 729–21 740.

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

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

[5] C. Sungwoo, C. Boussard, and B. d'Andréa Novel, "Easy path planning and robust control for automatic parallel parking," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 656–661, 2011.

[6] I. E. Paromtchik and C. Laugier, "Autonomous parallel parking of a nonholonomic vehicle," in *Proceedings of Conference on Intelligent Vehicles*. IEEE, 1996, pp. 13–18.

[7] R. Pepy, A. Lambert, and H. Mounier, "Path planning using a dynamic vehicle model," in *2006 2nd International Conference on Information & Communication Technologies*, vol. 1. IEEE, 2006, pp. 781–786.

[8] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan," *Algorithmic and computational robotics*, pp. 303–307, 2001.

[9] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on control systems technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[10] J. E. Doran and D. Michie, "Experiments with the graph traverser program," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 294, no. 1437, pp. 235–259, 1966.

[11] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[12] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[13] Z. Liang, G. Zheng, and J. Li, "Automatic parking path optimization based on bezier curve fitting," in *2012 IEEE International Conference on Automation and Logistics*. IEEE, 2012, pp. 583–587.

[14] G. Notomista and M. Botsch, "A machine learning approach for the segmentation of driving maneuvers and its application in autonomous parking," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 7, no. 4, pp. 243–255, 2017.

[15] 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, pp. 354–359, 2017.

[16] P. Weingertner, M. Ho, A. Timofeev, S. Aubert, and G. Pita-Gil, "Monte carlo tree search with reinforcement learning for motion planning," in *2020 IEEE 23rd international conference on intelligent transportation systems (ITSC)*. IEEE, 2020, pp. 1–7.

[17] S. Song, H. Chen, H. Sun, M. Liu, and T. Xia, "Time-optimized online planning for parallel parking with nonlinear optimization and improved monte carlo tree search," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2226–2233, 2022.

[18] 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, pp. 484–489, 2016.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, pp. 235–256, 2002.

[20] H. H. Johnson, "An application of the maximum principle to the geometry of plane curves," *Proceedings of the American Mathematical Society*, vol. 44, no. 2, pp. 432–435, 1974.

[21] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.