Deep Reinforcement Learning combined with RRT for trajectory tracking of autonomous vehicles.

Bálint Kővári[1] Gergő Bálint Angyal[1] Tamás Bécsi[1]

***Abstract*— Sample inefficiency is a long-standing problem in Deep Reinforcement Learning based algorithms, which shadows the potential of these techniques. So far, the primary approach for tackling this issue is prioritizing the gathered experiences. However, the strategy behind collecting the experiences received less attention, but it is also a legitimate approach for prioritizing. In this paper, the Rapidly exploring Random Trees algorithm and Deep Reinforcement Learning are combined for the trajectory tracking of autonomous vehicles to mitigate the issues regarding sample efficiency. The core of the concept is to utilize the tremendous explorational power of RRT for covering the state space via experiences for the Agent to diversify its training data buffer. The results demonstrate that this approach outperforms the classic trial-and-error-based concept according to several performance indicators.**

I. INTRODUCTION

A. Motivation

Autonomous Driving has recently gained much attention among researchers since it promises tremendous improvements in many fields concerning our mobility. Deep Learning (DL) based techniques are vital for reaching these goals, especially Deep Reinforcement Learning (DRL). DRL can outperform classic control methods in several domains, such as motion-planning [1] or traffic signal control [2] and so forth. However, DRL techniques also have shortcomings, mostly in sample efficiency and the potential for using an agent in the real-world trained with simulated environments. This is called the reality gap, which is often tackled with domain randomization [3]. Sample inefficiency is strongly tied to the utilized exploration technique since this concept influences the quality of the gathered training data. Moreover, the poor quality of the training data collected through insufficient exploration can make the agent end up in some local optima. Nevertheless, the pure concept of diversity in training data collection is also lacking since there is no connection with the agent and what kind of experiences it needs. Thus pure state space coverage is not the answer. It is basically the same problem as between Prioritized Experience Replay (PER) [4] and uniform sampling from the training data buffer. Consequently, our goal is to combine DRL and RRT in a way where the agent can drive the tree's growth based on its current knowledge.

*The research carried out by Budapest University of Technology and Economics was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Autonomous Systems National Laboratory Program. organization.
Supported by the ÚNKP-22-5-BME-272 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund

[1]Bálint Kővári, Gergő Bálint Angyal and Tamás Bécsi is with Faculty of Transportation Engineering and Vehicle Engineering, Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, Műegyetem rkp 3, H-1111 Budapest Hungary `[kovari.balint, becsi.tamas]@kjk.bme.hu, angyal.balintgergo@edu.bme.hu`

B. Related work

Pure DRL-based trajectory tracking solutions utilize two main approaches in terms of state representation. In the first group, one can utilize an image-like state representation that comes from a front-facing camera containing the vehicle's current state and some lookahead information in a limited manner [5], [6]. In the second group, the state representation is a feature-based value vector that almost always contains the vehicle's distance from the center line, and its yaw angle or relative yaw angle [7]. In some cases, the authors also utilize lookahead information that somehow encodes how the trajectory ahead will evolve, [8].

One of the first combination of RL and sampling-based motion planning methods is the PRM-RL concept [9], which is a hierarchical approach for long-range navigation tasks. The core of the idea is that the Probabilistic Road Maps (PRM) algorithm uses a trained RL agent to check whether the connection between two points is possible, substituting the simple collision-free straight line in C-space. The resulting method improved the potential of both RL agents and sampling-based planners in long-range navigation tasks. The combination of DRL and RRT has been proposed previously but differently. The authors in [10] proposed an algorithm called Planning for Policy Search (PPS), which is also used as a training sample generator for DRL algorithms with an Affine Quadratic Regulator (AQR) based cost function that determines the closest tree node. A Model Predictive Control (MPC) based local planner created the next node added to the existing tree. The authors applied it to OpenAI's classic control environments that utilize continuous action spaces. In [11], the authors train an RL algorithm that can provide a policy that avoids obstacles for a motion planning task from a start state to a goal state. Then with the trained agents, additional training data is gathered for training a Reachability Estimator that can accurately measure the Time To Reach cost of a state. The authors combine these components in the RRT, the trained RL agent used as a local planner to direct tree growth into promising regions, and the estimator as a cost function. The RL-RRT created shorter paths and used less planning thanks to these components. The authors in [12] proposed the Simultaneous Learning and Planning Algorithm (SLPA) for a robot navigation problem that combines Backward Q-learning (BQL) and RRT*, which, compared to

Manuscript 381 submitted to 2023 European Control Conference (ECC).
Received November 27, 2022.

RRT, provides optimized path lengths. The core of the idea is that RRT* balances the exploration-exploitation trade-off for BQL. The results show that SLPA can outperform the original RRT* algorithm. Although the method may have some scalability issues since BQL is used in tabular form, there is no generalization as such one would get with function approximators.

C. Contribution

As presented in the previous subsection, several papers focused on the combination of DRL and RRT. The main difference is that most papers deal with autonomous robot navigation or classic control problems. Furthermore, the combination of the methods is used to improve RRT's planning capabilities in terms of time and path quality with already trained agents. The idea of [10] is the most similar to this paper since it uses the RRT algorithm as a training data generator for DRL. However, the authors in [10] utilize an AQR-based cost function and an MPC controller as a local planner. At the same time, the method presented in this paper uses the predicted values of the Neural Network (NN) trained with RL for determining the exploration point, while a rollout strategy substitutes the local planner. Consequently, the key element of our paper, which is our main contribution that the process responsible for training data generation is interconnected with the current knowledge of the agent. Furthermore, the side contributions of this paper are the application it self, and the inflection point based rewarding concept.

II. ENVIRONMENT

In the RL concept, one of the crucial parts is the environment since this component realizes the state transition triggered by the action, and after that, it provides the reward signal along with the state representation. In the context of trajectory tracking, the environment has to contain a random trajectory generator that enables the thorough evaluation of the trained models. In this paper, a highly parameterizable trajectory generator has been implemented, where the curvature of the turns and also the number of turns can be controlled. An instance of a trajectory is shown in Figure 1. Another important part that also determines the complexity

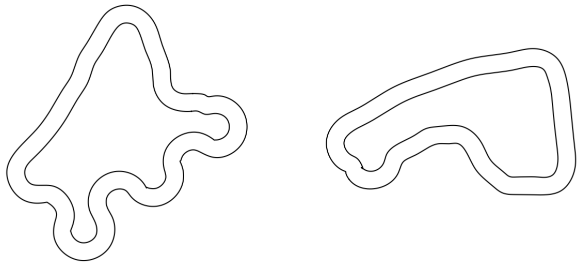
Fig. 1. Randomly generated trajectories.

of the problem is the used vehicle model, which is, in our case, a simple rigid kinematic vehicle model without slip. The vehicle model is looks as follows:

$$\dot{x_v} = v \cos \Psi_v \tag{1}$$

$$\dot{y_v} = v \sin \Psi_v \tag{2}$$

Between the δ steering angle and the radius of the curve the connection can be calculated based on the (3) equation.

$$\tan \delta = \frac{L}{R} \tag{3}$$

(4) equation calculates the Ψ yaw rate of the vehicle.

$$\dot{\Psi}_v = \frac{v}{R} \tag{4}$$

Thanks to the (3) and (4) equation a more practical version can be formulated which is shown in the (5) equation is .

$$\dot{\Psi}_v = \tan \delta \frac{v}{L} \tag{5}$$

However, with such a model, the trajectory problem is extremely easy since, without inertia, the agent can always steer back from the edge of the lane. This is avoided by limiting the steering angles to $< -0.1, 0, 0.1 >$ rad, and thanks to that, the agent has to "think ahead" in every turn if it wants to stay in the lane. This limitation makes the problem more difficult in terms of learning, but the vehicle model is still extremely lightweight from a computational point of view.

A. State representation

State representation is the only information an agent has for choosing the possible actions in a given state. The state representation in the trajectory tracking problem comprises two components. The first one represents the current situation of the vehicle in the lane. In our case, it is done using the distance from the centerline and the relative yaw angle. The distance is scaled with half of the lane width, and its sign is decided based on which wrapping curve the vehicle is closer to. The relative yaw angle is scaled with $\pi/2$. The second component is the lookahead information that provides information about the trajectory change ahead. This is done by calculating relative yaw angles into the points of the trajectory ahead that are equally distanced from each other. This lookahead information shows the agent how it should change its orientation in the future to align with the preferred direction of travel.

B. Reward strategy

The reward signal is also part of the abstraction used for formalizing an RL problem. This scalar feedback signal characterizes the immediate consequences of a chosen action or, at the end of an episode, the quality of the entire trajectory performed from the initial state s_0 until the terminal state encounters. In the trajectory tracking problem where only lateral control is realized -which is precisely our case- mostly in the literature, the in-lane position and the relative yaw angle is used for calculating the reward since these signal demands the agent to stay in the middle of the lane. Still, unfortunately, this approach is highly deceptive when there is some inertia or limitations in the system. Consider a simple scenario with a sharp turn ahead. Obviously, the agent should not enter the turn from the center of the lane since it will slip out, but if it does not, it gets less reward. Consequently, there is a clear contradiction between what is required and what is demanded from the agent. A better approach is to reward the agent at some checkpoints because it gets leeway about how it behaves between checkpoints until he reaches them. In this work, this idea is followed. What matters is how the vehicle enters and leaves the turns; hence this paper proposes a checkpoint-based rewarding concept that only gives non-zero rewards when the agent is at the inflection points of the trajectory and when the agents leave the lane. Thanks to that, the contradiction can be resolved.

C. Action-space

The action space is as simple as possible, which matches with the vehicle model, hence three discrete steering angles $< 0.1, 0, 0.1 >$ rad.

III. ALGORITHMS

A. DEEP REINFORCEMENT LEARNING

Deep Reinforcement Learning (DRL) has recently gained the attention of researchers thanks to its astonishing results in several complex and challenging domains that have lured interest for a long time. Such as the game of GO [13], the highly difficult videogame called Montezuma's revenge [14], or discovering novel algorithms for fundamental tasks [15]. The idea behind DRL is that the agent and the environment are separate entities, and the learning process is formulated as interactions between them. The interactions are composed with the help of the DRL's three main abstractions: the state representation, which is the sole information an agent has for decision-making, and the action space, which contains all possible actions in the given domain. The rewards signal is a scalar feedback value that immediately characterizes the action's effect and the entire trajectory made of S, A, R, S' at the end of the episode. The agent aims to maximize the cumulated reward during the episode in this concept.

$$G = \sum_{t=1}^{T} \gamma^t r_t, \tag{6}$$

Thus it has to develop a suitable behavior the accomplish as such. The interactions between the agent and the environment are shown in Figure 2. The main drawback of DRL is sample

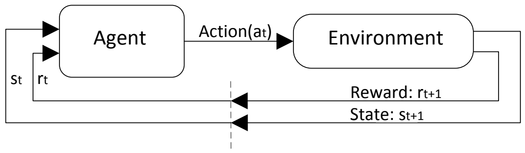

Fig. 2. Reinforcement Learning training loop.

efficiency and, to be precise, the lack of that. The rewarding concept tremendously influences sample efficiency since reward strategies are heuristics designed by the researchers' intuitions. Reward signals can be deceptive and sparse, making it difficult for the agent to find a reliable source of knowledge that is not seriously underrepresented in the agent's experience memory buffer. Consequently, experience prioritization concepts can mitigate issues regarding sample efficiency, which hugely influences the final performance. Experience prioritization strategies can be divided into two main groups. The first group of methods is concerned with the sampling strategy regarding the experiences that are already stored in the memory see [4]. The second group of methods tries to decide wisely which interaction's experience should be stored in the memory buffer and used for training [16]. Thus these methods are concerned with the exploration technique used by the agent. Our main goal is to develop a method that combines different methods along their advantages through collecting experiences to mitigate the issues regarding sample efficiency.

B. Deep Q-network algorithm

The DRL algorithm that is used in this research is a value-based one called Deep Q-Network (DQN). DQN algorithm is the method that reached the first tremendous success in solving complex sequential decision-making problems on a level that can be compared or, in several cases, outperform human experts [17]. The DQN algorithm tries to train the neural network to predict the action-value function in every possible state. The action values show the amount of reward that can be gathered through the episode until a terminal state is encountered. Thus, these action values are similar to an assessment of the given state since they provide information about the actions' worth in the long run. For training such a regression problem, the target values are calculated with Bellman's equation:

$$Q(s_t, a_t; \theta_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t; \theta_t^-). \tag{7}$$

Where θ' represents the weights of the target network that is used in the DQN to avoid divergence of the algorithm.

C. Rapidly Exploring Random Trees

The Rapidly Exploring Random Trees (RRT) are undoubtedly the most popular sampling-based motion planning algorithm thanks to their straightforward concept, which makes it easy to implement and it also has great convergence properties [18]. The steps of the algorithms are the following.

- Chose a point randomly in the configuration space.
- Find the closest node to the randomly chosen point based on some distance measure -cost function-.
- From the closest node, take a step into the direction of the randomly chosen point with a local planner.
- Add this newly generated point to the tree.

An example of the RRT algorithm are shown on Figure 3 for a lane keeping problem. Another important virtue of the RRT algorithm, which made it interesting for this research, is the exploration potential that makes it possible to cover the configuration space with a small number of samples

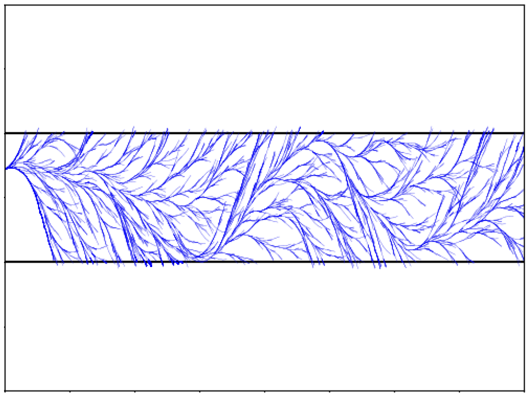

Fig. 3. RRT for lane keeping

compared to other methods. This virtue can be extremely beneficial since efficient exploration can be interpreted as a synonym for diverse training data, which can be a deal breaker in a state space of sparse and deceptive reward signals.

D. The combined method

The combined method integrates the RRT algorithm into DRL's training process. In the original idea, DRL agents gather their training data via trial and error and use some exploration techniques to make sure that the state space is sufficiently covered. But as mentioned earlier, these basic exploration techniques can not tackle the problem of spars reward signals and the lack of the training data's diversity extracted from the environment.

In the meantime, one of the main virtues of the RRT algorithm is its fascinating potential for exploring state spaces through its iterative process. If one analysis the RRT method thoroughly, it can be observed that it is a training sample factory for our agents' memory buffer. Since any node pair carries the same information as a training sample $< S,A,R,S' >$, the only thing needed is the reward signal, which can be easily collected from the environment. The trial and error-based training data generation procedure is entirely omitted by utilizing this approach. Instead, the environment is used generatively to support the RRT's iterative process, which is changed a bit. The first change is about the calculation of the cost that is used for finding the closest node. The combined method generates a random point as the original RRT. Still, it predicts the Q-values of the generated point, and the closest node is calculated based on the difference of the nodes' averaged Q-values. Hence the neural network determines where it should start the exploration from. Then a rollout is conducted with a pre-determined depth (10 actions) which process substitutes the original local planner. These actions are chosen based on a probability distribution proportional to the predicted Q values. Furthermore, the combined method is complemented with one additional step, which turns the newly added node and its parent into a training sample and stores it in the agent's memory buffer.

IV. Results

A. Training

In the training phase, the goal is to ensure that the results will be representative. Hence both the presented method and the original RL training procedure have the same opportunities. This is ensured by using one closed track for training. Both methods start from the same initial states and collect the same amount of experience through their procedure. With the same hyperparameters, these methods can train their network the same number of times -epochs. Hence, the only difference is how they gather their data; thanks to that, the data's quality will decide.

B. Statistical evaluation of the results

The statistical comparison is conducted by measuring the performance of both algorithms on the same one thousand randomly generated trajectories, while each of them should make 200, 500, and 1000 steps on the given racetrack. The success ratios and the average steps performed without leaving the lane are shown in Table I. It can be said

TABLE I
Comparison of statistical results on 1000 trajectories

-	200 steps	500 steps	1000 steps
DQN Avg step.	182.5	385.0	603.2
RRT-DRL Avg step.	190.9	437.5	749.4
DQN Success [%]	81.8	52.4	41.4
RRT-DRL Success [%]	91.1	69.5	60

that the presented method outperforms the original DQN algorithm, and the difference between the performance is rising along the steps needed to be performed. It is also important that when the agents have to perform 1000 steps on a given trajectory which basically means a full lap, then the difference between the success rations are nearly 19%. Another statistical comparison is created for thoroughness regarding the performance of the different methods. The quality of the steps taken during the one thousand randomly generated episodes is assessed with the following equation:

$$score = \cos \Psi_{rel} + |d| \qquad (8)$$

This equation calculates the score based on how close the vehicle is to the centerline and how much the vehicle's orientation is aligned with the preferred travel direction at the trajectory's given point. The results are shown in Figure 4.

Figure 4 clearly shows that the combined method reached superior performance since it gathers a greater amount from the highest scores and has a higher average score.

C. Strategical evaluation of the results

In the strategical comparison, the agents' behavior is investigated in the same scenarios. The first scenario is shown in Figure 5. As you can see, the agent trained with the proposed method utilizes the lookahead part of the state representation much more effectively. It can be observed

Manuscript 381 submitted to 2023 European Control Conference (ECC).
Received November 27, 2022.

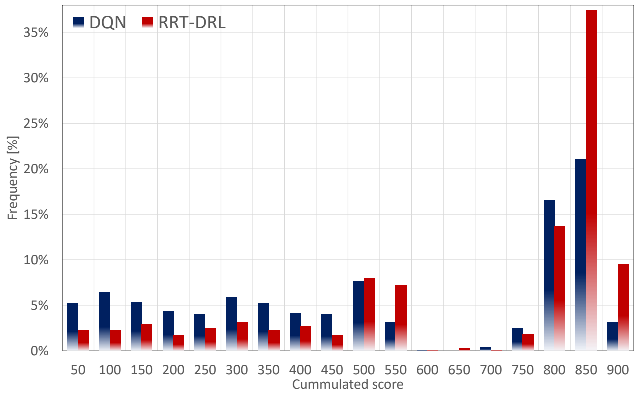

Fig. 4. Statistical comparison of the performances.

that it alters from the center of the lane in advance to the upcoming turn. This behavior makes it possible to reach a better success rate than the DQN algorithm despite the limited steering angles. Of course, the same can be seen in the behavior of the DQN algorithm but in a much more limited manner. It is also important that the agent returns to the middle of the lane between two turns when there is a somewhat straight trajectory part, which means it has learned which kind of behavior belongs to which scenario. Another example of the described behaviour is shown in Figure 6. In some turn combinations, the DQN algorithm can not tackle the challenge of staying in the lane, primarily because of the poor utilization of the lookahead information. However, the agent trained with the proposed method can easily cope with the challenge and keeps the vehicle in the lane, which clearly shows that it is developed a better understanding of the meaning of the lookahead information. This scenario is shown in Figure 7.

V. CONCLUSIONS

This paper proposes combining the RRT algorithm with DRL through some modifications in the RRT algorithm's iterative process. The main changes include calculating the distance between the randomly generated node and the tree nodes, which is now done based on the predicted Q-values of the neural network. Thus the neural network determines the node where the exploration starts. The second change is in the last part of the RRT algorithm, which defines how to add a node to the tree. In the proposed method, not just one node is added but a series of nodes through a rollout procedure where the actions are chosen proportionally to their predicted Q-values. The presented evaluation showed that by training both the baseline DQN method and the proposed method under the same circumstances, the proposed method could outperform the simple DQN algorithm. The performance evaluation is conducted on the same one thousand randomly generated trajectories, which shows the generalization potential of the used abstractions and ensures that the trained networks are robust and the results are representative. In our future endeavors, the potential of the proposed method will be further assessed but in different domains to see whether the results presented in this paper can be transferred to other domains. As for trajectory tracking, we would like to make the problem more difficult and closer to the real-world scenario by switching to a dynamic bicycle model and changing the action space to a continuous one.

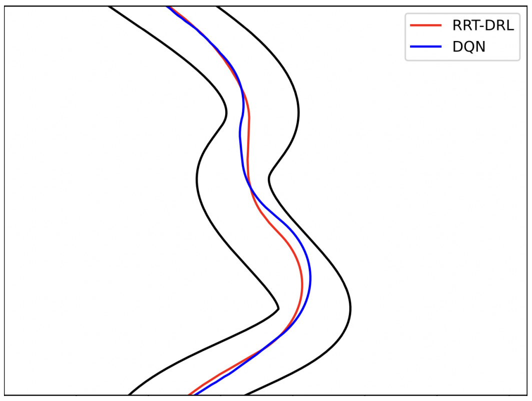

Fig. 5. Strategical comparison in turn combinations

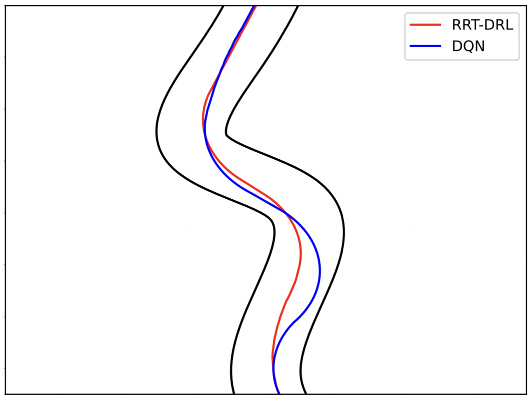

Fig. 6. Strategical comparison in a sharp turn combination.

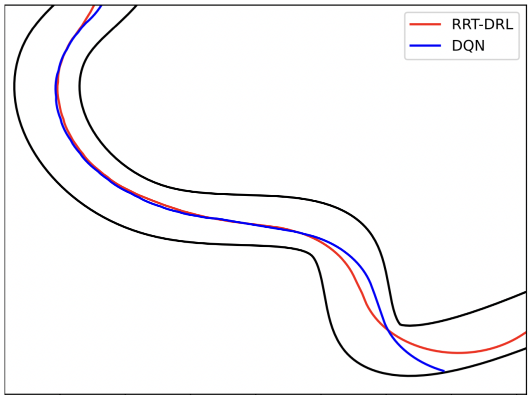

Fig. 7. Strategical comparison in case of lane departure.

REFERENCES

[1] X. Zhang, Y. Jiang, Y. Lu, and X. Xu, “A receding-horizon reinforcement learning approach for kinodynamic motion planning of autonomous vehicles,” *IEEE Transactions on Intelligent Vehicles*, 2022.

Manuscript 381 submitted to 2023 European Control Conference (ECC).
Received November 27, 2022.

[2] B. Kővári, L. Szőke, T. Bécsi, S. Aradi, and P. Gáspár, “Traffic signal control via reinforcement learning for reducing global vehicle emission,” *Sustainability*, vol. 13, no. 20, p. 11254, 2021.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[5] N. Xu, B. Tan, and B. Kong, “Autonomous driving in reality with reinforcement learning and image translation,” *arXiv preprint arXiv:1801.05299*, 2018.

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

[7] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “End-to-end deep reinforcement learning for lane keeping assist,” *arXiv preprint arXiv:1612.04340*, 2016.

[8] B. Kővári, F. Hegedüs, and T. Bécsi, “Design of a reinforcement learning-based lane keeping planning agent for automated vehicles,” *Applied Sciences*, vol. 10, no. 20, p. 7171, 2020.

[9] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, “Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5113–5120.

[10] J. J. Hollenstein, E. Renaudo, M. Saveriano, and J. Piater, “Improving the exploration of deep reinforcement learning in continuous domains using planning for policy search,” *arXiv preprint arXiv:2010.12974*, 2020.

[11] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, “Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[12] A. K. Sadhu, S. Shukla, S. Sortee, M. Ludhiyani, and R. Dasgupta, “Simultaneous learning and planning using rapidly exploring random tree* and reinforcement learning,” in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 71–80.

[13] 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.

[14] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “First return, then explore,” *Nature*, vol. 590, no. 7847, pp. 580–586, 2021.

[15] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[16] Z.-W. Hong, T. Chen, Y.-C. Lin, J. Pajarinen, and P. Agrawal, “Topological experience replay,” *arXiv preprint arXiv:2203.15845*, 2022.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

Manuscript 381 submitted to 2023 European Control Conference (ECC).
Received November 27, 2022.