2022 7th IEEE International Conference on
Advanced Robotics and Mechatronics
July 9-11, 2022, Guilin, China

2022 International Conference on Advanced Robotics and Mechatronics (ICARM) | 978-1-6654-8306-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICARM54641.2022.9959459

Improving the Ability of Robots to Navigate Through Crowded Environments Safely using Deep Reinforcement Learning

Qinfeng Shan, Weijie Wang, Dingfei Guo, Xiangrong Sun and Lihao Jia

***Abstract*— Autonomous robot navigation in unpredictable and crowded environments requires a guarantee of safety and a stronger ability to pass through a narrow passage. However, it's challenging to plan safe, dynamically-feasible trajectories in real-time. Previous approaches, such as Reachability-based Trajectory Design (RTD), focus on safety guarantee, but the lack of online strategy always makes the robot fail to pass through a narrow passage. This paper proposes to learn a policy that guides the robot to make successful plans using deep Reinforcement Learning (RL). We train a deep network based on the RTD method to create cost functions in real-time. The created cost function is expected to help the online planner optimize the robot's feasible trajectory, satisfying its kino-dynamics model and collision avoidance constraints. In crowded simulated environments, our approach substantially improves the planning success rate compared to RTD and some other methods.**

I. INTRODUCTION

Autonomous mobile robots, like autonomous cars, are always required to make successful navigation in crowded environments. Safe control of autonomous mobile robots for use in crowds directly impacts public safety, especially when more complicated environments such as narrow passages are considered. Without safety guarantees, autonomous mobile robots are prone to collisions in crowded environments, but robots often behave too conservatively with some algorithms based on safety guarantees. For instance, when facing a narrow passage, some planners with safety guarantees will make failed plans, which wastes the robots a lot of time to replan and turn around.

Receding-horizon strategies are often used in robot navigation in unpredictable environments with limited sensor horizons because the robot's final goal usually exceeds the sensing range. The planner always uses a cost-to-go heuristic to optimize subgoals, which is easy to accomplish in a static environment. The specified designed cost function may not suit different kinds of obstacles. Some previous works try to improve robots' ability to the neighbor selection, combining a more powerful local planner with hybrid sampling to solve the narrow passage problem [1]. Since the environment is unpredictable and complex, it is hard to preplan a feasible trajectory for every kind of obstacle. It is challenging to extend some algorithms designed for specific scenarios to complex environments. The planner needs an intelligent policy to help it make correct decisions when the robot faces different kinds of obstacles.

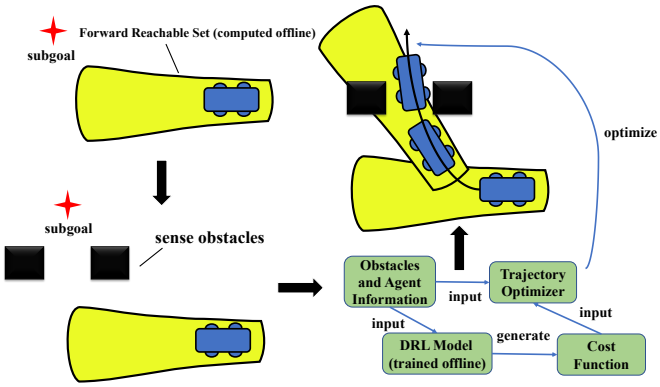

Figure 1: The schematic diagram of robot trajectory planning online. The forward reachable set is computed offline, which provides the robot with all feasible parameterized trajectories. Once the sensor detects the obstacles ahead, the trained deep reinforcement learning model will generate the cost function for the trajectory optimizer. Then the planner uses the generated cost function to optimize a feasible trajectory.

Learning methods can be used to improve the robot's motion [2]. Deep reinforcement learning has recently been used in autonomous driving [3], [4] and collision avoidance [5], [6], [7]. The agents learn a policy from a long-term offline training phase to adapt to different scenarios through simulation in the randomly generated environments. Still, the way learning is implemented does not give safety guarantees with respect to the robot dynamics. Reachability-based methods precompute a reachable set offline to ensure collision avoidance at runtime, which introduces some conservatism into trajectory planning [8]. Safe and efficient planners are what we want.

In this paper, we propose a feasible trajectory planner with a safety guarantee for mobile robots crossing complex static environments to improve the successful planning rate, as depicted in Figure 1. Based on the RTD method [8], we precompute the Forward Reachable Set (FRS) and use it to ensure safety during the offline RL training. After offline training, the robot learns a policy that uses the current state of the robot and obstacles generating cost functions.

Q.Shan, W.Wang, D.Guo, X.Sun and L.Jia are with the Research Center for Brain-inspired Intelligence, Institute of Automation, Chinese Academy of Sciences, Beijing 100190 China and Centre for Artificial Intelligence and Robotics, Hong Kong Institute of Science & Innovation, Chinese Academy of Sciences. Q.Shan and W.Wang are also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049 China. e-mail:shanqinfeng2020@ia.ac.cn.

Corresponding author is Lihao Jia.

This work was supported in part by the National Natural Science Foundation of China under Grant 61933001, Grant 61903361, Grant 62003340.

This work was supported in part by InnoHK program.

With the generated cost functions, the planner optimizes to generate a safe and feasible trajectory during online running. Our approach improves the success rate and saves the running time for robots in crowded situations, especially when passing through narrow passages.

Finally, we design some experiments and give results to demonstrate our strength compared to some other trajectory planning methods in crowded environments. We give some specially designed scenarios to show the advantage of our approach in passing through narrow passages.

II. RELATED WORK

Several collision avoidance methods have recently been used in planner design to ensure safety guarantees. Sample-based algorithms are widely used in robot navigation, like the Rapidly-exploring Random Tree (RRT) [9] algorithm and Probabilistic Road Maps (PRM) [10]. Sample-based algorithms usually linearize the agent's model for collision checking. They have difficulty dealing with high-dimensional models.

Model Predictive Control (MPC) with online optimization is used to solve trajectory planning for nonlinear vehicle robots to respect some non-convex constraints [11], [12]. Computation time is a crucial challenge for MPC methods to ensure real-time safety and feasibility guarantees, and reinforcement learning has recently been used with MPC to overcome this disadvantage. [13] replaced the MPC's cost function with the value function which is learned from offline RL training. [14] suggested combining MPC and RL to get a better planner and [15] proposed a goal-oriented model predictive control method, using RL to train a policy offline to optimize a feasible subgoal for the MPC online, which is similar to our approach.

Reachability-based methods focus on safety guarantees for robots. They usually precompute some reachable sets for the robot and then use them to ensure collision avoidance at runtime. Hamilton Jacobi (HJ) reachability analysis [16]–[18] suffers from the curse of dimensionality, so this kind of reachable analysis is difficult to extend to high-dimensional models. Because the HJ reachability's computation is expansive, [19] proposes an online algorithm to update the BRS using the sensed environmental information. Reachability-based Trajectory Design (RTD) methods compute the Forward Reachable Set (FRS) offline and use FRS to perform trajectory optimization online [8], [20]–[23]. The FRS contains all parameterized trajectories which the robot can follow. RTD can also supply safe trajectories for robots in offline reinforcement learning [24]. However, these methods always introduce conservatism into the planner. It makes the planner always fails to plan feasible trajectories for robots crossing narrow passages. Our approach is based on RTD methods to ensure safety guarantees and is combined with RL to overcome this challenge. We use RL to train a model offline to generate cost functions online, allowing the planner to optimize a safe and feasible trajectory when the robot faces different obstacles.

III. PRELIMINARIES

This section introduces the Forward Reachable Set (FRS) briefly. We follow the implement from [8], define FRS for our robot, compute it offline, and use it to get the safe trajectory set for online planning.

A. Notation

The real numbers are defined as $\mathbb{R}$ and the natural numbers are defined as $\mathbb{N}$. $\mathbb{R}^n$ is the euclidean space in $n \in \mathbb{N}$ dimensions. Having a set K, the Lebesgue measure on K is defined as λ_K. The volume of K is $vol(K) = \int_K \lambda_K$. The $\{s, a\}$ stands for the state and action variables in the RL training.

B. Dynamic Model

Consider a dynamic model for the vehicle robots:

$$\dot{x_{hi}}(t) = f_{hi}(t, x_{hi}(t), u(t)) \tag{1}$$

where $f_{hi} : [0, T] \times X_{hi} \times U \to \mathbb{R}^n$, $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$. T is the planning time horizon so $T > 0$. It's challenging to compute the reachability through this kind of high-fidelity model, so here comes a simple trajectory-producing model [8] :

$$\begin{bmatrix} \dot{x}(t) \\ \dot{k}(t) \end{bmatrix} = \begin{bmatrix} f(t, x(t), k(t)) \\ 0 \end{bmatrix} \tag{2}$$

where $f : [0, T] \times X \times K \to \mathbb{R}$. X is a subspace of X_{hi}, where $dim(X) \leq dim(X_{hi})$. f is a simpler model to generate trajectory while f_{hi} is the dynamic model.

We also need a tracking error function to constrain the tracking error and some other gaps between (1) and (2). There are two ways to get the tracking error function g, one is simulating the dynamic model (1) and the other is using Sums-of-Squares optimization to compute it [25]. For each $i \in 1, ..., n_s$, there exists $g_i : [0, T] \times X \times K \to \mathbb{R}$ such that:

$$\max_{x_{hi} \in A} |x_{hi,i}(t; x_{hi,0}, k) - x_i(t; x_0, k)| \leq \int_0^t g_i(\tau, x, k) d\tau, \tag{3}$$

for all $x \in X$, $t \in T$, and $k \in K$ where $A = \{x \in X | x_{hi} = x_i\}$ is the set that the model (1) and (2) have the shared states. Tracking error function is $g = [g_1, ..., g_{n_s}]^T$ and g is Lipschitz continuous in t, x and k.

The function g is used to close the gap between the models, so the trajectory-tracking model can be written as [8]:

$$\dot{x}(t, x(t), k, d) = f(t, x(t), k) + g(t, x(t), k) \circ d(t) \tag{4}$$

where $d(t) \in [-1, 1]$ for almost each $t \in [0, T]$. The d is manually set to make the trajectory-producing model similar to the true dynamic model in shared states, and $\circ$ denotes the Hadamard product.

C. Forward Reachable Set

Using the trajectory-tracking model, we can define and compute the Forward Reachable Set (FRS) for our robot offline. FRS shows all the positions that are reachable by a robot over a time horizon T. In other words, FRS contains all

the possible trajectories for a robot. At run-time, obstacles are discretized and intersect with the FRS in the parameterized trajectory space to return the safe trajectory parameters set. FRS can be defined as [8]:

$$FRS = \Big\{(x,k) \in X \times K | \ \exists\ x_0 \in X_0, \tau \in [0,T], \quad (5)$$
$$d : [0,T] \to [-1,1]\ s.t. x(\tau) = x, where$$
$$\dot{x}(t) = f(t,x(t),k) + g(t,x(t),k) \circ d(t)$$
$$a.e.\ t \in [0,T]\ and\ x(0)\ =\ x_0 \Big\}$$

FRS is computed offline by solving a linear program. To implement and solve it, we follow the implementation from [8]. We relax the continuous function to polynomial functions and transform the inequality constraints into SOS constraints to change the linear program to a Semi-Defined Program (SDP) [26]. By solving the SDP offline, we get the FRS, and we will use it as a trajectory set online.

IV. METHOD

Having the FRS precomputed offline is not enough for the robot to make successful online plans, so we will next combine the FRS with reinforcement learning to propose a feasible trajectory planning policy π. We first formulate the problem and then introduce our specific design for RL.

A. Problem Formulation

We want the robot to make a successful plan during navigation. First, the robot must ensure collision avoidance and reach the goal successfully. Second, the robot can't stop or advance slowly, which extends the running time a lot. Third, our work focuses on challenging environments, such as a narrow gap between two obstacles. Planning a smooth, feasible trajectory and avoiding re-planning to save running time is what we want.

Computing FRS offline brings us parameterized trajectories used for real-time planning. The RTD plans a set of safe trajectory parameters online. Tracking the trajectory from this set ensures the robot's safety. So we aim to get a policy via RL training that guides the robot to make a successful plan when facing challenging obstacles.

Considering a scenario where a robot with limited sensor horizon, needs to navigate from a start q to a goal g on the plane $\mathbb{R}^2$. The robot had precomputed a FRS that contains all reachable trajectories with some constraints over a time interval [0, T] . At each time-step t, the robot senses obstacles, combines its state to get s_t, then takes action a_t, formulating a cost function. Perform trajectory optimization with this generated cost, to optimize a trajectory parameter $k \in K$. Given $k \in K$ to the controller, it will lead the robot to the next state s_{t+1}. Then the robot gets a reward $R(s_t, a_t)$.

The goal for RL is to learn a policy π_θ that minimizes the agent's time to reach the goal while ensuring safety, which is defined as:

$$\pi_\theta^* = \underset{\pi_\theta}{argmax}\ \ \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R(s_t, \pi_\theta(s_t))\Big] \quad (6)$$
$$s.t.\ \ x(t+1) = f(x(t), u(t)),$$
$$x_T = g,\ \ x_0 = q,$$
$$u(t) \in U,\ s(t) \in S,\ x(t) \in X,$$
$$t \in [0,T]$$

where f is the dynamic model, and U, S, X are the set of admissible control states, states for reinforcement learning, and the robot's own states. We will introduce more details of the reinforcement learning model based on the RTD methods in the remainder of the section.

B. Cost Function Design

When facing challenging obstacles, the robot needs to make a decision and choose a feasible trajectory from the parametered safe trajectory set. The online trajectory optimization algorithm picks a new trajectory parameter k when given a specific cost function, and the robot uses a feedback controller u_k to track k. So cost function design is the key point to guide the robot to make the right decisions.

Figure 2 shows how the cost function design greatly influences the trajectory choice. Past work on RTD has paid little attention to cost function design. They just use the current robot's distance to the sub-goal as the cost function. This cost function tends to choose straight trajectories, so it often performs poorly when the robot needs to pass through a narrow space. Figure 2c shows the robot failing to pass through the narrow space between two obstacles using the previous cost function. But we can design a specific cost function to help the robot make a successful plan, which is shown in Figure 2d. Through the cost function we designed, the robot will plan a smooth and feasible trajectory.

Considering that the real environment is so complex that we can't design a cost function that adapts to all situations, we want the robot to have the ability to adjust the cost function when facing different situations. The cost function J is defined as:

$$J = \sum_{t=0}^{T}[(x(t) - z_x) + (y(t) - z_y)]^2$$
$$+ a * CUR + b * XAA + c * YAA \quad (7)$$
$$CUR = \sum_{t=0}^{T-1} \frac{|v_x(t) * a_y(t) - a_x(t) * v_y(t)|}{(v_x(t)^2 + v_y(t)^2)^{\frac{3}{2}}}$$
$$XAA = \sum_{t=0}^{T-2} |\frac{d(v_x(t))^2}{dt}|$$
$$YAA = \sum_{t=0}^{T-2} |\frac{d(v_y(t))^2}{dt}|$$

The CUR is trajectory's curvature, XAA, and YAA are the jerk in x and y direction. These three terms play different roles in different situations, so we use reinforcement learning to learn a strategy to dynamically adjust the weight parameters a, b, and c.

For example, if the robot is so close to the obstacles that it needs to make a sharp turn, as shown in Figure 2b. In this

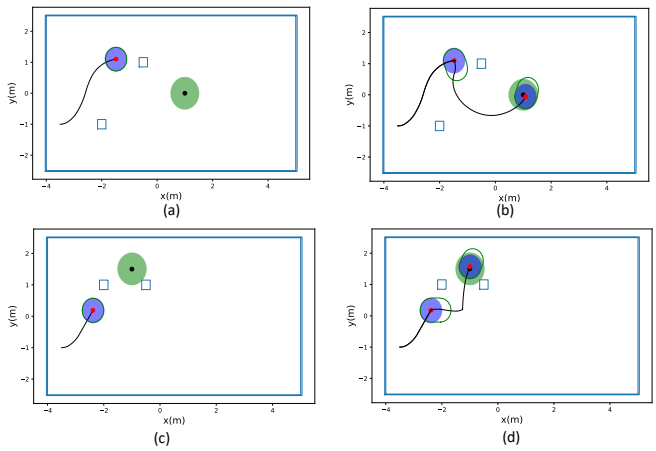

Figure 2: The simulation diagram. Figure 2a and 2c show that the planner used the previous cost function and the robot failed to reach the target point. Figure 2b and 2d show that the planner uses our artificially designed cost function, and the robot reaches the target point successfully.

situation, we want the weight parameters a, b, and c to take a small value to let the robot swerve sharply to avoid collisions. In the situation shown in Figure 2d, it's challenging for the robot to pass through the narrow gap. This time we want CUR, XAA, and YAA to make contributions to planning a feasible trajectory. These three terms make the robot tend to choose a smooth trajectory, which can improve the rate of successful planning.

C. Online Simulation Environment

In this subsection, we will give the specific design of the simulation environment for reinforcement learning.

1) Action Space: The action a_t defines the weight of three cost items described in IV-B. Before training the model, we manually narrowed the scope of the parameters.

2) State Space: The state $s_t = (x_t, o_t)$ at time t has two components. The first is the observable robot own state x_t at the time step t and the second component is the states of obstacles that are closest to the robot currently. We set the relative position of the two obstacles closest to the agent on the forward horizon as o_t.

3) Goal: Although the RTD online planning ensures safety guarantees and we set replan methods to reach the goal in actual running, we treat replan as a failure in the simulation. Navigation from the start point to the goal without stopping, replaning, and collision seems like a successful plan in our simulation environment.

D. Reward Function

The reward function is designed to encourage the agent to avoid collisions and take less time to reach the goal:

$$R(s,a) = \begin{cases} r_{goal} & if\ x_t = g \\ r_{collision} & if\ d_t < d_{min}\ or\ v_t < v_{min} \\ r_t & otherwise \end{cases} \quad (8)$$

where $d_t = \min_i \left\| x - x_{obs}^i \right\|$ is the robot's current distance to the closest obstacle at the time-step t. r_{goal} is set to reward the agent if it reaches the goal, and $r_{collision}$ penalizes it if the agent collides with obstacles. Sometimes the robot fails to pass through the narrow gap, tries to turn around, and bypasses the obstacles. We also treat it as a collision and use $r_{collision}$ to penalize it. At other time-step, $r_t = -0.1$ is designed as a time reward to penalize the agent until it reaches the goal. Spending less time reaching the goal while avoiding collision is what we aim to do.

E. Learning Model

We use a state-of-the-art method, Proximal Policy Optimization (PPO) [27] with clipped gradients to train our policy, and we combine the PPO training with the RTD online planning. Algorithm 1 shows the specific training strategy.

Algorithm 1: PPO Training with RTD Planning

Inputs :initial policy π_θ, trajectory sets K, cost function J, reward function $R(\mathbf{s}_t, \mathbf{a}_t)$, dynamic model f, RL training episodes N_{ep};
while $episode < N_{ep}$ **do**
  for *k=0, ... ,N* **do**
    initialize robot state $\mathbf{x}_0$, goal position $\mathbf{g}$, t=0;
    subgoal $\{\mathbf{g}_0, ..., \mathbf{g}_n\} \leftarrow UpPlanner(\mathbf{x}_0, \mathbf{g})$;
    while *not done* **do**
      sense obstacles $\{X_{obs,i}\}_{i=1}^{n_{obs}}$;
      compute $\mathbf{o}_t$ from X_{obs};
      $\mathbf{s}_t \leftarrow \{\mathbf{x}_t, \mathbf{o}_t\}$;
      run policy π_θ, get $\mathbf{a}_t$;
      compute $J \leftarrow PrecomputeCost(\mathbf{a}_t)$;
      compute safe trajectories set K_{safe} ;
      compute $\mathbf{u}_t \leftarrow Optk(J, K_{safe})$;
      $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$;
      collect $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t\}$;
      $t \leftarrow t + 1$;
    end
  end
  PPO training;
end
return π_θ

After the robot's initial and target position are given, the $UpPlanner$ function will give a series of subgoals $\{g_0, ..., g_n\}$ using a path planning algorithm extend form the A-star algorithm. If the robot has not reached the goal, the algorithm senses surrounding obstacles. We use the same method in [12, Section 6] to represent the simulation obstacles as a finite, discrete set, to get the safe trajectories set $K_{safe} \in K$. $PrecomputeCost$ function is used to

generate the specific cost function based on the action at the current time-step. Finally, the optimization $Optk$ is defined as:

$$\min_{k} J(k) \tag{9}$$
$$s.t.\ k \in K_{safe}$$

Finite constraints and low-dimension variable k allow $Optk$ to be used in online planning. We get the control input u_t by solving the optimization using the interior point method.

V. RESULTS

We use a different-drive robot named Segway. We train the policy using PPO in simulation and apply the trained policy to the Robot to indicate that our planner can optimize feasible and safe trajectory in unstructured, complex environments without collision. We also present a comparison of our method to RRT, NMPC, and RTD.

A. Experiment Setup

The Segway's Parameters are summarized in the Table II. It runs in a $9 \times 5\ m^2$ room in the simulation training. The room is filled with 7 randomly-generated boxes to simulate crowded environments. The length of each box is $0.3\ m$. The starting position is randomly set on the left side of the simulation room, the goal position is randomly set on the right side.

Our RL algorithm implementation is based on the open-source PPO algorithm provided in the Stable-Baselines.

We first study some hyperparameters' influence on policy performance, and the results are shown in Figure 3. The curves have been smoothed. Each experiment only changes one hyperparameter compared with the hyperparameters summarized in Table IV. The final hyperparameter values we used are summarized in Table IV.

TABLE I

SEGWAY PARAMETERS

Footprint radius	$0.83m$
Maximum yaw rate	$\omega = \pm 1\ rad/s$
Acceleration	$[\gamma, \bar{\gamma}] = [-5.5, +5.5]\ rad/s^2$
Angular acceleration	$[\alpha, \bar{\alpha}] = [-3.50, +3.50]\ m/s^2$

TABLE II

HYPER-PARAMETERS

Batch size	64	Num. mini batches	2048
Num. epochs	10	γ	0.99
Clip factor	0.1	Learning rate	$2 * 10^{-4}$
r_{goal}	5	$r_{collision}$	-9

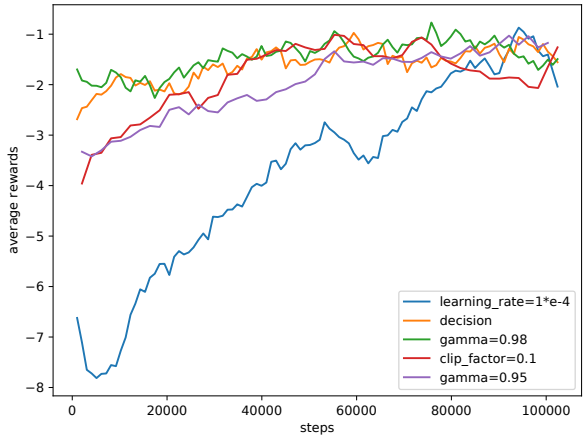

Figure 3: The ablation experiment

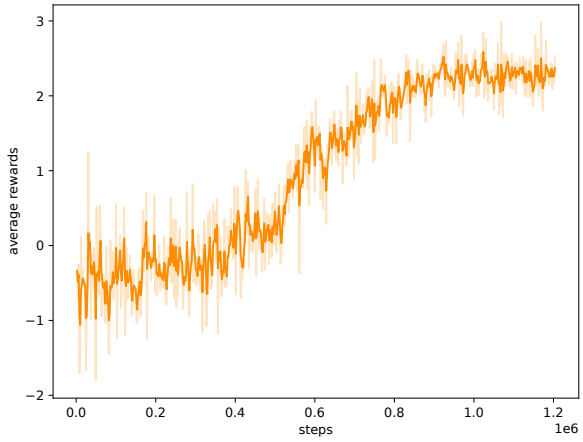

Figure 4: Moving average rewards during training

B. Simulation Results

Figure 4 shows the robot's average reward curve during long-term training.

We generate 1000 random trials for each method to show the advantages of our approach in complex environments. The results are summarized in Table VI and our method outperforms each of the compared methods in random, complex environments. Since the environments are randomly generated, there may be no feasible paths from the start to the goal. We also manually design some narrow passages where feasible passing paths exist to test whether our method outperforms baselines. The results in Table VIII shows our method improves the robot's ability to pass through narrow passages. The We also design experiments to show that using our method spends less time reaching the goal. Under the same environment, both the RTD and our method successfully reach the goal, the average time we calculated from 100 experiments shows that our method saves 14.6 % of the time.

TABLE III

RANDOM COMPLEX ENVIRONMENT

Planner	success rate
RRT	61.3%
NMPC	66.7%
RTD	75.2%
RTD with manually designed cost	79.5%
RTD trained with DRL	**85.4%**

TABLE IV

PASSING NARROW PASSAGE

Planner	success rate
RRT	68.0%
NMPC	72.0%
RTD	87.0%
RTD with manually designed cost	93.0%
RTD trained with DRL	**98.0%**

VI. CONCLUSION AND FUTURE WORK

We propose a reachability-based trajectory planner trained with DRL for crowded environments and passing narrow passages, which generates safe and feasible trajectories. Offline training using deep reinforcement learning improves the ability of mobile robots to pass through narrow passages.

The current work is mainly focused on mobile robots passing through narrow passages. Next, we will extend our method to some other complex environments, like uneven terrain. We also want our planner to be more intelligent for different situations to generate different cost functions.

REFERENCES

[1] R. Geraerts and M. H. Overmars, "Reachability-based analysis for probabilistic roadmap planners," *Robotics and Autonomous Systems*, vol. 55, no. 11, pp. 824–836, 2007.

[2] J. Chen and H. Qiao, "Motor-cortex-like recurrent neural network and multi-tasks learning for the control of musculoskeletal systems," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 14, no. 2, pp. 424–436, 2020.

[3] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[5] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 285–292, 2017.

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

[7] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6252–6259, 2018.

[8] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1419–1469, 2020.

[9] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[12] J. Wurts, J. L. Stein, and T. Ersal, "Collision imminent steering using nonlinear model predictive control," *2018 Annual American Control Conference (ACC)*, pp. 4772–4777, 2018.

[13] D. Hoeller, F. Farshidian, and M. Hutter, "Deep value model predictive control," *Conference on Robot Learning*, pp. 990–1004, 2020.

[14] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: a comparison on a power system problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 517–529, 2008.

[15] B. Brito, M. Everett, J. P. How, and J. Alonso-Mora, "Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4616–4623, 2021.

[16] K. Margellos and J. Lygeros, "Hamilton–jacobi formulation for reach–avoid differential games," *IEEE Transactions on automatic control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[17] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on automatic control*, vol. 50, no. 7, pp. 947–957, 2005.

[18] J. Ding, E. Li, H. Huang, and C. J. Tomlin, "Reachability-based synthesis of feedback policies for motion planning under bounded disturbances," *2011 IEEE International Conference on Robotics and Automation*, pp. 2160–2165, 2011.

[19] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin, "An efficient reachability-based framework for provably safe autonomous navigation in unknown environments," *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 1758–1765, 2019.

[20] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," *Dynamic Systems and Control Conference*, vol. 58271, p. V001T44A005, 2017.

[21] S. Vaskov, S. Kousik, H. Larson, F. Bu, J. Ward, S. Worrall, M. Johnson-Roberson, and R. Vasudevan, "Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments," *arXiv preprint arXiv:1902.02851*, 2019.

[22] P. Holmes, S. Kousik, B. Zhang, D. Raz, C. Barbalata, M. Johnson-Roberson, and R. Vasudevan, "Reachable sets for safe, real-time manipulator trajectory design," *arXiv preprint arXiv:2002.01591*, 2020.

[23] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, aggressive quadrotor flight via reachability-based trajectory design," *Dynamic Systems and Control Conference*, vol. 59162, p. V003T19A010, 2019.

[24] Y. S. Shao, C. Chen, S. Kousik, and R. Vasudevan, "Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3663–3670, 2021.

[25] J. B. Lasserre, "Convergent sdp-relaxations in polynomial optimization with sparsity," *SIAM Journal on Optimization*, vol. 17, no. 3, pp. 822–843, 2006.

[26] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical programming*, vol. 96, no. 2, pp. 293–320, 2003.

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