Unguided Self-exploration in Narrow Spaces with Safety Region
Enhanced Reinforcement Learning for Ackermann-steering Robots

Zhao feng Tian¹, Zichuan Liu², Xingyu Zhou³ and Weisong Shi³

Abstract—In narrow spaces, motion planning based on the traditional hierarchical autonomous system could cause collisions due to mapping, localization, and control noises, especially for car-like Ackermann-steering robots which suffer from non-convex and non-holonomic kinematics. To tackle these problems, we leverage deep reinforcement learning which is verified to be effective in self-decision-making, to self-explore in narrow spaces without a given map and destination while avoiding collisions. Specifically, based on our Ackermann-steering rectangular-shaped ZebraT robot and its Gazebo simulator, we propose the rectangular safety region to represent states and detect collisions for rectangular-shaped robots, and a carefully crafted reward function for reinforcement learning that does not require the waypoint guidance. For validation, the robot was first trained in a simulated narrow track. Then, the well-trained model was transferred to other simulation tracks and could outperform other traditional methods including classical and learning methods. Finally, the trained model is demonstrated in the real world with our ZebraT robot. https://sites.google.com/view/z14exploration

I. INTRODUCTION

Autonomous driving and robotics technologies have undergone significant advancements in recent times. A notable emerging application of these technologies is in hazardous environment operations, where robots can be utilized to perform essential tasks without human intervention. Examples of such applications include underground mining, tunnel/pipeline inspection, and post-disaster search [1], [2]. In these scenarios, in addition to the inherent risk to human life, a key challenge is the need for autonomous robots to self-explore narrow spaces while avoiding collisions. This task is particularly arduous for car-like Ackermann-steering robots for two reasons. First, such robots are subject to non-convex kinematic constraints [3], e.g., the robot cannot drive sideways or turn in place, which limits its maneuverability. Second, these robots typically have a rectangular shape, introducing challenges to define free space for collision avoidance in geometric methods [4].

One straightforward approach to the aforementioned application is to adopt a traditional hierarchical methodology, which involves a sequence of tasks such as sensing, perception, planning, and control. Nevertheless, this method faces several significant limitations that render it insufficient for the current tasks. First, planning and control necessitate a premade map, which is unavailable in the self-exploration context. Second, even with a premade map, obstacle parameterization, and non-convex obstacle convexification can pose significant challenges. In general, obstacle convexification based on the decomposition and dilation of the original premade map can limit the accessible areas and result in planning failures, particularly in narrow spaces where the obstacle is non-convex and curvy. Third, the dynamic uncertainty introduced by stacked mapping, localization, and control errors cannot be overlooked.

Reinforcement Learning (RL), particularly Deep Reinforcement Learning (DRL) – which employs deep neural networks for function approximations or representations – has recently achieved significant success across a diverse range of applications. These include Atari video games [5], Alpha Go [6], and real-world scenarios like robot manipulators [7], navigation [8], and transportation [9]. The fundamental concept behind RL involves sequential decision-making in unknown environments where the agent learns to act by interacting with the environment through trial-and-error to maximize the long-term reward. Furthermore, the use of deep neural networks in DRL allows the agent to handle large-scale states and learn the optimal policy directly from raw inputs without hand-engineered features or domain heuristics. To harness the potential of DRL, various learning algorithms have been developed in recent years, including Deep Q-Network (DQN) for discrete action space and Deep Deterministic Policy Gradient (DDPG) [10], Soft Actor-Critic (SAC) [11], and Proximal Policy Optimization (PPO) [12] for continuous action space.

In this paper, we present the first step towards applying Deep Reinforcement Learning (DRL) to address the problem of self-exploration in narrow spaces without using pre-made maps of the environments. Specifically, we focus on the scenario where a rectangular-shaped Ackermann-steering robot aims to efficiently explore general narrow spaces without any guidance from waypoints or destinations, while avoiding collisions. By narrow spaces, we refer to non-branching passages that constrain an Ackermann-steering robot from turning around for 180 degrees without backing up or the space that is narrower than the circumscribed circle of the robot body when viewed from above. Two unique challenges arise in our case: (i) detecting collisions in narrow spaces is a challenging task when the robot is rectangular-shaped and uses a Lidar or laser scanner to sense the environment. The standard approach of setting a scan range value to the cover the robot body can lead to an over-coverage problem, which, in turn, can result in inefficient exploration; (ii) as
with many DRL applications, designing a suitable reward function for the agent is a critical challenge. In our scenario, motivating the car-like robot to explore without guidance from a destination or waypoint and designing a steering and reversing mechanism for its non-convex kinematics to avoid collisions are not addressed in the previous traditional waypoint-guided DRL reward function for differential robots. This requirement necessitates more delicate reward shaping to balance various working conditions.

Our contributions. We make two main contributions to address the challenges outlined above. First, we introduce a rectangular safety region that is tailored for rectangular-shaped robots to accurately represent robot states and enable precise collision detection. Second, we carefully design a reward function that balances exploration and collision avoidance while not relying on waypoint guidance for narrow spaces. To systematically evaluate our proposed methods, we conduct a four-step procedure as follows:

(i) Collision tests: We compare our proposed safety region representation for rectangular-shaped robots with other laser-scan-based representations for collision detection. The results show that our safety region can detect collisions more accurately with the same number of discretized laser scans.

(ii) Training in simulation: We incorporate our proposed reward function into five mainstream DRL algorithms - DDPG, DQN, PPO, PPO-discrete, and SAC - and benchmark their performance in a simulated narrow track. Using the selected DDPG algorithm, we then train multiple models with other reward functions for contrastive and ablation evaluation studies. We use multi-seeds for each algorithm and model, and each seed is episode-fixed for fairness.

(iii) Sim-to-sim evaluations: We transfer the different trained models from step (ii) to seen and unseen scenarios and evaluate their performance by success rate, fail rate, collision rate, and time cost. We compare the proposed reward function with a traditional waypoint-guided reward function (contrastive model) and models with different reward components (ablation models). We also benchmark other classical map-based geometry methods and the Imitation Learning (IL) method. The model using the proposed reward function demonstrates a convincing generalization ability and outperforms other methods in most scenarios.

(iv) Sim-to-real demonstration: We transfer the learned policy from step (ii) directly to our real-world ZebraT robot and test it on three real-world narrow tracks. The trained model with the proposed reward function successfully completes all three tracks without collisions. This not only validates the effectiveness of our proposed methods but also demonstrates the generalization power of the learned policy.

II. RELATED WORK

A. Classical Motion Planning

Traditional map-based methods of motion planning have dominated the area for a long time, which are built upon mapping, global planning, and local planning workflow. Global planning primarily refers to pathfinding problems, where A* [13] and Dijkstra [14] algorithms are two classical methods used in Robot Operating System (ROS) [15] navigation stack. Whereas local planning pays more attention to collision avoidance and kinodynamic feasibility. Dynamic Window Approach (DWA) [16] and Timed Elastic Band (TEB) [17] planners are two local motion planners integrated with ROS and have been widely used in the community, where DWA is suitable for differential drive robots, and TEB planner is able to consider the kinematic constraints of a car-like robot. And some motion planning methods aim to solve collision avoidance problems in a dynamic environment, e.g., Velocity Obstacle (VO) method [18], and its derivatives Reciprocal Velocity Obstacle (RVO) [19], Optimal Reciprocal Collision Avoidance (ORCA) methods [20].

B. DRL Motion Planning

A navigation problem for a mobile robot is usually defined as how to move from point A to point B. Machine learning-based motion methods are promising to solve navigation problems in an unknown environment. [21] practices end-to-end navigation through IL, where the model is trained with human actions in a supervised paradigm. However, this supervised learning method is highly constrained by human policy and not guaranteed when validating set changed [8]. Plenty of research [22], [23], [24], [25], [26] has demonstrated DRL is suitable to generate motions for robot navigation problems, where the difference between the current position and the destination is taken as a reward to facilitate the robot approaching the destination. Meanwhile, research [27], [28], [29] studies obstacle avoidance among dynamic obstacles using DRL methods. However, all these studies in subsections A and B, do not scope in narrow spaces and the robot models are round or near-round-shaped and use differential driving. While for a rectangular Ackermann-steering robot, the fixed range detection and the policy trained in open spaces would be insufficient. The other challenge for the Ackermann-steering robot is its non-convex action space, hence robot may need to back up for heading angle adjustments, so it needs to explore actions with negative linear velocities whereas a differential robot does not, which induces more uncertainties for learning convergency.

C. Narrow Space Motion Problem

Moving a “piano” in a narrow space without collision is a challenging task [30]. A collision-free path sometimes is unachievable under some circumstances when using obstacle dilation on a map. Research [31] used a Lyapunov-based control strategy to move a car-like robot in a narrow lane. However, this method costs daunting computations and requires a parameterized environment that is not available in exploration scenarios.

D. Innovation of This Study

A comparison between related works with this study is listed in Table I. This study considers narrow space and
Then by using the value iteration equation $Q_{t+1}(s_t, a_t) = \mathbb{E}_{s_{t+1}|s_t, a_t} [r_t + \gamma \max_a Q_{t+1}(s_{t+1}, a_{t+1})]$, the $Q_t$ value can converge to $Q^*$ as time step $t \to \infty$. In the practice of DQN, the $Q_t$ value can be predicted by a deep neural network parameterized by $\theta$, and the network is trained with the loss function $L(\theta_t) = \mathbb{E}_{s_t, a_t} [(y_t - Q(s_t, a_t, \theta_t))^2]$, where $y_t = \mathbb{E}_{s_{t+1}|s_t} [r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}, \theta_{t+1})]$ is the target for the iteration at $t$.

B. Deep Deterministic Policy Gradient (DDPG)

Derived from the actor-critic model [38] and policy gradient method [39], DDPG not only estimates the $Q$ value with a critic network parameterized by $\theta^\pi$ but also estimates optimal actions with an actor network parameterized by $\theta^\pi$. Wherein the critic network is updated with a method similar to DQN that uses a loss function, while the actor network is updated with a policy gradient:

$$\nabla_{\theta^\pi} J \approx \mathbb{E}[\nabla_{\theta^\pi} Q(s, a|\theta^\pi)|s=s_t, a=\pi(s_t)|\theta^\pi] = \mathbb{E}[\nabla_a Q(s, a|\theta^\pi)|s=s_t, a=\pi(s_t)] \nabla_{\theta^\pi} \pi(s|\theta^\pi)|s=s_t]$$

where $J = \mathbb{E}_{s, a, \theta} [G_t]$ is the expected return from the beginning state which is subject to the start distribution, $Q(s, a|\theta^\pi)$ and $a = \pi(s|\theta^\pi)$ respectively denote the action value w.r.t. the critic network and the policy function w.r.t. the actor network. Framed in such an actor-critic structure, DDPG is suitable to compute an action for continuous control problems with high efficiency.

### III. PROBLEM FORMULATION AND BACKGROUND

We model the narrow space self-exploration problem as a Markov Decision Process (MDP). In an MDP, an agent (i.e., the robot) interacts with the environment through a sequence of states, actions, and rewards. At each time step $t \in [0, T]$, the agent observes the current state $s_t$, executes an action $a_t$, obtains a reward $r_t$, and transitions to next state $s_{t+1}$. The goal is to maximize the cumulated reward from step $t$ onward, which is denoted by $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, where $\gamma$ is a factor to discount the future reward. In this work, we benchmark several DRL algorithms to tackle the MDP problem, wherein DQN and DDPG are two typical algorithms that respectively work on discrete and continuous action space.

A. Deep Q Networks (DQN)

Combining the original Q-learning with deep neural networks, DQN approximates an action-value function in a Q-learning framework with a neural network. The optimal action-value function $Q^*(s_t, a_t) = \max_a Q^*(s_t, a_t)$ is defined as the maximum expected return w.r.t. the state $s_t$ and the action $a_t$, where $\pi$ is a policy that maps states to actions. The optimal action-value function is subject to the Bellman equation so that it can be written as equation (1).

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}|s_t, a_t} [r_t + \gamma \max_a Q^*(s_{t+1}, a)|s_t, a_t] \quad (1)$$

### IV. METHODS

In this section, we present our main methods to apply DRL to solve the problem of narrow space self-exploration. To this end, we first develop a novel state representation method tailored for rectangular-shaped robots, which in turn helps to detect collisions in narrow spaces. Further, we carefully validated with a rectangular-shaped Ackermann-steering car-like robot with 3D Lidar mounted, which is different from previous studies that are based on open space and smaller round-shaped differential robots. Additionally, this study uses a tunable safe region to detect a collision instead of a fixed range value. Moreover, no map and destination information is needed in this study, and backing up is allowed to avoid the obstacles straight ahead.
design a reward function with various components to trade off specific task requirements.

A. State Representation with Rectangular Safety Region

As mentioned in the “piano mover” paper [30], it is a complex problem to figure out whether the “piano” will collide with a curved wall or not. In most previous research, the robot is round-shaped, the state of which is represented by discretized laser scans with a fixed angle interval [23], [8]. Consequently, collision detection can be implemented simply with a fixed range value. However, for a rectangular-shaped robot, this method could cause over-coverage problems shown in Fig. 1. Since the Lidar is not usually mounted at the robot center, such that using a fixed range to cover the robot may generate a larger area that can be detected as collisions by mistakes, which subsequently makes more spaces inaccessible for the robot and detriments narrow space performance.

To detect collisions more accurately, as expressed in Algorithm 1, we propose a rectangular-shaped safety region to represent the robot state. Also shown in Fig. 1(b), the laser scans now are not discretized by a fixed interval anymore, instead, the angle intervals become adaptive to the robot’s rectangular-shaped body so that the discretized scans could evenly cover the four sides of the rectangular safety region (yellow rectangular) with a resolution value \( r \), which means for every side, the discretized scans could fall into it with an interval distance \( r \). Besides, the scans in the directions of some important axis and diagonals (bold red lines) are selected, which split the rectangular safety region into 8 phases. Then in every phase, the indices \( V_{\text{index}} \) and the collision ranges \( V_{\text{range}} \) of the discretized scans can be calculated to detect collisions. And further, we use the ranges (the distance or length of each ray) of the discretized scans \( V_{\text{obs}} \) to represent the robot state.

B. Reward Function Shaping

One key that distinguishes our reward function is that we do not take the destination into account because, in our considered scenario, a car-like robot needs to explore a narrow space without a certain destination while obeying its Ackermann-steering constraints. In contrast, the destination information in the previous papers is taken into reward function shaping, which could help the model convergence and training efficiency. Hence how we urge the robot to move forward towards an “open” direction without a destination, instead of dawdling or going back, becomes a question. In this case, we craft a reward function comprises of four components, \( R_f, R_o, R_m, \) and \( R_t \), which respectively award moving forward towards an open direction, distance to obstacles, keeping in middle, saving time (FOMT). The complete expression is presented by equation (3).

\[
R = \begin{cases} 
R_c & \text{if collision} \\
R_t & \text{if reach open space} \\
c_1R_f + c_2R_o + c_3R_m + c_4R_t & \text{otherwise} 
\end{cases}
\]  

(3)

Where we have a negative reward for collision, and also a positive reward for reaching the open space. Notice that we consider if the robot reaches the open space (set as the end of the narrow space), an episode is done. To detect the open space is simple, once the sum of two scans, in the straight right and left direction, becomes bigger than a threshold, we consider the robot has reached the open space - the end of the narrow space. Otherwise, in other general steps within an episode, the reward function comprises mentioned four components, \( R_f, R_o, R_m, \) and \( R_t \), named by FOMT as well.
\[
R_f = \sum_{k=0}^{n_f} \alpha_2^k \times v \times (V_{obs}[f + k] + V_{obs}[f - k])
\]

The moving-forward reward component \(R_f\) defined above aims to control the heading direction and urge the robot to move forward, where the heading control is implemented by awarding the most open direction, which means the obstacle along this direction is none or far away by laser range value. Specifically, \(V_{obs}[f]\) denotes the observation of the ahead direction (on the mid-line of the robot), and \(V_{obs}[f \pm k]\) denotes its \(n_f\) neighboring observations, the reward of which will be larger with larger range values (longer distance to the obstacle), and function for mentioned heading control. While the linear speed \(v\) can urge the robot to move forward because without it the robot can stop or back up to keep a larger distance to the obstacle ahead to obtain a higher reward from \(V_{obs}\). Correspondingly, \(v\) can be 0 or negative to punish the stop or back up, hereby the robot would be encouraged to move forward with a positive \(v\) to strike a higher score. Besides, discount factor \(\alpha_2\) is to make the observation straight ahead more prominent than other neighboring tilted observations.

\[
G = \{V_{obs}[i] - V_{range}[i] | i \in \mathbb{N}\}_{ascending}
\]

\[
R_o = \sum_{k=0}^{n_o} \alpha_3^k \times \log G[k]
\]

In equation (5), set \(G\) denotes a vector that contains the gap value between each observation \(V_{obs}[i]\) and its corresponding safe range \(V_{range}[i]\) in ascending order. In equation (6), the obstacle-distance reward component \(R_o\) uses a log function to punish the close distance using the sorted gap vector \(G\), wherein a discount factor \(\alpha_3\) is used to make the impact of closer distance more prominent.

\[
R_m = -\sum_{k=0}^{n_m} \alpha_3^k \times |V_{obs}[r - k] - V_{obs}[l + k]|
\]

The keeping-in-middle reward component \(R_m\), defined in equation (7), would help keep the robot in the middle of the obstacles around the robot’s two sides. The difference between the observation in the straightly right direction \(V_{obs}[r]\) and the observation in the left direction \(V_{obs}[l]\), as well as the difference between their \(n_m\) neighboring symmetric observations with respect to the robot’s symmetry axis, are leveraged to evaluate the robot’s performance of keeping in the middle. Where \(\alpha_3\) serves as a factor to discount the reward from the neighboring observations.

\[
R_t = \alpha_4
\]

The last time-saving component \(R_t\) is quite simple, where \(\alpha_4\) is a negative constant to punish the time passing.

| Parameter | Value |
|-----------|-------|
| \(n_{scan}\) | 32    |
| \(r\)      | 0.095(m) |
| \(R_c\)    | -50   |
| \(R_t\)    | 50    |
| \(c_{1-4}\) | 1     |

C. Reinforcement Learning Workflow

The training process is operated under an OpenAI gym framework, where in each step, the robot with a current state \(s_t\) outputs an action \(a_t\), and waits for a step time interval to get new state \(s_{t+1}\) and a reward \(r_t\). Until “done”, the number of steps will keep growing in an episode. In our settings, each time step last for 0.2s, which means 5 actions will be generated per second. As Fig. 2 shows, the DDPG method is given as an example to explain the workflow within a training episode. Wherein the input state \(S_t\) is comprised of the action pair of the linear speed \(v \in [-0.6, 0.6](m/s)\) and the steering angle \(w \in [-0.6, 0.6](rad)\), i.e., \((v, w)\), and the observation vector \(V_{obs}\), which is derived from our proposed rectangular safety region and contains the laser range observation \(O_1\) to \(O_8\) \((0, 6)(m)\). The actor and critic networks are both composed of two fully linked layers of size 512, and output respectively the action pair \(a_t\) and the \(Q\) value. Then the robot in the simulation conducts \(a_t\) and obtains a new state \(S_{t+1}\) and a done flag, which indicates whether the episode is ended by a collision or reaching the open space. Furthermore, they will be fed back to the networks and will update their parameters. While for discrete action settings (DQN), action pairs are formulated by \((v, w) \in \{-0.6, 0.6\} \times \{-0.6, 0.6\}\).
V. EXPERIMENTS AND EVALUATIONS

Our experiments are conducted using the ZebraT platform [40], a rectangular-shaped robot with Lidar sensors, and a ROS/Gazebo simulator [40] tailored for this robot. We conduct experiments in a simulated narrow “big track” in the Gazebo simulator, which is a combination of Track 1, 2, and 3, as shown at the top of Figure 5. The “big track” contains 45, 90, and 180-degree corners, narrow corridors with a width of fewer than 2 meters, and the narrowest section is only around 1 meter wide, providing a rich training environment. The parameters used in the experiments are presented in Table II. The contrastive and ablation study sets used in subsections B and C are explained in Table III.

### TABLE III

| Set | Note  | Set | Note  |
|-----|-------|-----|-------|
| DRA | map-based | TEB | map-based |
| IL  | imitation learning | RLIC | waypoint-guided RL |
| RLFT | RL with FT | RLWG | RL with FOMT |
| RLWOGT | RL with FOMT |

Fig. 3. State representation comparison. (a) FIFR, fixed interval fixed range. (b) FIRect, fixed interval but different range to fit the rectangular contour. (c) Comparison by the number of detected collisions.

Fig. 4. Training curve, the best score among every 20 episodes to formulate the training curve. Where the solid line denotes the average value of the best score of 5 random seeds and the shaded area denotes the trust region among random seeds.

A. State Representation Comparison

Collision detection is a critical aspect of operating in narrow spaces, as missing detections could lead to collisions and result in inappropriate learning by the DRL model. To validate our Safe Region (SR) state representation, we set up two baselines. Baseline FIFR uses Fixed Interval and Fixed Range (FIFR) to discretize laser scans, while Baseline FIRect uses Fixed Interval Rectangular-fitting (FIRect) ranges. FIFR is a popular representation paradigm used in previous works for round/near-round-shaped robots and serves as a contrastive set. FIRect serves as an ablation set to validate the interval discretization in our SR method. In Fig. 3(a, b), we present a visualization of FIFR and FIRect. To avoid over-coverage, we set the range value in FIFR to only cover the robot width. By operating robots in the “big track” and conducting random collisions, we benchmarked the three paradigms by the number of detected collisions, as shown in Fig. 3(c). Our SR could detect most collisions, while FIRect and FIFR could detect 19.88% and 28.54% fewer collisions, respectively. This validates our SR state representation method.

B. Training and Baseline Settings

To select an appropriate DRL algorithm, five algorithms including DDPG, DQN, SAC, PPO, and PPO-discrete are benchmarked with our proposed state representation and reward function in the training process in the “big track”. And the training curve is shown in Fig. 4, where each algorithm is trained with five random seeds in 1000 episodes with memory buffer size at $2 \times 10^5$. The buffer size is designed based on an average number of sequences recorded in 1000 episodes, which varies from 3 to $10 \times 10^5$ based on the total steps (a robot with better training performance would navigate longer, which takes more steps in an episode). DDPG could get the highest return and complete the track with learned back-up collision-avoiding skills from our observation, thereby we select DDPG as the DRL algorithm in the subsequent experiments. And since the training rewards are the only factors to motivate robots for their training instead of final evaluation metrics, e.g., success rate and collision rate, to decrease noise and illustrate the training tendency, we pick the best score among every 20 episodes to formulate the training curve.

$$R = \begin{cases} R_c & \text{if collision} \\ R_g & \text{if reach goal} \\ c_5(dist_{t-1} - dist_t) & \text{otherwise} \end{cases}$$ (9)

More importantly, we set up contrastive study and ablation study to benchmark reward function shaping methods. In contrastive set RLWG, a traditional waypoint-guided reward function [8], [23], [29] is defined in (9) where the $R_c$ and $R_g$ are collision and goal rewards, which are set as -50 and 50 in the same scale as our reward function. In general steps, the distance difference to the waypoint formulates the reward with a coefficient $c_5$, which is set as 100 for scaling to the range of our reward function, and the waypoints are set manually along the tracks. Also, the relative distance between the robot pose and the goal pose, and the yaw angle difference are taken as additional 2 elements in our original safety region state vector of 32 laser range elements.
For ablation study, our set RL\textsubscript{FOMT} denotes that four components $R_f, R_o, R_m, R_t$ formulates our unguided reward function, whereas the ablation set RL\textsubscript{FT} and RL\textsubscript{FOT} are formulated respectively without $R_o, R_m$ and without $R_m$. The reason we do not ablate F or T component is that without F the robot will be not motivated to move, and T is just a general component that functions as a constant value. Moreover, the classical method sets DWA, TEB, and imitation learning set IL are formulated as well, and all the sets can be looked up in Table III. Specifically, to operate DWA and TEB, a premade map is provided, and local and global dilation radius is tuned with both 0.3 and 0.5 to obtain the best performance, and the robot wheelbase and steering radius are set up for TEB. Whereby the robot tends to encounter fewer planning failures but more collision risks with a 0.3 setting and tends to fail or oscillate with a 0.5 setting since the accessible area shrinks. While distance keeping ahead is set to 0.3 for TEB because a larger value could frequently cause failures in narrow space. In the IL training phase, a human expert operates the robot to drive in the “big track” for around 1 hour with 10000 states (laser range vectors) and actions recorded. Then a network with two 512 full-linked layers (similar to RL networks for fairness) is trained with state inputs with its action labels in a supervised paradigm. Besides, for each RL study set, training is conducted using 5 random seeds, 1000 training episodes with extra 500 episodes of best-model fine-tuning, and memory buffer size $2 \times 10^5$, while DDPG actor and critic-network learning rate at $1 \times 10^{-4}$ and $2 \times 10^{-4}$.

C. Sim-to-sim Evaluation

We conduct extensive evaluation experiments in 8 simulated narrow tracks, where Track 1-3 is from the training track, the “big track”, and others are unseen tracks. Study sets listed in Table III is compared by the metrics including success, fail, collision rate, and average time cost of success. For learning sets including IL, RL\textsubscript{WG}, RL\textsubscript{FT}, RL\textsubscript{FOT}, RL\textsubscript{FOMT}, all the best models are transferred to new tracks for evaluation and run 70 episodes. And for map-based sets, DWA and TEB, each setting of dilation radius 0.3 and 0.5 is respectively tested for 15 episodes, and only the well-performing setting (by success rate) is presented in the final result. Quantitive evaluation w.r.t. each method and each track are shown in Fig. 5(a), where our method RL\textsubscript{FOMT} strikes the highest success rate in most tracks with a lower time cost.

Observations and analysis. i) In classical map-based sets DWA and TEB, planning failures often happen in 90-degree turns, where the robot either oscillates or stays in place. Generally, TEB could outperform DWA since the algorithm per se considers car-like robot kinematics. While TEB enables some backing-up recovery actions but tends to reverse the robot’s heading in some planning cases, which costs more time and is perceived as aberrant. ii) For IL set, the model training requires high-quality demonstrations
and time investments from human operators. We perceive significant difficulties for the IL model to be both validated in the seen scenario, i.e., “big track” and generalized to new scenarios. iii) For RLWG set using the traditional waypoint guided reward function, we perceive the method needs dense waypoints for training and the model could hardly master the 90-degree turning skills. We consider this could be induced by its “collision-sparse” reward function. Although it awards general steps and is not the traditional “sparse reward function” that only grades rewards for success or failure, we consider it grade collision in a sparse way so that only robot collides could make it learn the discrete collision boundary. Whereas in our FOMT reward function, the O component would grade on the distance to obstacles, which makes “collision-sparse” to “collision-gradient”. Also, the ablation set RLFT is also considered “collision-sparse”, hereby their collision rate is comparatively higher in Fig. 5(a). iv) Among the ablation sets RLFOMT, RLFT, and RLFOT, RLFT has the highest fail and collision rate shown in Fig. 5(b). By adding the O component, RLFOT achieves better collision avoidance performance than RLFT. And we perceive O component helps the robot to learn a backing-up skill to complete a 90-degree turn and thus RLFOT, and RLFOMT have higher success rates in Track 7 which contains continuous turns. However, RLFT and RLFOT both fail in Track 4 and 8 where the track is uneven or curvy, this unseen features impact their performance. While thanks to the M component, RLFOMT could motivate the robot to move in the middle of the side obstacles regardless of their features, hereby getting better performance. In summary, O helps avoid a near collision and learn a continuous collision boundary, and M helps navigate among uneven or curvy obstacles in narrow spaces.

D. Sim-to-real Demonstration

Trained in the “big track”, the well-performed RLFOMT model then is transferred to real-world narrow tracks shown in Fig. 6. As a result, our ZebraT robot could complete all three tracks without any collisions. This demonstrates the scalability of the trained model and the validity of our design of reward function and related training works.

VI. Conclusion

In this study, we propose a safety region-based representation paradigm tailored for the car-like rectangular-shaped robot and a FOMT-integrated reward function to explore narrow spaces without a map and waypoint guidance. By collision detection test among different state representation paradigms, we validate our claim on the effectiveness of the safety region. By intensive experiments in transferred seen and unseen tracks, we demonstrate the model with FOMT reward design could outperform the classical, IL methods and the RL method with guided reward design by contrastive studies. Besides, we analyzed the functionalities of the O and M component by ablation studies. Hereby we prove our proposed reward shaping is effective and interpretable. By final demonstration in the real-world narrow tracks, the policy generalizing power is validated.

Acknowledgment

The work is partially supported by the US National Science Foundation (NSF) under Grant No. 2140346.
authors also wish to express their gratitude to all individuals and organizations that contributed directly or indirectly to the success of this work.

REFERENCES

[1] V. Androulakis, J. Sotiile, S. Schafrik, and Z. Agioitiantis, “Concepts for development of autonomous coal mine shuttle cars,” IEEE Transactions on Industry Applications, vol. 50, no. 3, pp. 1908–1915, 2014.

[2] J. Q. Cui, K. K. Fan, L. Z. Fang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, F. Lin et al., “Drones for cooperative search and rescue in post-disaster sites,” in 2015 IEEE 7th forum on cybernetics and intelligent systems (CIS) and IEEE conference on robotics, automation and mechatronics (RAM). IEEE, 2015, pp. 167–174.

[3] Y. Ouyang, B. Li, Y. Zhang, T. Acarman, Y. Guo, and T. Zhang, “Fast and optimal trajectory planning for multiple vehicles in a nonconvex and cluttered environment: Benchmarks, methodology, and experiments,” in 2022 International Conference on Robotics and Automation (ICRA). IEEE, 2022, pp. 10746–10752.

[4] N. San, Q. Li, B. Li, and L. Li, “A successive linearization in feasible set algorithm for vehicle motion planning in unstructured and low-speed scenarios,” IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 4, pp. 3724–3736, 2021.

[5] V. Mohi, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” arXiv preprint arXiv:1312.5603, 2013.

[6] D. Silver, A. Huang, C. J. Madsen, 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.

[7] K. Nishi and N. Nakasu, “Evolvable motion-planning method using deep reinforcement learning,” in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 4079–4085.

[8] L. Xue, S. Wang, S. Rosa, A. Markham, and N. Trigoni, “Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 6276–6283.

[9] A. Haydari and Y. Yilmaz, “Deep reinforcement learning for intelligent transportation systems: A survey,” IEEE Transactions on Intelligent Transportation Systems, 2020.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, J. V. Den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacle,” in 2022 International Conference on Robotics and Automation Letters, vol. 4, no. 2, pp. 2007–2014, 2019.

[11] Z. J. Feng, R. J. Lu, Y. Qiu, Y. Jin, and C. Hu, “Navigation in unknown dynamic environments based on deep reinforcement learning,” Sensors, vol. 19, no. 18, p. 3837, 2019.

[12] F. Leiva and J. Ruiz-del Solar, “Robust rl-based map-less local planning: Using 2d point clouds as observations,” IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 5787–5794, 2020.

[13] D. Silver, A. Huang, C. J. Madsen, 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.

[14] K. Nishi and N. Nakasu, “Evolvable motion-planning method using deep reinforcement learning,” in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 4079–4085.

[15] L. Xue, S. Wang, S. Rosa, A. Markham, and N. Trigoni, “Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 6276–6283.

[16] A. Haydari and Y. Yilmaz, “Deep reinforcement learning for intelligent transportation systems: A survey,” IEEE Transactions on Intelligent Transportation Systems, 2020.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, J. V. Den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacle,” in 2022 International Conference on Robotics and Automation Letters, vol. 4, no. 2, pp. 2007–2014, 2019.

[18] Z. J. Feng, R. J. Lu, Y. Qiu, Y. Jin, and C. Hu, “Navigation in unknown dynamic environments based on deep reinforcement learning,” Sensors, vol. 19, no. 18, p. 3837, 2019.

[19] F. Leiva and J. Ruiz-del Solar, “Robust rl-based map-less local planning: Using 2d point clouds as observations,” IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 5787–5794, 2020.

[20] D. Silver, A. Huang, C. J. Madsen, 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.