Criticality-Guided Deep Reinforcement Learning for Motion Planning

Linling Xu[†], Fenghua Wu[‡], Yuan Zhou[‡*], Hesuan Hu[§], Zuohua Ding[†*], Yang Liu[†‡]
[†]Zhejiang Sci-Tech University, Hangzhou 310018, China
Email: 201930605055@mails.zstu.edu.cn, zuohuading@zstu.edu.cn
[‡]Nanyang Technological University, Singapore 639798
Email: {fenghua.wu, y.zhou, yangliu}@ntu.edu.sg
[§] Xidian University, Xi'an, Shaanxi 710071, China
Email: hshu@mail.xidian.edu.cn
[*]Corresponding authors

***Abstract*—Real-time and efficient collision avoidance is still challenging for mobile robots running in dynamic and crowded environments. Recent research shows that deep reinforcement learning (DRL) provides a framework to plan collision-free trajectories efficiently. However, most of the current DRL-based methods focus on a fixed number of obstacles in the environments, which limits their applications. In this paper, we propose a learning-based model, Crit-LSTM-DRL, for a robot moving in environments with a variable number of obstacles. It combines an LSTM (Long Short-Term Memory) model and a value-based DRL model. Given the states of a set of obstacles, Crit-LSTM-DRL first sorts the obstacles according to their possible collision time to the robot and then feeds to the LSTM model to generate a fixed-size hidden state. Then, the value-based DRL model takes the hidden state and robot state as input to compute the value. Hence, at any time step, an action is selected that maximizes the value function defined in the DRL framework. Finally, we compare the performance of Crit-LSTM-DRL with a state-of-the-art DRL-based planning method that aims to deal with a variable number of obstacles. The simulation results show that the three models of Crit-LSTM-DRL can improve the success rate by 4%, 20.1%, and 3.8%, and reduce the collision rate by 35.5%, 75%, and 66.7%, respectively.**

***Index Terms*—Motion Planning, Collision Avoidance, Obstacle Criticality, Deep Reinforcement Learning**

I. Introduction

Mobile robots can help people to do labor-consuming and dangerous tasks, such as environmental surveillance, disaster rescue, and minefield mapping [1]. Motion planning is the essential requirement for mobile robots to complete different tasks. It aims to generate a collision-free trajectory from an initial position to a destination. However, real-time collision avoidance is still challenging, especially in unknown and crowded environments.

In the past decades, researchers have proposed many methods for collision avoidance, such as discrete event systems [2]–[4], roadmap-based methods [5], [6], cell decomposition methods [7], state lattices [8], sampling-based methods [9], artificial potential fields [10], reciprocal collision avoidance [11], [12], and mathematical programming [13], [14]. However, in crowded environments, these methods may cause a high computation cost and sacrifice real-time efficiency.

Recently, to improve computational efficiency, deep reinforcement learning (DRL) has been applied for motion planning [15]–[17]. These works consider environments with a fixed number of obstacles. The most related work is [18]. In [18], a variable number of obstacles are processed by a long-short-term memory (LSTM) model and transformed into a fixed-size hidden state. Then, the DRL framework takes the hidden state and the robot state as input and selects an optimal action. In detail, the states of obstacles are first sorted according to their distances to the robot in descending order and then fed to the LSTM model to compute the hidden state. Hence, the closest obstacle has the biggest effect on the action selection. However, an obstacle is more critical and should be remembered for collision avoidance if it has a shorter collision time to the robot, even though it is far away. Hence, the method proposed in [18] may cause a high collision rate in crowded environments.

In this paper, we propose a value-based DRL approach, Crit-LSTM-DRL, to motion planning in environments having a variable number of obstacles. Rather than sorting obstacles according to their distances to the robot, we first propose a metric called criticality to evaluate the importance of an obstacle in the robot's collision avoidance. An obstacle has higher criticality if it has a shorter collision time to the robot. Hence, given a set of obstacles, Crit-LSTM-DRL first predicts their collision time and sorts them according to the collision time in descending order. If some obstacles do not cause collisions with the robot in the current situation, their collision time is positive infinity. In that case, they are sorted according to the distances to the robot. Then, the LSTM model takes the states of the sorted obstacles as the input sequence and generates a fixed-size hidden state. Third, Crit-LSTM-DRL sends the hidden state and the robot state to the value network to compute the corresponding value. Finally, Crit-LSTM-DRL selects the action that maximizes the value function. To evaluate the effectiveness of Crit-LSTM-DRL, we train three models in three kinds of environments: environments with 5 obstacles, 10 obstacles, and a variable number of obstacles from 1 to 10. The three models are tested on test sets

containing different numbers of obstacles, varying from 1 to 14. We compare Crit-LSTM-DRL with the method proposed in [18], which is the state-of-the-art DRL-based method to deal with a variable number of obstacles. The simulation results show that Crit-LSTM-DRL outperforms the existing method. The three trained models from Crit-LSTM-DRL improve the success rate by 4%, 20.1%, and 3.8%, and reduce the collision rate by 35.5%, 75%, and 66.7%, respectively.

The main contributions of this paper are that (1) we propose a new method based on the LSTM model to deal with a variable number of obstacles, and (2) we propose a value-based DRL model for motion planning of robots moving in the environments with a variable number of obstacles.

II. Preliminaries and Problem Statement

In this paper, we focus on the motion planning of holonomic robots moving around in unknown environments with a variable number of obstacles. However, our method can also extend to unicycle kinematics. The robot is equipped with different sensors, such as GPS and LiDAR, to identify its surrounding obstacles', i.e., position and velocity. The motion task for the robot is to move from the initial position $\mathbf{p}_0 = (x_0, y_0)$ to the target position $\mathbf{p}_g = (x_g, y_g)$ with a prefer speed v_f. Suppose the robot's radius is r, and the time is discretized into a set of time instants with an equal time step Δt. At any time instant k, $k \in \{0, 1, 2, \ldots\}$, the state of the robot is described as $s_k = (\mathbf{p}_k, \mathbf{v}_{k-1}, \mathbf{p}_g, v_f, r) \in \mathbb{R}^8$, where $\mathbf{p}_k = (x_k, y_k)$ is the robot's position at k, and $\mathbf{v}_{k-1} = (vx_{k-1}, vy_{k-1})$ is the velocity in the time duration $[(k-1)\Delta t, k\Delta t)$. The motion command at k is $\mathbf{u}_k = \mathbf{v}_k$, i.e., the velocity in the duration $[k\Delta t, (k+1)\Delta t)$. Note that for unicycle kinematics, $\mathbf{u}_k = (v, \theta)$, i.e., the speed and the orientation, and $vx_k = v\cos\theta$ and $vy_k = v\sin\theta$. The set of detected obstacles at k is denoted as $\mathcal{O}_k = \{o_1, \ldots, o_{n_k}\}$, and the state of each obstacle o_i is denoted as $s_k^i = (\mathbf{p}_k^i, \mathbf{v}_k^i, r_i)$, where $\mathbf{p}_k^i = (x_k^i, y_k^i)$, $\mathbf{v}_k^i = (vx_k^i, vy_k^i)$, and r_i are the position, velocity and radius of o_i at k, respectively. Note that $\mathbf{v}_k^i$ can be the current detected velocity of o_i or predicted by the optimal reciprocal collision avoidance (ORCA) method [11]. The state sequence of the obstacles at k are denoted as $s(\mathcal{O}_k)$. Hence, the motion problem for the robot can be described as:

$$\underset{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1}}{\arg\min} \quad T \tag{1}$$

$$s.t. \qquad \mathbf{p}_{k+1} = \mathbf{p}_k + \mathbf{u}_k \Delta t, \forall k \in \{0, 1, \ldots, T-1\}; \tag{2}$$

$$\|\mathbf{p}_k - \mathbf{p}_k^i\| \geq r + r_i, \forall o_i \in \mathcal{O}_k, k \in \{0, 1, \ldots, T\}; \tag{3}$$

$$\mathbf{p}_0 = \mathbf{p}_0, \ \mathbf{p}_T = \mathbf{p}_g. \tag{4}$$

According to [17], [18], such a problem can be resolved efficiently with the DRL framework by maximizing the value function:

$$\mathbf{u}_k^* = \arg\max_{\mathbf{u} \in \mathcal{A}} R(s_k, \mathcal{O}_k, \mathbf{u}) + \gamma^{\Delta t v_f} V^*(s_{k+1,\mathbf{u}}, s(\mathcal{O}_{k+1})),$$

$$V^*(s_k, s(\mathcal{O}_k)) = \sum_{k'=k}^{T-1} \gamma^{(k'-k)\Delta t v_f} R((s_{k'}, s(\mathcal{O}_{k'})), \mathbf{u}_{k'}^*),$$

where $\mathcal{A}$ is the set of predefined actions, $\gamma \in [0, 1]$ is a discount factor, $R(s_k, s(\mathcal{O}_k), \mathbf{u}_k)$ is one-step reward at s_k by taking $\mathbf{u}_k$, $s_{k+1,\mathbf{u}}$ is the robot's next state under the action $\mathbf{u}$. Following [18], the reward function is defined as follows:

$$R(s_k, \mathcal{O}_k, \mathbf{u}_k) = \begin{cases} 1, & \mathbf{p}_{k+1} = \mathbf{p}_g \\ -0.25, & d_{\min} \leq 0 \\ -0.1 + 0.5 d_{\min}, & 0 < d_{\min} \leq 0.2 \\ 0, & otherwise \end{cases}$$

where $d_{\min}$ is the minimal distance between the robot and the obstacles in $\mathcal{O}_k$ during the time duration $[k\Delta t, (k+1)\Delta t]$.

In this paper, we target to design a DRL-based controller for the robot moving around in crowded and dynamic environments, which means that the dimension of $s(\mathcal{O}_k)$ may change over time. Hence, our problem can be formulated as follows.

Problem: *Given the initial position $\mathbf{p}_0$, target position $\mathbf{p}_g$, and prefer speed v_f of a robot in an unknown, crowded, and dynamic environment, design a DRL controller to control the robot to move from $\mathbf{p}_0$ to $\mathbf{p}_g$ and avoid collisions with a variable number of obstacles in the environment.*

III. Criticality-Guided Deep Reinforcement Learning

The section illustrates our criticality-guided DRL solution for the stated motion problem. The main idea is that the detected obstacles are first organized based on their criticality levels and then transformed into a fixed-size state.

A. Determination of Obstacle Criticality

First, following the descriptions of [16]–[18], we transform the states of the robot and the obstacles from the global coordinate system to the robot-centric one. As shown in Fig. 1, $\mathbf{p}_k$ and $\mathbf{p}_g$ are the current and target positions of the robot, respectively, $\mathbf{v}$ is the current velocity of the robot, and $\mathbf{p}_k^i$ is the position of an obstacle o_i in $\mathcal{O}_k$. The transformation from xy to $\overline{xy}$ can be described as follows:

$$\overline{\mathbf{p}} = M(\mathbf{p} - \mathbf{p}_k), \ \overline{\mathbf{v}} = M\mathbf{v},$$

where $M = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$ and $\varphi = \arctan\frac{y_g - y_k}{x_g - x_k}$. Hence, we can rewrite the states of the robot and any obstacle as $\overline{s}_k = (d_g, v_f, \overline{vx}_{k-1}, \overline{vy}_{k-1}, r)^T$ and $\overline{s}_k^i = (\overline{x}_k^i, \overline{y}_k^i, \overline{vx}_k^i, \overline{vy}_k^i, r_i, d_i, r_i + r)^T$, where $d_g = \|\mathbf{p}_g - \mathbf{p}_k\|_2$ and $d_i = \|\mathbf{p}_k^i - \mathbf{p}_k\|_2$. Let $\overline{s}(\mathcal{O}_k) = (\overline{s}_k^1, \overline{s}_k^2, \ldots, \overline{s}_k^{n_k})$, where $o_i \in \mathcal{O}_k$, $i = 1, 2, \ldots, n_k$.

Given the current position $\overline{\mathbf{p}}_k^i$ and velocity $\overline{\mathbf{v}}_k^i$ of obstacle o_i in the local coordinate system, and the robot's velocity $\overline{\mathbf{v}}_{k-1}$, the determination of collision between them can be evaluated as follows. As shown in Fig. 1, the relative velocity can be computed as $\Delta\overline{\mathbf{v}}_k = \overline{\mathbf{v}}_{k-1} - \overline{\mathbf{v}}_k^i$, and the relative position is $\overline{\mathbf{p}}_k^i$, i.e., $\overrightarrow{\mathbf{p}_k \mathbf{p}_k^i}$. Hence, the minimal distance between the robot and o_i is:

$$d(\mathbf{p}_k, A) = \frac{\overline{\mathbf{p}}_k^i \cdot \Delta\overline{\mathbf{v}}_k}{\|\Delta\overline{\mathbf{v}}_k\|_2}, \tag{5}$$

$$d_{i,\min}^2 = \|\overline{\mathbf{p}}_k^i\|_2^2 - d^2(\mathbf{p}_k, A). \tag{6}$$

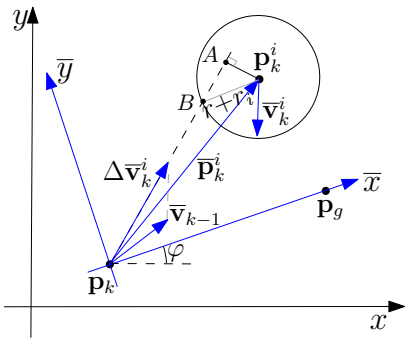


Fig. 1. Transformation of coordinate systems.

Clearly, if $d_{\min} \leq r + r_i$, the robot and o_i would cause a collision if they continued their current velocities. In this case, the computation of possible collision time, denoted as t_i, can be computed as follows.

$$d(A,B) = \sqrt{(r+r_i)^2 - d_{i,\min}^2}, \tag{7}$$

$$t_i = \frac{d(\mathbf{p}_k, A) - d(A,B)}{\|\Delta\overline{\mathbf{v}}_k\|}. \tag{8}$$

Clearly, the smaller t_i is, the more dangerous and higher priority o_i is for collision avoidance. Note that when $d_{i,\min} > r + r_i$, the robot and o_i cannot cause a collision if they continue their current velocities. Hence, at any time instant k, the obstacles in $\mathcal{O}_k$ can be divided into two classes: $\mathcal{O}_k^\alpha = \{o_i : d_{i,\min} \leq r + r_i\}$ and $\mathcal{O}_k^\beta = \{o_i : d_{i,\min} > r + r_i\}$.

Definition 1. *Given two obstacles o_i and o_j in $\mathcal{O}_k$, o_i has higher criticality, denoted as $c(o_i) \geq c(o_j)$, if it satisfies one of the following conditions: (1) $o_i \in \mathcal{O}_k^\alpha$ and $o_j \in \mathcal{O}_k^\beta$; (2) $o_i, o_j \in \mathcal{O}_k^\alpha$ and $t_i \leq t_j$; and (3) $o_i, o_j \in \mathcal{O}_k^\beta$ and $d_i \leq d_j$.*

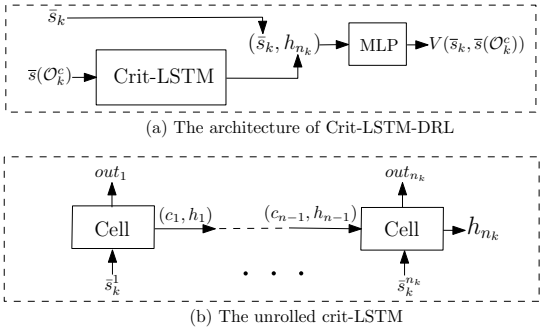
Definition 2. *The sequence of obstacles $\{o_1, o_2, \ldots, o_{n_k}\}$, $o_i \in \mathcal{O}_k$, is called a critical sequence, denoted as $\mathcal{O}_k^c$, if it satisfies $\forall i > j$, $c(o_i) \geq c(o_j)$.*

At any time instant k, the critical sequence can be determined as follows. First, we divide all the detected obstacles into $\mathcal{O}_k^\alpha$ and $\mathcal{O}_k^\beta$. Second, we sort the obstacles in $\mathcal{O}_k^\beta$ according to their distance to the robot in descending order, then sort the obstacles in $\mathcal{O}_k^\alpha$ according to their collision time in descending order. Finally, the two sub-sequences are concatenated, and we can obtain the critical sequence of the obstacles. The critical state sequence is denoted as $\overline{s}(\mathcal{O}_k^c) = (\overline{s}_k^1, \overline{s}_k^2, \ldots, \overline{s}_k^{n_k})$, where $\mathcal{O}_k^c = \{o_1, o_2, \ldots, o_{n_k}\}$.

B. Deep Reinforcement Learning with Criticality-LSTM

Crit-LSTM-DRL is a value-based DRL method. The main step in Crit-LSTM-DRL is to train a value network that takes the states of the robot and the critical obstacle sequence as input to compute the value. The network architecture of Crit-LSTM-DRL is shown in Fig. 2(a), where the Crit-LSTM is an LSTM model taking $\overline{s}(\mathcal{O}_k^c)$ as the input sequence (shown in Fig. 2(b)), and MLP is a multi-layer perceptron taking the concatenation of $\overline{s}_k$ and h_{n_k} as input.

The learning process is shown in Algorithm 1. Recall that s and $\overline{s}$ denote the states in the global and the robot-centric coordinate systems, respectively. First, the value network is



(a) The architecture of Crit-LSTM-DRL

(b) The unrolled crit-LSTM

Fig. 2. The network architecture of Crit-LSTM-DRL, where $\mathcal{O}_k^c$ is the critical obstacle sequence.

Algorithm 1: Deep V-Learning

Input: The maximal number of episodes $N_{\max}$, the number of replay iterations N_{replay}, the action space $\mathcal{A}$, the maximal time steps for robot motion $T_{\max}$, probability for greedy selection ϵ, and update frequency for the target value network C.

Output: Value network V

1 Generate a set of trajectories via ORCA;
2 Initialize a replay memory E based on the trajectories;
3 Initialize a value network V with supervised learning based on memory E;
4 Initialize the target value network $\widetilde{V} = V$, $episode = 0$;
5 **for** $episode = 1 : N_{\max}$ **do**
6 **for** $ite = 1 : N_{replay}$ **do**
7 Sample s_0 and $s(\mathcal{O}_0)$;
8 $traj = \{(s_0, s(\mathcal{O}_0))\}$, $done = False$, $k = 0$;
9 **while** *not done* **do**
10 generate a random value p between $[0, 1]$;
11 **if** $p < \epsilon$ **then**
12 Select $\mathbf{u}_k$ randomly from $\mathcal{A}$;
13 **else**
14 $\mathbf{u}_k = \arg\max_{\mathbf{u}\in\mathcal{A}} R(s_k, \mathcal{O}_k, \mathbf{u}) + \gamma^{\Delta t v_f} V(\overline{s}_{k+1,\mathbf{u}}, \overline{s}(\mathcal{O}_{k+1}^c))$;
15 Move to the next state $\mathbf{s}_{k+1}$;
16 Detect and update the obstacles' states $s(\mathcal{O}_{k+1})$;
17 $traj = traj \cup \{(\mathbf{s}_{k+1}, s(\mathcal{O}_{k+1}))\}$, $k = k + 1$;
18 **if** $k \geq T_{\max}$ *or collided or reached* **then**
19 $done = True$;
20 **if** *collided or reached* **then**
21 **for** $\forall(\mathbf{s}_k, s(\mathcal{O}_k)) \in traj$ **do**
22 $v_k = \widetilde{V}(\overline{\mathbf{s}}_k, \overline{s}(\mathcal{O}_k^c))$;
23 update the memory E with the pair $((\overline{\mathbf{s}}_k, \overline{s}(\mathcal{O}_k^c)), v_k)$;
24 Update the network V by gradient decent with E;
25 **if** $(episode \ mod \ C) == 0$ **then**
26 $\widetilde{V} = V$;
27 **return** V.

initialized using a supervised learning process (Lines 1–3), where the training data consists of a set of $(state, value)$ pairs based on a set of trajectories generated from the ORCA method. Second, for each episode, we replay robot motion N_{replay} times and then update the replay memory E. For each replay, we first initialize an environment $(s_0, s(\mathcal{O}_0))$. At each time step k, the previous value network is used

to select an action from the action space $\mathcal{A}$ that maximizes the value (Line 14). Note that $s(\mathcal{O}_{k+1})$ is predicted by assuming that each obstacle moves at a constant velocity $\mathbf{v}_k^i$ in $[k\Delta t, (k+1)\Delta t)$. It will also be used to compute the one-step reward. To improve exploitation, ϵ-greedy policy is applied (Line 12). The replay is terminated if the robot reaches the destination, or a collision is detected, or the maximal motion time is reached (Lines 18 and 19). Only when the current replay is terminated with task achievement or collisions will the generated trajectory be used to update the replay memory using the $(state, value)$ pairs. During the training process, the value $value$ at each state $state$ is computed based on the target value network $\widetilde{V}$ (Lines 20–23). The target value network is updated every C episodes.

IV. Experimental Evaluation

We evaluate Crit-LSTM-DRL via simulation based on the project [1] implemented in [16]. The dimension of the hidden state of the LSTM model is 50, and the neurons in the MLP are $(150, 100, 100, 1)$. The prefer speed is 1, and the discrete action space is $\mathcal{A} = \{(v_i \cos\theta_j, v_i \sin\theta_j) : v_i = \frac{e^{i/5}-1}{e-1}, \theta_j = \frac{j}{8}\pi, i = 0, 1, \ldots, 5, j = 0, 1, \ldots, 15\}$. At the training phase, the model is trained in three environments, i.e., environments with 5 obstacles, 10 obstacles, and a variable number of obstacles (from 1 to 10), respectively. Each model is trained with 10000 episodes (i.e., $N_{\max} = 10000$), and each episode contains 100 training batches. The trained models are denoted as Crit-LSTM-DRL-5, Crit-LSTM-DRL-10, and Crit-LSTM-DRL-D, respectively. At the testing phase, we repeat the testing 10 times. For each testing run, we randomly generate 1500 test cases, dividing into five test sets equally, i.e., the sets of test cases containing 1–4 (set1), 5 (set2), 6–9 (set3), 10 (set4), and 11–14 (set5) obstacles, respectively. We compare our method with the LSTM-DRL method [18]. It is also trained 10000 episodes in the same training environments, resulting in three models: LSTM-DRL-5, LSTM-DRL-10, and LSTM-DRL-D.

A. Evaluation of Crit-LSTM-DRL

1) Computational Performance: The computation time for training and prediction is evaluated on a computer with an Intel® Xeon(R) CPU E5-2697 v3. Crit-LSTM-DRL takes 26.38, 55.9, and 31.43 hours to complete 10000 training episodes in the three training environments, respectively. The average one-step decision-making time of the three models trained by Crit-LSTM-DRL is 174.31, 177.06, 180.50ms, respectively. Note that the training and prediction time relies on the action space. The larger the action space is, the longer the time is, as Crit-LSTM-DRL needs to query each action in the action space and select the best one. In our experiments, there are 80 actions, so the average query time for one action is around 2.2ms.

[1] https://github.com/vita-epfl/CrowdNav

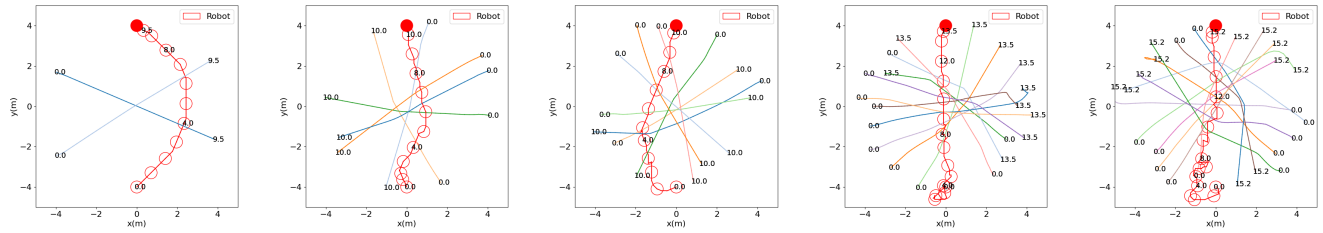
2) Simulation Results: In the sequel, to show the effectiveness of Crit-LSTM-DRL, we visualize some representative trajectories generated by different trained models with a different number of obstacles. Fig. 3 shows the trajectories of five test cases generated by the three trained models, where the numbers in the trajectories denote motion time, and each circle denotes the robot at each second. From Fig. 3(a), we can find that Crit-LSTM-DRL-5 performs well in the test cases with 2, 5, and 7 obstacles, respectively, but there are violent oscillations on the trajectories generated for the test cases with 10 and 12 obstacles. In Fig. 3(b), Crit-LSTM-DRL-10 performs well for the test cases with 5, 7, and 10 obstacles, respectively, and causes a few oscillations for the test case with 12 obstacles; however, for the test case with 2 obstacles, Crit-LSTM-DRL-10 makes a long detour. As shown in Fig. 3(c), Crit-LSTM-DRL-D generates proper trajectories for the test cases with 2, 5, 7, and 10 obstacles, respectively, but a trajectory with oscillations for the case with 12 obstacles.

B. Performance Comparison with LSTM-DRL

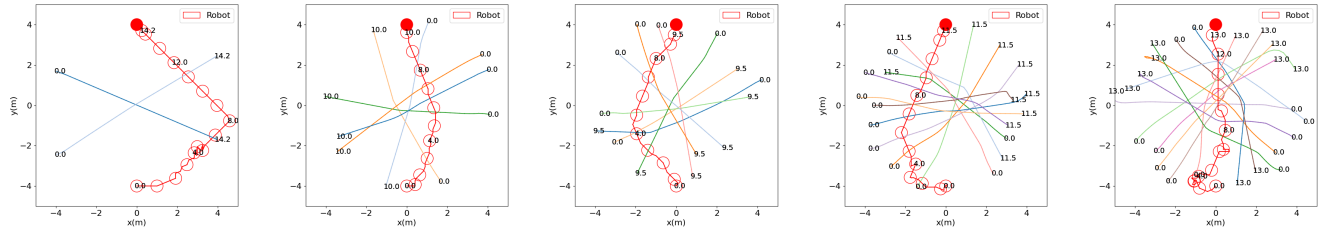
We compare the performance of Crit-LSTM-DRL with LSTM-DRL. The metrics for comparison are the cumulative rewards during the training phase, and the success, collision, and timeout rates during the testing phase. Note that a motion is a success (resp., collision) if the robot arrives at the destination (resp., causes a collision) within the given time budget, while a motion is a timeout if the robot does not cause collisions or reach the destination within the time budget.

Fig. 4 shows the cumulative reward at each episode during the training. The results show that in each environment, Crit-LSTM-DRL outperforms LSTM-DRL. Either method obtains the lowest cumulative rewards in the environment with 10 obstacles. It is because, with the increase of obstacles in the environments, the robot has a higher chance of getting close to the obstacles, resulting in a negative one-step reward. Hence, either method obtains the lowest cumulative reward with 10 obstacles and the highest cumulative reward with 5 obstacles.

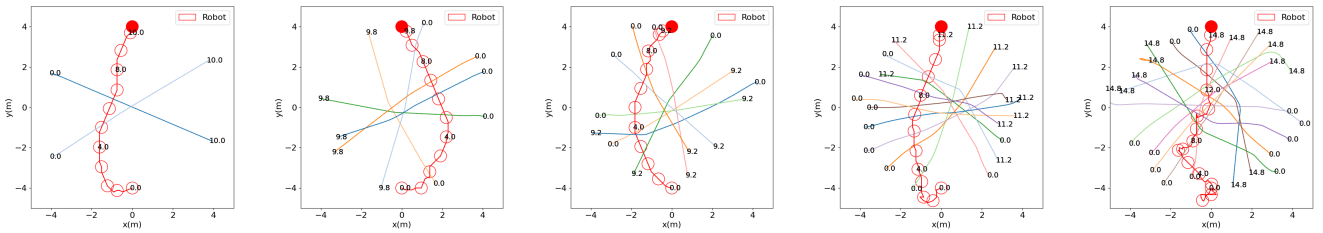
Fig. 5 shows the success/collision/timeout rate of each test set in each testing run, and Table I gives each testing set's average rates of the 10 testing runs, as well as the total average rates for each trained model. First, for Crit-LSTM-DRL-5, it shows a higher success rate and a lower collision rate than LSTM-DRL-5. In detail, Crit-LSTM-DRL-5 and LSTM-DRL-5 show similar performance on the first two test sets, but Crit-LSTM-DRL-5 shows a much higher success rate than LSTM-DRL-5 on the third and fourth sets; both do not perform well on the test case containing 12 obstacles. Second, Crit-LSTM-DRL-10 also outperforms LSTM-DRL-10: Crit-LSTM-DRL-10 can achieve 96.8% success rate and only 0.8% collision rate on average, while the success and collision rates are 80.6% and 3.2%, respectively. LSTM-DRL-10 performs much better than LSTM-DRL-10 on each test set. For example, Crit-LSTM-DRL-10 can achieve more than 90% success rate on the first set, while the success rate of LSTM-DRL-10 is less than 50%. Third, Crit-LSTM-DRL-D shows similar performance to LSTM-DRL-D on the first two



(a) Trajectories generated by Crit-LSTM-DRL-5. Left to Right: Test case with 2, 5, 7, 10, 12 obstacles, respectively.



(b) Trajectories generated by Crit-LSTM-DRL-10. Left to Right: Test case with 2, 5, 7, 10, 12 obstacles, respectively.



(c) Trajectories generated by Crit-LSTM-DRL-D. Left to Right: Test case with 2, 5, 7, 10, 12 obstacles, respectively.

Fig. 3. Trajectories generated by Crit-LSTM-DRL-5, Crit-LSTM-DRL-10, and Crit-LSTM-DRL-D.

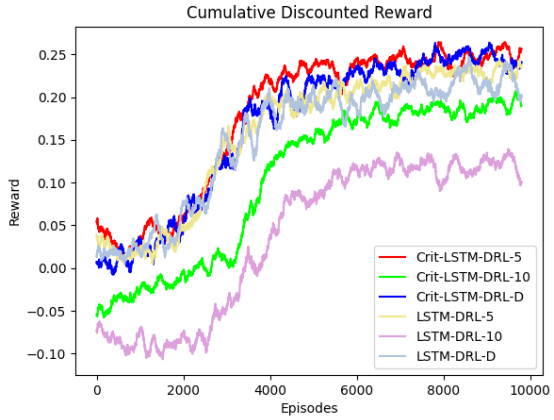
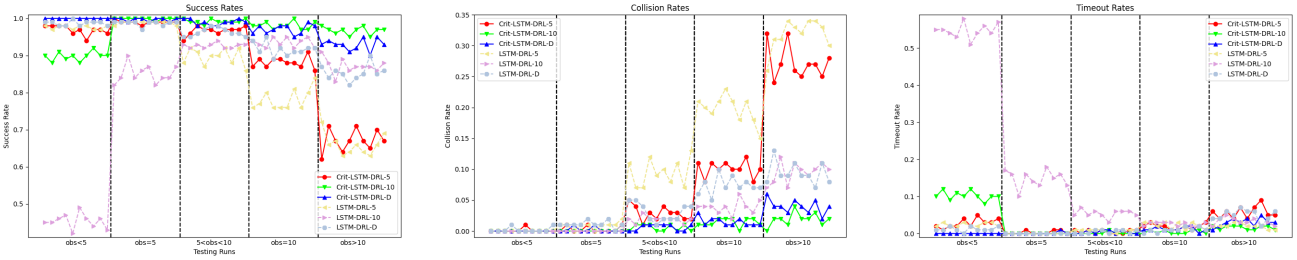


Fig. 4. The curves of cumulative rewards during training.

test sets and performs much better (i.e., higher success rate, lower collision rate, and lower timeout rates) on the rest three test sets. In conclusion, compared with LSTM-DRL-5, Crit-LSTM-DRL-5 improves the total average success rate by 4% $((0.896 - 0.862)/0.862)$, and reduces the collision rate by 35.5% $((0.124 - 0.08)/0.124)$; Crit-LSTM-DRL-10 and Crit-LSTM-DRL-D improve the success rate by 20.1% and 3.8%, respectively, and reduce the collision rate by 75% and 66.7%, respectively. Hence, Crit-LSTM-DRL significantly improves the performance, e.g., increasing success rate and reducing collision rate, of LSTM-DRL.

In the sequel, we compare the performance of the three models of Crit-LSTM-DRL. Crit-LSTM-DRL-5 only performs well in the first two sets, and the performance degrades when the number of obstacles increases, limiting its application in crowded environments. Crit-LSTM-DRL-10 shows good performance in all testing sets except set1, in which Crit-LSTM-DRL-10 achieves a low success rate and a high timeout rate. It is because Crit-LSTM-DRL-10 is trained in a crowded environment, and the reward function is too sparse to inspire the robot to move to the destination. Crit-LSTM-DRL-D achieves a high success rate in the first four test sets, where the range of the number of obstacles is the same as that in the training environments, and has a little downgrade in the environments whose number of obstacles exceeds the training one. Hence, if the maximal number of obstacles in different applications can be determined, Crit-LSTM-DRL is preferred to be trained in environments with a variable number of obstacles. Otherwise, we may want to train two Crit-LSTM-DRL models: one is trained with a fixed number of obstacles and the other is trained with a variable number of obstacles, and make decisions between them based on the detected number of obstacles.



(a) Success rate. (b) Collision rate (c) Timeout rate.

Fig. 5. Performance of models trained in different environments for each test run.

TABLE I
AVERAGE RATES IN 10 RUNS ($Success/Collision/Timeout$)

Models	Total Average Rates	Average Rates in Different Test Sets				
		1-4 (set1)	5 (set2)	6-9 (set3)	10 (set4)	11-14 (set5)
Crit-LSTM-RL-5	0.896/0.080/0.024	0.97/0.00/0.03	0.99/0.00/0.00	0.97/0.03/0.01	0.88/0.10/0.02	0.67/0.27/0.06
Crit-LSTM-RL-10	0.968/0.008/0.026	0.90/0.00/0.10	1.00/0.00/0.00	**0.99/0.01/0.00**	**0.98/0.01/0.00**	**0.97/0.02/0.02**
Crit-LSTM-RL-D	0.978/0.014/0.008	**1.00/0.00/0.00**	**1.00/0.00/0.00**	**0.99/0.01/0.00**	0.97/0.02/0.01	0.93/0.04/0.03
LSTM-RL-5	0.862/0.124/0.014	0.98/0.00/0.02	0.99/0.01/0.00	0.90/0.09/0.01	0.78/0.20/0.02	0.66/0.32/0.02
LSTM-RL-10	0.806/0.032/0.162	0.45/0.00/0.55	0.85/0.00/0.15	0.93/0.02/0.05	0.93/0.04/0.03	0.87/0.10/0.03
LSTM-RL-D	0.942/0.042/0.016	0.99/0.00/0.01	0.99/0.01/0.00	0.96/0.03/0.01	0.92/0.07/0.01	0.85/0.10/0.05

V. CONCLUSION

In this paper, we propose a learning-based approach, Crit-LSTM-DRL, to real-time motion planning for a robot moving in dynamic environments. It combines an LSTM model and a value-based DRL model. At any time step, the LSTM model transforms the varying-size obstacle vector into a fixed-size one according to the obstacles' criticality. Then, the DRL model selects an optimal action to maximize the value. We compare Crit-LSTM-DRL with LSTM-DRL. The simulation results show that Crit-LSTM-DRL can improve the success rate and reduce the collision rate significantly.

In the future, we will evaluate our method on a real-world robot platform and design new reward functions to reduce the timeout rate. Another interesting topic is investigating the combination of learning-based methods and conventional planning methods to guarantee both safety and efficiency.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grant 61973242, the Major Fundamental Research Program of the Natural Science Foundation of Shaanxi Province under Grant No. 2017ZDJC-34, and Singapore MOE Academic Research Fund Tier 2 grant (MOE-T2EP20120-0004).

REFERENCES

[1] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooper. Robots Sensor Netw.*, pp. 31–51, 2015.

[2] J. Luo, H. Ni, and M. Zhou, "Control program design for automated guided vehicle systems via petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 1, pp. 44–55, 2014.

[3] Y. Zhou, H. Hu, Y. Liu, and Z. Ding, "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 7, pp. 1712–1726, 2017.

[4] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A distributed method to avoid higher-order deadlocks in multi-robot systems," *Automatica*, vol. 112, pp. 108 706:1 – 108 706:13, 2020.

[5] J. P. Van Den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Trans. Robot.,*, vol. 21, no. 5, pp. 885–897, 2005.

[6] J. D. Marble and K. E. Bekris, "Asymptotically near-optimal planning with probabilistic roadmap spanners," *IEEE Trans. Robot.*, vol. 29, no. 2, pp. 432–444, 2013.

[7] M. Kloetzer, C. Mahulea, and R. Gonzalez, "Optimizing cell decomposition path planning for mobile robots using different metrics," in *Proc. Int. Conf. Syst. Theory Control Comput.*, 2015, pp. 565–570.

[8] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *J. Field Robot.*, vol. 26, no. 3, pp. 308–333, 2009.

[9] A. Bircher, K. Alexis, U. Schwesinger, S. Omari, M. Burri, and R. Siegwart, "An incremental sampling-based approach to inspection planning: The rapidly exploring random tree of trees," *Robotica*, vol. 35, no. 6, pp. 1327–1340, 2017.

[10] H. G. Tanner and A. Boddu, "Multiagent navigation functions revisited," *IEEE Trans. Robot.*, vol. 28, no. 6, pp. 1346–1359, 2012.

[11] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, 2011, pp. 3–19.

[12] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Trans. Robot.*, vol. 34, no. 2, pp. 404–420, 2018.

[13] P. Abichandani, G. Ford, H. Y. Benson, and M. Kam, "Mathematical programming for multi-vehicle motion planning problems," in *IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3315–3322.

[14] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 12, pp. 2636–2650, 2017.

[15] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1343–1350.

[16] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6015–6022.

[17] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE Int. Conf. Robot. Autom.*, 2017, pp. 285–292.

[18] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *IEEE/RSJ Int.l Conf. Intell. Robot. Syst.*, 2018, pp. 3052–3059.