Deep Reinforcement Learning Based Collision Avoidance Algorithm for Differential Drive Robot

Xinglong Lu, Yiwen Cao, Zhonghua Zhao(✉), and Yilin Yan

Department of Instrument Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China
zhaozh@sjtu.edu.cn

Abstract. In this paper, collision avoidance problem is investigated for differential drive robot running in pedestrian environment, which requires for natural and safe interaction between robot and human. Based on deep reinforcement learning, a human-aware collision avoidance algorithm is proposed to find a smooth and collision-free path. A well designed reward function ensures the robot navigates without collision and obeys right-pass norm simultaneously. The slow convergence problem during training is addressed by pre-training the neural network using supervised learning. The simulation results show that the proposed algorithm can find a feasible and norm-obeyed path which achieves a natural human-robot interaction compared with traditional method.

Keywords: Deep reinforcement learning · Collision avoidance
Differential drive robot

1 Introduction

Collision avoidance is a crucial part of autonomous navigation under pedestrian environment for service robot, such as companion robot [1], guide robot [2] and intelligent wheelchair [3]. These robots usually use two-wheel differential drive as a chassis because of its simple structure and omnidirectional movement characteristics. The traditional collision avoidance algorithm that only seeks for finding a collision-free and time efficient path. However, collision avoidance between robot and pedestrian should also consider how to make robot's behavior acceptable, which means that the generated trajectory should be as smooth as possible with no sudden start/stop or sharp turn. Unfortunately, planning a collision-free, time efficient and smooth path under pedestrian environment is still challenging, because of the uncertain behavior of pedestrian. This task requires for precise perception of the human behavior and estimation of the pedestrian's intent (goal).

The common method to solve this problem is trying to model the control strategy or the motion of pedestrian, which can be considered as model-based

© Springer Nature Switzerland AG 2018
Z. Chen et al. (Eds.): ICIRA 2018, LNAI 10984, pp. 186–198, 2018.
https://doi.org/10.1007/978-3-319-97586-3_17

method. A Multi-Policy Decision Making (MPDM) framework [4] is proposed to make robot navigate amongst pedestrians by dynamically switching between *Go-Solo*, *Follow-other*, and *Stop* three policies. Social force model (SFM) [5] is designed for social force analysis when robot interacts with pedestrians and SFM has been implemented by many robot navigation applications [2,6]. But model-based method is difficult to generalize to unseen scenarios, because tuning a set of parameters that fit all environment is almost impossible. In addition, this method usually causes unnatural behavior, such as sudden start/stop and sharp turn.

Learning-based method is proposed to avoid time-consuming parameters tuning by off-line learning and on-line execution. An end-to-end framework is designed to map 2D-Lidar output to control command for multi-agent collision avoidance problem in [7]. Reinforcement learning is used to learn the optimal parameters of SFM [2]. To achieve autonomous navigating, the work in [8] teaches robot to imitate the behavior of pedestrians with inverse reinforcement learning. With the development of deep neural network (DNN), reinforcement learning uses DNN to approximate the value function or action-value function, which has been successfully implemented on playing Atari computer games [9] and controlling quadrotor [10]. The reinforcement learning method utilizes reward function to teach agent the rules that it should obey, rather than designing complicated control strategy. If adequate training data provided or exploring enough state space, deep reinforcement learning could obtain a optimal policy that maps current state to the best action.

In this work, we focus on collision avoidance when differential drive robot navigates under pedestrian environment and learn a policy that map the current state to best action using deep reinforcement leaning. The remainder of this paper is organized as follows. Section 2 introduces the detail of the problem formulation. The approach to train and test the deep reinforcement learning model is described in Sect. 3. Section 4 shows the simulation results and analysis. Finally, the conclusion of our work is delivered in Sect. 5.

2 Problem Formulation

In this section, the dynamics of differential drive robot and pedestrian is modeled at first, and then collision avoidance problem is represented in the format of Markov decision process (MDP). At last, deep reinforcement learning is formulated to solve the MDP problem.

2.1 Agent Modeling

Differential drive robot and pedestrian are considered as two types agent in this work. To get state space and action space in reinforcement learning, the dynamics of robot and pedestrian should be modeled first. As show in Fig. 1, the two-wheel geometry and gray circle represent differential drive robot and pedestrian respectively. The star gives the goal position for robot.

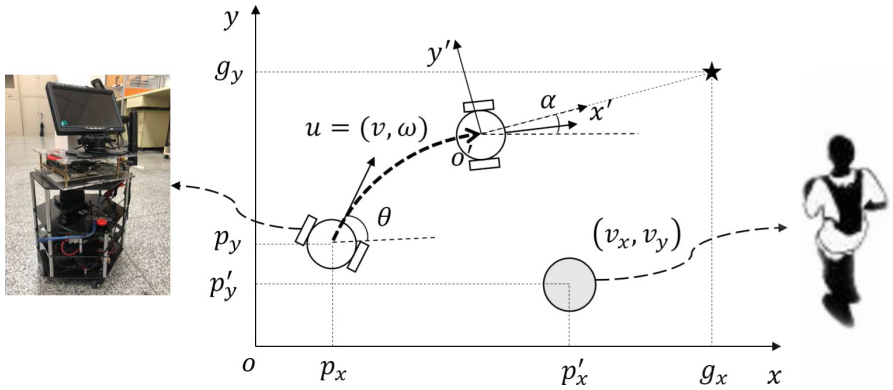

Fig. 1. Differential drive robot and pedestrian modeling

Differential Drive Robot: The state of differential drive robot contains position (p_x, p_y), heading angle θ, linear velocity v, angular velocity ω, radius of robot's projection circle r. The goal position (g_x, g_y) is also necessary in navigation task. To be compatible with ROS navigation package [13], velocity tuple (v, ω) is used as control command u. Suppose the differential drive robot runs from point $A(p_{x_t}, p_{y_t})$ to point $B(p_{x_{t+1}}, p_{y_{t+1}})$ with the command $u = (v_t, \omega_t), \omega_t \neq 0$ in Δt. The motion function is

$$p_{x_{t+1}} = p_{x_t} - \frac{v_t}{\omega_t} sin(\theta_t) + \frac{v_t}{\omega_t} sin(\theta_t + \omega_t \Delta t) \tag{1}$$

$$p_{y_{t+1}} = p_{y_t} + \frac{v_t}{\omega_t} cos(\theta_t) - \frac{v_t}{\omega_t} cos(\theta_t + \omega_t \Delta t) \tag{2}$$

$$\theta_t = \theta_t + \omega_t \Delta t \tag{3}$$

Pedestrian: Actually the state of pedestrian should be the same as robot, but the goal of pedestrian is not observable. Because it is hard to measure the heading angle of pedestrian, angular velocity can not be computed directly. For similarity, pedestrian's state includes position (p'_x, p'_y) and velocity (v_x, v_y) and its motion function is

$$p'_{x_{t+1}} = p'_{x_t} + v_x \Delta t \tag{4}$$

$$p'_{y_{t+1}} = p'_{y_t} + v_y \Delta t. \tag{5}$$

2.2 Collision Avoidance and MDP Representation

A collision avoidance problem under pedestrian environment can described as that a robot navigates from start point to goal position without collision and interacting with human safely and naturally. Generally a collision avoidance consist two part: perception module and decision module. Perception module is responsible for getting the state of robot and surrounding objects, such as

static obstacle and moving obstacle. Decision module provides control command following a designed strategy based on the perception. In this work, only decision module is considered and moving obstacle is pedestrian. We are working on the assumption that only the nearest pedestrian along the path to goal is considered in multi-pedestrian environment. So the collision avoidance in pedestrian environment can be simplified to one robot interacting with one pedestrian situation.

To apply reinforcement learning to solve collision avoidance problem, a MDP representation should be abstracted. The MDP is a tuple (S, A, R), where S is state space, A is action space, R is reward function.

State Space S: The collision avoidance algorithm should only rely on the robot local coordination, which means the state need to be coordination invariant. So a robot-fixed coordinate $x'o'y'$ is used, which sets origin as robot's center point and x axis pointing to the robot's goal, as show in Fig. 1. Then the state is transformed to the robot-fixed coordination using Eqs. 6 and 7.

$$\begin{bmatrix} \tilde{p}'_x \\ \tilde{p}'_y \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\alpha) & sin(\alpha) & -p_x \\ -sin(\alpha) & cos(\alpha) & -p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \\ 1 \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \tilde{v}_x \\ \tilde{v}_y \end{bmatrix} = \begin{bmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{7}$$

The state can be formulated as $s = [v, \omega, \theta, r, d_g, \tilde{p}'_x, \tilde{p}'_y, \tilde{v}_x, \tilde{v}_y, d_p] \in \mathbf{R}^{10}$, where d_g is the distance between robot and its goal, d_p is the distance between pedestrian and robot.

Action Space A: As velocity command is used to steer the service robot, action can be described as linear velocity v and angular velocity ω. The action space is discretized into finite tuples (v, ω).

Reward Function R: The reward function gives the reward feedback when robot executes an action from current state. It can be used to train the agent to obey some rules. Inspired by dynamic window approach (DWA) to collision avoidance algorithm [14], the reward function is set to reward reaching goal, high linear velocity and punish nearing pedestrian and collision.

* Reaching goal: reward 1;
* High linear velocity: if $|v| > \tau_v, 0 < \tau_v < v_{max}$, reward 0.5, where τ_v is the threshold of linear velocity, v_{max} is the max linear velocity;
* Nearing Pedestrian: if $0 < d_p < \tau_d$, reward $-\tau_d + d_p$, where τ_d is the threshold of distance between robot and pedestrian, which can be used to set a safety margin for robot when navigates near pedestrian;
* Collision: if $d_p < 0$, reward -0.5.

To make the interaction between robot and human more acceptable, a right-pass norm is added into reward function. As shown in Fig. 2, the sad face and

happy face means penalty and reward respectively. Specifically, the right-pass norm reward right hand passing, left hand overtaking and crossing behind. If the pedestrian does not appear in the norm-obeyed area, reward −0.5.

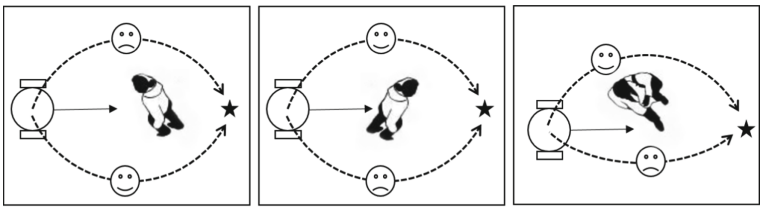

Fig. 2. Right-pass norm. (a) Passing; (b) Overtaking; (c) Crossing.

2.3 Deep Reinforcement Learning

The goal of deep reinforcement learning is to learn an optimal policy π through exploring the environment with reward feedback. This means the agent chooses actions in a way that maximizes cumulative future reward. Deep Q-network (DQN) [9] uses a convolutional neural network (CNN) to approximate the optimal action-value function

$$Q^{\star}(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{8}$$

which is the sum of reward r_t at time step t and discounted by γ. And the reward is obtained by following the given policy π and executing an action a selected based on state s.

In this work, the action space is continuous and there is strong correlation between action and state. So it is hard to directly approximate action-value function. Instead, we approximate value function that denotes the expected time to reach goal based on the current state. A deep neural network (deep V-network) is used to approximate the value function, that is:

$$V^{\star}(s) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, \pi], \tag{9}$$

where only state is considered. The optimal policy $\pi^{\star}$ can be obtained by choosing the action that maximizes immediate reward $R(s,a)$ and next state's value,

$$\pi^{\star}(s) = \max_{a}(R(s,a) + V^{\star}(s'(s,a))), \tag{10}$$

where $s'(s,a)$ is the next state that can be computed by motion function of differential drive robot and pedestrian in Sect. 2.1.

The optimal value function $V^{*}(s;w)$ is obtained through off-policy deep reinforcement learning. w is trainable parameters. We adopt two keypoints in DQN:

experience replay and soft parameter update. A dataset D is setup as experience container that stores a tuple (s_t, a_t, r_t, s'_t) at each time step t and the state value. The value for each experience state is calculated using a target network $V(s'; w^-)$ and w^- softly updated every C episodes. The loss function of deep V-network is

$$L(w) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} V(s'; w^-) - V(s; w))^2], \tag{11}$$

which is the expectation of square differential value on samples of experience $(s, a, r, s') \sim U(D)$, uniformly selected at random from the dataset D during learning.

3 DRL-Based Collision Avoidance

This section begins by describing the details about how to generate training dataset for pre-training the deep neural network utilizing ORCA algorithm [11]. Next, we elaborate the network architecture and training details about the deep reinforcement learning.

3.1 Dataset Generation

Different from the general reinforcement learning, off-policy reinforcement learning framework evaluates and updates different policies. So the training process can be accelerated using experience that produced by other collision avoidance algorithm. This work collects a sizable training data D from ORCA, which is a multi-agent collision avoidance simulator. The setup for ORCA can be seen as Fig. 3, in which a two-agent system is simulated where circle with an arrow represents robot and rectangle with an arrow represents pedestrian. Three common scenarios are setup. Crossing scenario: change the angle α from 0 to 2π. Overtaking scenario: change the vertical distance d. Passing scenario: change the vertical distance d.

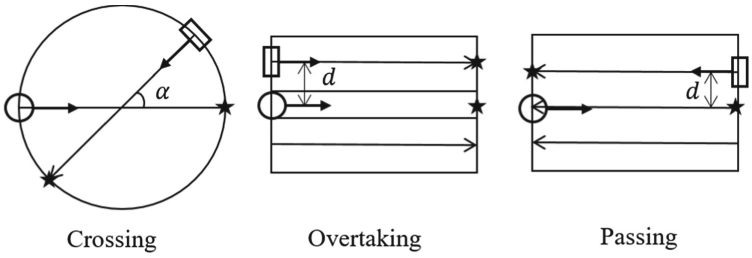

Fig. 3. Our setup for collecting dataset from ORCA simulator.

Trajectory is generated at first, and then processed into state-value pairs $D = \{(s_k, y_k)\}_{k=0}^{N}$, where $y = \gamma^{t_g \cdot v_{max}}$, t_g is the time to reach goal and v_{max} is

the maximum linear velocity for the robot. This work collects about 450 trajectories containing 10809 state-value pairs. Before feeding to the network, feature scaling is necessary to make each feature contribute equally to the state value. Otherwise, the pre-trained network make poor prediction and can not be used to explore environment efficiently.

3.2 Deep V-Network

The architecture of our Deep V-network is a multilayer perception with two hidden fully connected layers using ReLU activation function.

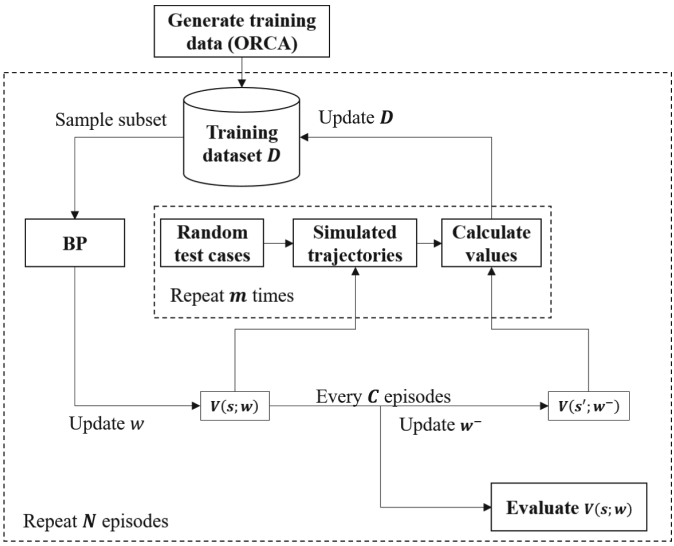

Fig. 4. The learning procedure of Deep V-Network.

Pre-training: To reduce agent randomly exploring time and accelerate the convergence of Deep V-network, the dataset D is used to pre-train the network, which can be regarded as initialization procedure of deep reinforcement learning. Stochastic gradient descent (SGD) algorithm is used to minimize loss function $L_i(w_i)$ as

$$L_i(w_i) = \frac{1}{N}\sum_{k=1}^{N}(y_k - V(s_k, w_i))^2, \tag{12}$$

where w_i is the parameters of Deep V-network at ith training iteration.

Here we make some notes about the pre-training process. First, the pre-training process essentially gives the robot a ability to explore environment more effectively, rather than explore randomly from scratch. And this process can

also reduce the complexity of designing reward function, because there is no need to use continues reward to guide the robot reaching its goal. Second, the value function is no need to be optimal approximation after pre-training. We get the optimal policy through deep reinforcement learning rather than supervised regression. Third, this process is not just a copy of ORCA algorithm, which actually learns a nonlinear function to evaluate the state value that represents the expected time to reach goal.

Deep Reinforcement Learning: The deep V-network after pre-training can guide the robot to accomplish navigation. In this section, reinforcement learning is used to refine the network by ϵ-greedy exploring environment (state space). The whole learning procedure can be seen as in Fig. 4.

In order to utilize the experience replay skill, the state generated by exploration during the training process is collected and stored into dataset D. To explore the environment, we first generate some random test cases consists of initial position, velocity, heading angle and goal position for robot and pedestrian. Then, $\epsilon-$greedy algorithm is iteratively used to choose current optimal action calculated using (10) or random action to guarantee explore enough space until reaching destination. To apply the soft parameter updating skill, the deep V-network $V(s;w)$ is duplicated as $V(s;w^-)$ after pre-training. $V(s;w^-)$ is used to process the exploring trajectories into state-value pairs and update the training data D by replacing old data. The parameters of $V(s;w^-)$ is updated every C episodes. The above procedures is repeated m times and update the parameters of $V(s;w)$ by back-propagation with a batch of training data, meanwhile the episode counter increases one. The whole training process has N episodes. To monitor the convergence of $V(s;w)$ network, some test cases are pre-defined and tested every C episodes.

4 Simulation and Results

In this section, simulation setup is detailedly introduced at first. Then the simulation results and analysis is shown. Finally, we discuss the advantages and disadvantages of the deep reinforcement learning based collision avoidance algorithm.

4.1 Simulation Setup

A simulator is designed using OpenAI gym [12] and implemented with Python on Nvidia Jetson TX2 developer kit with dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57 CPU, 256-core Pascal GPU and 8 GB of memory, which is shown in Fig. 5. The hidden layers of the deep value network consist of 100, 100 nodes respectively. Learning rate is set as 0.002 in pre training process and deep reinforcement leaning process. The training process lasts about 2 h when whole training episode N set as 500. Refer to Table 1 for more parameter setting details.

(a) (b)

Fig. 5. Nvidia Jetson TX2. (a) TX2 with a monitor; (b) TX2 developer kit.

Table 1. Deep reinforcement learning parameters setting

States dimension	15
Number of actions	35
Learning rate	0.002
Pre-training epoch	20
DRL epoch N	500
Discount factor γ	0.9
Initial ϵ	0.5
Exploring times m	10
Update period C	20

4.2 Simulation Results

The following presents the simulation results under different test scenarios as shown in Fig. 6, where circle and rectangle with a line represent robot and pedestrian respectively. The stars give the positions of their goals. Simple geometric shape is used, because it is hard to draw complicated models in OpenAI gym environment.

Swap: In this scenario, robot and pedestrian move in opposite directions and swap their position at last, as shown in Fig. 6(a). The figure shows that the robot passes right under the right-pass norm while keeping a safe margin with pedestrian.

Crossing: In this scenario, robot and pedestrian move to their goals respectively and their path will interact in the middle area, which is responsible for the most collision cases, as shown in Fig. 6(b). As can be seen that the robot slow down when it knows collision will happen if it still keep current velocity and turn a little bit right to keep a distance with the pedestrian. This test can prove that our algorithm can make collision avoidance effectively.

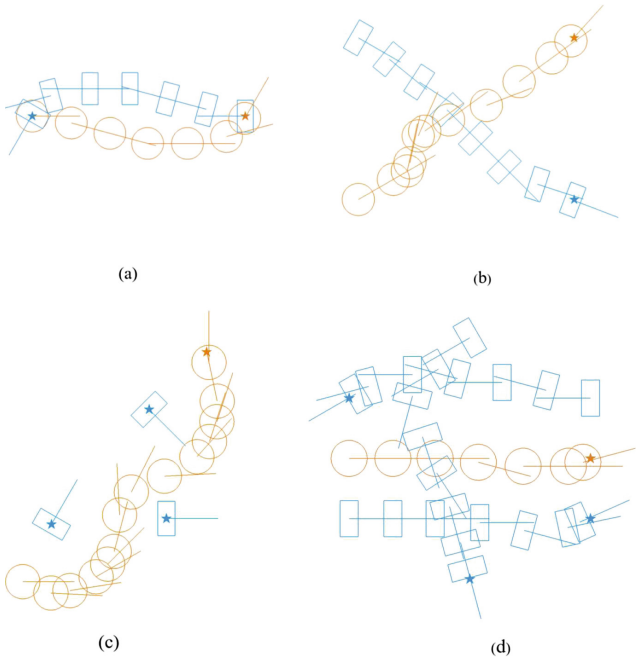

Fig. 6. Simulation results on four test scenarios. (a) Swap; (b) Crossing; (c) Static obstacle; (d) Multi-pedestrian.

Static Obstacle: In this scenario, pedestrians' current position is the same as their goals, which can be regarded as static obstacles, as shown in Fig. 6(c). As can be observed that the robot navigates around the obstacle smoothly. This test is used to demonstrate the generalization capability of our algorithm, because the deep V-network has never been trained on such scenarios.

Multi-pedestrian: To test robot navigates under multi-pedestrian environment, a three-pedestrians scenario is setup, in which one pedestrian passing the robot, one pedestrian running side by side to the sample direction with the robot and the last pedestrian navigating from up to down crosses with both robot and pedestrians, as shown in Fig. 6(d). The robot successfully finds a collision-free path and reaching its goal. The pedestrians are also guided by the same collision avoidance policy and reach their goals eventually.

4.3 Quantitative Analysis

To quantitatively evaluate our algorithm, here three performance metrics are defined:

(1) *safety margin*, the minimum distance between robot and pedestrians during traveling time. This metric denotes the collision avoidance performance, which is reflected in reward function.

(2) *right-pass obey proportion*, this can be measured by pre-defining some test cases and counting the proportion of the cases obeyed right-pass norm.
(3) *success rate*, successful reaching goal without collision. This metric can be used to estimate the collision avoidance effectiveness.

We set crossing, passing, overtaking and random test scenarios separately to estimate the above three metrics. And modified DWA like [16] is used to compare with our algorithm. The ideal safety margin should be τ_d meters, because collision avoidance reward function only cares about margin less τ_d, here set as 0.3.

Table 2. Quantitative analysis

Test scenarios configuration		Safety margin (m)		Right-pass rate		Success rate	
Scenarios	Times	DWA	Ours	DWA	Ours	DWA	Ours
Crossing	20	0.94	0.39	NA	15/20	13/20	18/20
Passing	20	1.00	0.25	NA	19/20	11/20	20/20
Overtaking	20	0.99	0.31	NA	14/20	14/20	19/20
Random	100	1.69	1.28	NA	81/100	87/100	93/100

As shown in Table 2, our algorithm achieved that right-pass norm obey proportion is 70% to 95% and success rate is 90% to 93% under all scenarios. While DWA has only 55% to 81% success rate and has no right-pass norm aware. Safety margin is about 0.3 m under crossing, passing and overtaking scenarios and is a little large under random test cases. The reason is randomly generated test cases are simple, requiring robot and pedestrian to travel strait towards their goals. DWA obviously has large safety margin. So compared to DWA, it can be concluded that the robot can successfully find a collision-free path that keeps a proper safety margin with pedestrians and also makes robot obey right-pass norm using our algorithm.

5 Conclusion and Discussion

We explored a deep reinforcement learning based collision avoidance algorithm for differential drive robot. By designing a set of reward and penalty items, the learned policy can guide a robot to navigate under pedestrian environment safely. Meanwhile, the robot is taught to obey right-pass norm that pedestrians always follow, which achieved a nature interaction between robot and human. The details of the architecture and training process of the deep V-network is presented. We demonstrate our algorithm is effective to navigate under pedestrian environment in simulation and perform better than DWA on safety margin, right-pass norm obey proportion and success rate.

Although the simulation results show the promising of learning-based collision avoidance method, some issues should be discussed to analyze the shortage of this work and guide the future work. It is hard to obtain the optimal policy. We repeat the learning procedure several times and the value function convergence to different target. The reason is that training data is not enough to represent the true state space. In the future, some exploring strategy during learning will be tried to get more broadly distributed samples like [10] does. The performance of the proposed algorithm on real robot need to be test. There are two ways to improve the deployment on real robot, considering state noise and using real robot to train. [15] uses real robot to do deep reinforcement learning and get a good result. In the future, we will try to compare the performance of the pure simulation model, model with noise estimation and real robot model.

References

1. Ferrer, G., Garrell, A., Sanfeliu, A.: Robot companion: a social-force based approach with human awareness-navigation in crowded environments. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 40, no. 6, pp. 1688–1694 (2013)
2. Dewantara, B.S.B., Miura, J.: Generation of a socially aware behavior of a guide robot using reinforcement learning. In: Electronics Symposium (2017)
3. Chao, W., Matveev, A.S., Savkin, A.V., Nguyen, T.N.: A collision avoidance strategy for safe autonomous navigation of an intelligent electric-powered wheelchair in dynamic uncertain environments with moving obstacles. In: 2013 European Control Conference, pp. 4382–4387 (2013)
4. Mehta, D., Ferrer, G., Olson, E.: Autonomous navigation in dynamic social environments using Multi-Policy Decision Making. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1190–1197 (2016)
5. Helbingp, D., Molnor, P.: Social force model for pedestrian dynamics. Phys. Rev. E **51**(5), 4282 (1995)
6. Ferrer, G., Garrell, A., Sanfeliu, A.: Social-aware robot navigation in urban environments. In: 2013 European Conference on Mobile Robots (ECMR), pp. 331–336 (2013)
7. Pinxin, L., Wenxi, L., Jia, P.: Deep-learned collision avoidance policy for distributed multiagent navigation. IEEE Robot. Autom. Lett. **2**(2), 656–663 (2017)
8. Kretzschmar, H., Spies, M., Sprunk, C., Burgard, W.: Socially compliant mobile robot navigation via inverse reinforcement learning. Int. J. Robot. Res. (2016)
9. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
10. Hwangbo, J., Sa, I., Siegwart, R., Hutter, M.: Control of a quadrotor with reinforcement learning. IEEE Robot. Autom. Lett. **PP**(99), 1 (2017)
11. Van den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. In: Pradalier, C., Siegwart, R., Hirzinger, G. (eds.) Robotics Research. Springer Tracts in Advanced Robotics, vol. 70, pp. 3–19. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19457-3_1
12. Brockman, G., et al.: OpenAI gym (2016)

13. ROS navigation package. http://wiki.ros.org/navigation
14. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robot. Autom. Mag. **4**(1), 23–33 (1997)
15. Kahn, G., et al.: Uncertainty-aware reinforcement learning for collision avoidance. arXiv preprint arXiv:1702.01182 (2017)
16. Chen, Y.: Service robot navigation in large dynamic and complex indoor environments. Diss. University of Science and Technology of China (2017)