Journal of Intelligent Manufacturing (2023) 34:151–180
https://doi.org/10.1007/s10845-022-01988-z

Attention-based advantage actor-critic algorithm with prioritized experience replay for complex 2-D robotic motion planning

Chengmin Zhou[1,2] · **Bingding Huang**[2] · **Haseeb Hassan**[2] · **Pasi Fränti**[1,3]

Received: 13 October 2021 / Accepted: 1 July 2022 / Published online: 7 August 2022

Abstract

Robotic motion planning in dense and dynamic indoor scenarios constantly challenges the researchers because of the motion unpredictability of obstacles. Recent progress in reinforcement learning enables robots to better cope with the dense and unpredictable obstacles by encoding complex features of the robot and obstacles into the encoders like the *long-short term memory* (LSTM). Then these features are learned by the robot using reinforcement learning algorithms, such as the deep Q network and asynchronous advantage actor critic algorithm. However, existing methods depend heavily on expert experiences to enhance the convergence speed of the networks by initializing them via imitation learning. Moreover, those approaches based on LSTM to encode the obstacle features are not always efficient and robust enough, therefore sometimes causing the network overfitting in training. This paper focuses on the advantage actor critic algorithm and introduces an *attention-based actor critic algorithm with experience replay algorithm* to improve the performance of existing algorithm from two perspectives. First, LSTM encoder is replaced by a robust encoder *attention weight* to better interpret the complex features of the robot and obstacles. Second, the robot learns from its past prioritized experiences to initialize the networks of the advantage actor-critic algorithm. This is achieved by applying the *prioritized experience replay* method, which makes the best of past useful experiences to improve the convergence speed. As results, the network based on our algorithm takes only around 15% and 30% experiences to get rid of the early-stage training without the expert experiences in cases with five and ten obstacles, respectively. Then it converges faster to a better reward with less experiences (near 45% and 65% of experiences in cases with ten and five obstacles respectively) when comparing with the baseline LSTM-based advantage actor critic algorithm. Our source code is freely available at the GitHub (https://github.com/CHUENGMINCHOU/AW-PER-A2C).

Keywords Motion planning · Path planning · Reinforcement learning · Intelligent robot · Deep learning

Abbreviations

A2C	Advantage actor critic
A3C	Asynchronous advantage actor-critic
AW	Attention weight
CNN	Convolutional neural network
DDPG	Deep deterministic policy gradient
DL	Deep learning
DPG	Deterministic policy gradient
DQN	Deep Q learning
DWA	Dynamic window approach
IID	Independently identical distribution
IL	Imitation learning
LSTM	Long-short term memory
MDP	Markov decision process
MLP	Multiple layer perceptron
ORCA	Optimal reciprocal collision avoidance
PER	Prioritized experience replay
PPO	Proximal policy optimization
RL	Reinforcement learning
RRT	Rapidly-exploring random tree
RG	Relation graph
TD	Temporal difference

✉ Bingding Huang
huangbingding@sztu.edu.cn

✉ Pasi Fränti
franti@cs.uef.fi

[1] School of Computing, University of Eastern Finland, Joensuu, Finland

[2] College of Big Data and Internet, Shenzhen Technology University, Shenzhen, China

[3] Machine Learning Group, School of Computing, University of Eastern Finland, Joensuu, Finland

TRPO Trust region policy optimization

Introduction

Service robots, especially the indoor service robots, appeared in scenarios like airports, restaurants, and train stations nowadays to provide simple services to visitors. For instance, luggage delivery, food delivery, and direction consulting. However, these robots suffer poor motion planning performance in scenarios with dense and dynamic obstacles (pedestrians) because of the motion unpredictability of these obstacles. This barricades the further commercial use of these service robots. Motions of some robots are controlled by *classical path planning algorithms* like the graph search algorithm [e.g., A* (Hart et al., 1968)], sample-based algorithm [e.g., the *rapidly-exploring random tree* (RRT) (Bry & Roy, 2011)], and interpolating curve algorithms (Farouki & Sakkalis, 1994; Funke et al., 2012; González et al., 2014; Reeds & Shepp, 1990; Xu et al., 2012). These algorithms work well in static environments or low-speed scenarios with less obstacles. However, they make robots suffer more collisions in cases with dense and dynamic obstacles because these algorithms generate the motions or paths in an online way. Online motion generation depends on the update of environmental maps that require much computing resources. *Reaction-based algorithms* like the *dynamic window approach* (DWA) (Fox et al., 1997) and *optimal reciprocal collision avoidance* (ORCA) (Van Berg et al., 2008) perform fast to handle the obstacle's unpredictability. This enables the robot to better avoid the slow-speed obstacles. However, these algorithms still require the online updates of environmental information that should not be ignored in cases with dense and dynamic obstacles.

Deep learning (DL) algorithms generate the robotic motions by performing a trained model, in which the time consumed is short and it can be ignored. *Classical DL* like the *convolutional neural network* (CNN) (Bai et al., 2019) can generate instant motions to dynamic obstacles. These motions are one-step predictions which do not consider task goals, therefore obtaining suboptimal solutions or trajectories eventually. Recent progress in the deep *reinforcement learning* (RL) like the optimal value RL [e.g., Dueling *deep Q network* (DQN) (Wang et al., 2016)] and policy gradient RL [e.g., *asynchronous advantage actor critic algorithm* (A3C) (Mnih et al., 2016)] enable the robot to consider the task goal and instant obstacle avoidance simultaneously. These algorithms obtain near-optimal solutions to better cope with cases with dense and dynamic obstacles. However, RL algorithms face many challenges, like overfitting and slow convergence speed caused by high bias or variance. It is still not enough to obtain a desired performance of motion planning by improving the RL algorithms merely. The performance of robotic motion planning can be further improved by improving other factors, such as the input quality.

Input quality in this paper is defined as *the efficacy of data* and *the efficiency to make the best of useful data*. This means: (1) The input data of the algorithm should fully represent the environmental information or environmental state (e.g., the relationship of obstacles, the speed, radius, moving direction, and position of obstacles). (2) The algorithm should first select the high-quality input data (e.g., the trajectory in which the robot reaches the goal), and then makes the best of it by the data replay and replay strategies. The efficacy of data denotes *how good the data represents the environmental information*. High data efficacy depends on suitable methods to describe the environmental information. This provides qualified or comprehensive data without noise for RL algorithms to learn from.

Some works use source data from the environment, like the source images (Bai et al., 2019), as the input of algorithms directly. This type of data includes much noise (e.g., background information), hence may causing the overfitting of algorithms. Some works use the methods (e.g., LSTM) that partially interpret the environmental information (Everett et al., 2018) to generate dataset. Algorithms based on these methods cannot fully learn the core or needed information from dataset and then the overfitting or slow convergence speed follows. High efficiency to make the best of data relies on the methods to reuse the dataset generated by performing the algorithm itself on the robot. Recent RL algorithms based on the experience replay (Wang et al., 2017) and the *prioritized experience replay* (PER) (Schaul et al., 2016) make it possible to better make the best of dataset, therefore realizing decent performances in many continuous-control problems. These methods are data-efficient and independent than the online RL [e.g., online A3C(Mnih et al., 2016)] and the *imitation learning* (IL) based method (Chen et al., 2019b). Online RL is the data guzzler which discards the data after the training. IL based method heavily depends on other algorithms or artificial data to generate expert experiences (e.g., trajectories generated via ORCA, and artificial trajectories planned by human) to initialize the RL algorithms.

To cope with shortcomings mentioned above, this paper first focuses on the *advantage actor critic algorithm* (A2C) which is a robust policy gradient RL algorithm to cope with the sequential-decision robotic motion planning problems. Then the performance of robotic motion planning is improved by enhancing the input quality (efficacy of data, and efficiency to make the best of useful data). The main contributions of this paper include:

(1) The improvement in efficacy of data: Improving the state-of-art *long-short term memory* (LSTM) based actor critic algorithm by applying the *attention weight*

(AW) encoder to replace the LSTM encoder for the interpretation of environmental information.

(2) Combination of online and offline A2C with PER: Fitting the PER into combined online and offline A2C to fast improve its convergence speed. Therefore AW-based A2C converges steeply to a better reward without expert experiences from other methods. To our knowledge, our method is the first to specially focus on two aspects of input quality to better improve the robotics motion planning in dense and dynamic scenarios. Other works are the data guzzler, or they depend heavily on expert experiences.

The main contents of this paper include following four sections: Sect. 2 is the research background that describes related works, preliminary of RL, and problem formulation of complex robotic motion planning; Sect. 3 is the research methods that consist of the principle to design the RL network, combination of online/offline A2C, LSTM/AW encoders, AW-PER-based A2C and its training strategies; Sect. 4 is the experiment results that include the design of network architecture, model trainings, and model evaluations; Sect. 5 analyzes the problems found in the training and their solutions, followed by the future research directions.

Research background

Research background includes three parts that are the preliminary of RL, related works and problem formulation. Some preliminary of RL are listed to provide basic understanding or definition of RL-related terms. Related works focus on recent progresses of RL algorithms in solving continuous-control problems. These progresses are elaborated from two different levels: *algorithm level*, and *input quality level*. Algorithm level here denotes the state-of-art RL algorithms to solve the time-sequential problems in games. Input quality level here includes recent algorithms to interpret and reuse the data for improving the data efficacy and efficiency to make the best of it.

Preliminary of RL

Markov decision process

The *Markov decision process* (MDP) is the sequential decision process based on Markov Chain (Bas, 2019) which is a variable set $\boldsymbol{X} = \{X_n : n > 0\}$ and $p(X_{t+1}|X_t, \ldots, X_1) = p(X_{t+1}|X_t)$. This means the state and action of the next step only depend on the state and action of the current step. MDP is described as a tuple $< S, A, P, R >$. State S: S denotes the state and here it refers to the state of robot and obstacles. Action A: A denotes an action taken by the robot. Reward R: R denotes the reward or punishment received by the robot after executing actions. State transition probability P: P denotes the possibility to transit from one state to the next state.

Value function

The values denote *how good one state is* or *how good one action is in one state*, and they are called the *state value* (V value) and *state-action value* (Q value) respectively. Values are defined as the expectation of accumulative rewards $V(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+1} + \cdots + \gamma^{T-1} R_T | s_t\right]$ or $Q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+1} + \cdots + \gamma^{T-1} R_T | (s_t, a_t)\right]$ where γ is a discounted factor. The value function in deep RL scope is represented by neural networks to estimate the value of environmental state via the function approximation (Baird, 1995).

Policy function

The policy denotes the way to select actions. Policy function is also represented by neural networks in deep RL, and actions are decided by either indirect way (e.g., $a \leftarrow argmax_a R(s, a) + Q(s, a; \theta)$ in DQN (Mnih et al., 2013; Mnih et al., 2015)) or direct way [e.g., $\pi_\theta : s \rightarrow a$ in actor-critic algorithm (Konda & Tsitsiklis, 2000)].

Related works

Algorithm level

RL algorithms basically include the optimal value RL and policy gradient RL. They are primarily tested in games like Atari game, and their representatives are DQN (Mnih et al., 2013) and actor-critic algorithm (Konda & Tsitsiklis, 2000). Then, these two algorithms are continuously improved from different perspectives and many variants follow. DQN evolves into double DQN (Van Hasselt et al., 2016) and dueling DQN (Wang et al., 2016), while actor-critic algorithm basically evolves from three perspectives: *multi-thread policy improvement*, *deterministic policy improvement*, and *monotonous policy improvement*. Variants from multi-thread policy improvement focus on making the best of the multiply thread method and policy entropy to accelerate the convergence speed, and typical examples are A3C and A2C (Mnih et al., 2016). Deterministic policy improvement proves that the policy to select actions is stable in one state s and actions can be directly decided by this state $a \leftarrow \mu_\theta(s)$, while its counterpart the stochastic policy selects actions by the possibility $a \leftarrow \pi_\theta(a|s)$. Typical examples of variant based on the deterministic policy improvement are the *deterministic policy gradient* (DPG) (Silver et al., 2014) and the *deep deterministic policy gradient* DDPG (Munos et al., 2016). Monotonous policy improvement introduces the trust region constraint, surrogate, and adaptive penalty to ensure

the monotonous update of policy. Typical instances are the *trust region policy optimization* (TRPO) (Schulman et al., 2015) and the *proximal policy optimization* (PPO) (Schulman et al., 2017).

Input quality level

(1) Data interpretation: early-stage RL research makes use of the CNN (Bai et al., 2019) to preprocess the source images for extracting the features. These methods seem like the "black box", because it is hard to know what these CNN-based RL algorithms learnt from the source images. Source images also include much noise which probably causes the overfitting of network. Then, the feature of the agents in the environment (e.g., robots, obstacles, or pedestrians) are specially defined into clear forms (e.g., vector or tensor) according to the requirements of the motion planning task, and these features are encoded accordingly to form an integrated description or interpretation of the environmental information for RL algorithms to learn. Representatives of encoding methods include LSTM (Everett et al., 2018; Inoue et al., 2019), AW (Chen et al., 2019b; Lin et al., 2017) and the *relation graph* (RG) (Chen et al., 2019a). They are all robust methods, but performance of motion planning based on these encoding methods varies, because it not only depends on the robustness of these methods but also relies on how to use these methods. For example, LSTM (Everett et al., 2018) just encodes partial environmental information therefore RL algorithms also partially learn the needed information required by motion planning tasks, hence may causing the overfitting and suboptimal solutions in the training. (2) Experience replay: online RL is a data guzzler therefore many researchers turn to offline learning or batch learning for better making the best of limited dataset to improve the convergence speed. A milestone work is the DQN (Mnih et al., 2013) which stochastically samples and learns dataset stored in the memory. However, stochastic sampling method ignores the importance or priority of data which is the essential part for further improvement of convergence, hence greedy sampling method (Schaul et al., 2016) is introduced to fast improve the convergence but the results are always suboptimal. PER (Schaul et al., 2016) solves this problem by finding a better trade-off between the stochastic sampling and greedy sampling. Following works in experience replay are about the justification of PER from mathematical perspectives (Li et al., 2021). There are also some variants (Jiang et al., 2020; Zha et al., 2019) but their improvements in convergence are limited.

Problem formulation

Recent work (Van Berg et al., 2008) introduced a competent simulative environment (Fig. 1) that includes dynamic robot and obstacles in a fix-size 2D indoor area. The robot and obstacles move towards their own goals simultaneously and avoid collisions to each other. They obey the same or different policies for motion planning to avoid collisions. This simulative environment creates *circle-crossing* and *square-crossing* scenarios that add *predictable complexity* to the environment. It is therefore a good platform to evaluate algorithms adopted by the robot or obstacles.

The robot and obstacles plan motions towards their goals and avoid collisions by sequential decision making. Let s represents the state of the robot. Let a and v represent action and velocity of the robot, and $a = v = \left[v_x, v_y\right]$. Let $p = \left[p_x, p_y\right]$ represents position of the robot. Let s_t represents state of the robot at time step t. s_t is composed by observable and hidden parts $s_t = \left[s_t^{obs}, s_t^h\right]$, $s_t \in R^9$. Observable part refers to factors of state that can be measured by others. It is composed by the position, velocity, and radius $s^{obs} = \left[p_x, p_y, v_x, v_y, r\right]$, $s^{obs} \in R^5$. Hidden part refers to factors of state that cannot be seen by others. It is composed by the planned goal position, preferred speed and heading angle $s^h = \left[p_{gx}, p_{gy}, v_{pref}, \theta\right]$, $s^h \in R^4$. The state, position, and radius of obstacles are described by $\widehat{s}$, $\widehat{p}$ and $\widehat{r}$.

To analyze this decision-making process, we first introduce the one-robot one-obstacle case. Then extend it to the one-robot multi-obstacle case. The robot plans its motion by obeying policy $\pi : \left(s_{0:t}, \widehat{s}_{0:t}^{obs}\right) \rightarrow a_t$ while obstacles obey $\widehat{\pi} : \left(\widehat{s}_{0:t}, s_{0:t}^{obs}\right) \rightarrow a_t$. The objective of the robot is to minimize the time to its goal $E\left[t_g\right]$ (Eq. 1) under the policy π without collisions to obstacles. Constraints of robot's motion planning in this sequential decision problem can be formulated via Eqs. 2–5 that represent the *collision avoidance constraint*, *goal constraint*, *kinematics of the robot* and *kinematics of obstacle*, respectively. The collision avoidance constraint denotes that the distance of the robot and obstacles $\|p_t - \widehat{p}_t\|_2$ should be greater than or equal to the radius sum of the robot and obstacles $r + \widehat{r}$. The goal constraint denotes that the position of the robot p_{tg} should be equal to the goal position p_g if the robot reaches the goal. Kinematics of the robot denotes that the position of the robot in time step t p_t is equal to the sum of the robot position in time step t-1 p_{t-1}. The change of the robot position $\Delta t \cdot \pi : \left(s_{0:t}, \widehat{s}_{0:t}^{obs}\right)$ where $\pi : \left(s_{0:t}, \widehat{s}_{0:t}^{obs}\right)$ is a velocity decided by the policy π. Kinematics of obstacles is the same as that of the robot.

$$minimize\, E\left[t_g \vee s_0, \widehat{s}_0^{obs}, \pi, \widehat{\pi}\right] \tag{1}$$

$$s.t. \|p_t - \widehat{p}_t\|_2 \geq r + \widehat{r} \forall t \tag{2}$$

$$p_{tg} = p_g \tag{3}$$

$$p_t = p_{t-1} + \Delta t \cdot \pi : \left(s_{0:t}, \widehat{s}_{0:t}^{obs}\right) \tag{4}$$

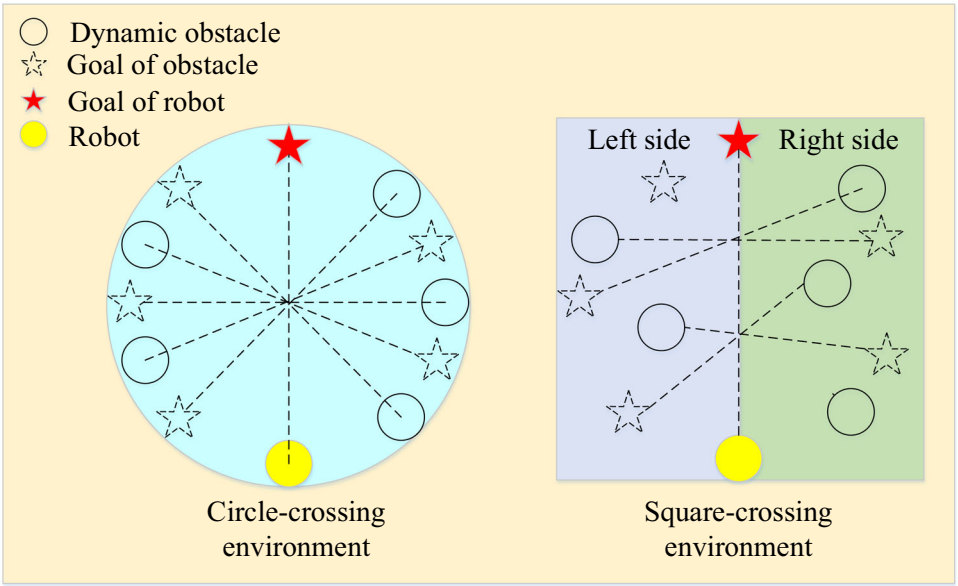


Fig. 1 Circle-crossing (left) and square-crossing simulators (right). Obstacles are randomly generated near the brink of the circle in the circle-crossing environment. Then they move towards their opposite sides. In the square-crossing environment, obstacles are randomly generated on the left side or right side and then they move towards any positions on their opposite side

$$\widehat{p}_t = \widehat{p}_{t-1} + \Delta t \cdot \widehat{\pi} : \left(\widehat{s}_{0:t}, s_{0:t}^{obs}\right) \tag{5}$$

Constraints of one-robot one-obstacle case can be easily extended into one-robot N-obstacle case where the objective (Eq. 1) is replaced by the $minimize E\left[t_g | s_0, \left\{\widehat{s}_0^{obs} \ldots \widehat{s}_N^{obs}\right\}, \pi, \widehat{\pi}\right]$ where we assume that obstacles use the same policy $\widehat{\pi}$. Collision avoidance constraint (Eq. 2) is replaced by

$$\begin{cases} \|p_t - \widehat{p}_{0:t}\|_2 \geq r + \widehat{r} \\ \|p_t - \widehat{p}_{1:t}\|_2 \geq r + \widehat{r} \\ \quad \ldots \\ \|p_t - \widehat{p}_{N-1:t}\|_2 \geq r + \widehat{r} \end{cases} \forall t \tag{6}$$

assuming that obstacles are in the same radius $\widehat{r}$. $\widehat{p}_{N-1:t}$ denotes the position of the N-th obstacle in time step t. Kinematics of the robot is replaced by $p_t = p_{t-1} + \Delta t \cdot \pi : \left(s_{0:t}, \{\widehat{s}_{0:t}^{obs} \ldots \widehat{s}_{N-1:t}^{obs}\}\right)$. Kinematics of obstacles is replaced by

$$\begin{cases} \widehat{p}_{0:t} = \widehat{p}_{0:t-1} + \Delta t \cdot \widehat{\pi} : \left(\widehat{s}_{0:t}, s_{0:t}^{obs}\right) \\ \widehat{p}_{1:t} = \widehat{p}_{1:t-1} + \Delta t \cdot \widehat{\pi} : \left(\widehat{s}_{1:t}, s_{0:t}^{obs}\right) \\ \quad \ldots \\ \widehat{p}_{N-1:t} = \widehat{p}_{N-1:t-1} + \Delta t \cdot \widehat{\pi} : \left(\widehat{s}_{N-1:t}, s_{0:t}^{obs}\right) \end{cases} . \tag{7}$$

Methods

This section first describes the principles to design RL network. These principles are about the strategies to configure the number of layer and node in each layer (Sect. 3.1). Then, principles of combining the online A2C and offline A2C with PER are given (Sect. 3.2). It is followed by the principles of the LSTM encoder and AW encoder (Sect. 3.3). The last, AW-based A2C can be further improved by our algorithm AW-PER-based A2C while some training strategies are also presented (Sect. 3.4).

Our contributions are the AW-based A2C and AW-PER-based A2C (Sect. 3.4). Note that AW-based A2C is just a part of the AW-PER-based A2C. Hence, we just present the pseudocode of AW-PER-based A2C (Algorithms 1–4). AW-based A2C is the optimized version of LSTM-based A2C by replacing the LSTM encoder with AW encoder. AW-PER-based A2C is the optimized version of AW-based A2C by combining the online and offline learning (batch learning) to update the network. Architectures of AW-based A2C and AW-PER-based A2C are shown in Fig. 2, and their workflows are almost the same:

(1) These two algorithms collect the same source environment information (the robot and obstacles) which is encoded by the attention encoder.
(2) The encoded environment information is then fed to A2C algorithm to update the network (model). However, the network of AW-based A2C is just updated by online A2C, while that of AW-PER-based A2C is updated by online and offline A2C simultaneously.
(3) The network configurations of these two algorithms are always suboptimal. Hence, the number of layer and node of each layer need to be configured to obtain the optimal network configurations.

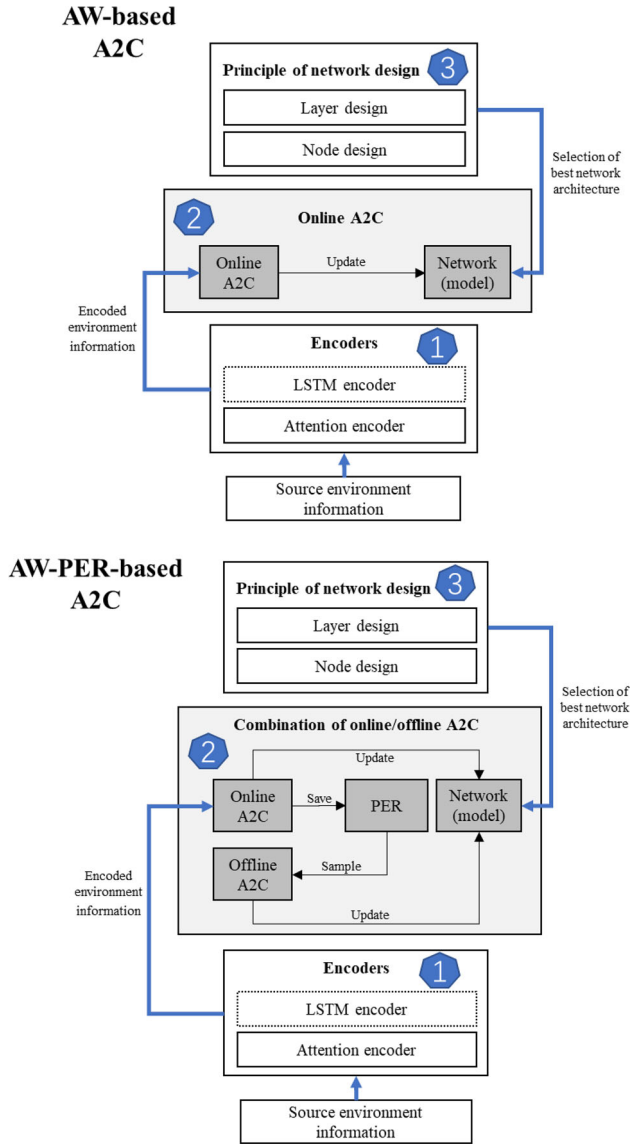


Fig. 2 Architectures of our proposed algorithms: AW-based A2C and AW-PER-based A2C

Principles to design networks for RL

Principles of configuration for number of layer and node

The network with *one layer* is only used for simple linear problem (e.g., two-class classification). A *multiple layer perceptron* (MLP) with *one hidden layer* can approximate any function that we require (Hornik et al., 1989). A feedforward network can approximate any Borel measurable function. This network should be with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g., the logistic sigmoid activation function). The feedforward network approximates from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units (Goodfellow et al., 2016). Artificial neural network (e.g., MLP) with *two hidden layers* is sufficient for creating any classification regions of desired shape (Lippmann, 1988). Although a single hidden layer is optimal for some functions, there are others for which a single-hidden-layer-solution is very inefficient compared to solutions with *two or more hidden layers* (Reed & MarksII, 1999).

However, the question of "how many nodes should be used in each layer" had not been solved yet. Currently, the selection of number in the layer and node is more art than science. This means how to configure layers and nodes is more likely to consider the *empirical findings* in literature or *intuitions* from experience (Goodfellow et al., 2016).

Strategies in configuration of layer and node

It is an efficient starting point to find better solutions for the configuration of layers and nodes, but it still requires robust test harnesses and controlled experiments that include some basic strategies (Brownlee, 2018):

(1) *Random test*: random configurations for the number of layers and the number of nodes in each layer.
(2) *Grid test*: systematic search for the number of layers and the number of nodes in each layer.
(3) *Heuristic test*: directed search for the number of layers and the number of nodes in each layer according to search algorithms [e.g., genetic algorithm (Stathakis, 2009)].
(4) *Exhaustive test*: all combinations of the number of layers and the number of nodes in each layer, but this strategy is feasible for simple network or dataset.

Combination of online A2C and offline A2C with PER

Online A2C

A2C is a variant of A3C (Mnih et al., 2016), and they share the same objectives. Their difference is that A3C collects experience and updates the wight of network individually and asynchronously in each thread. However, A2C collects experience only in each thread and the weight of network is updated synchronously. The objective (loss function) of A2C is defined as the expectation of losses in the policy function and value function:

$$L^{online} = \mathbb{E}_{s,a\sim\pi_\theta}(L^{online}_{policy} + \beta^{online} L^{online}_{value}) \quad (8)$$

where β^{online} is a discounted factor. Losses of policy function and value function are defined by

$$L_{policy}^{online} = -\log\pi_\theta(a_t|s_t)\big(V_t^n - V_\theta(s_t)\big) - \alpha\mathcal{H}_t^{\pi_\theta} \tag{9}$$

$$L_{value}^{online} = \frac{1}{2}\|V_\theta(s_t) - V_t^n\|^2 \tag{10}$$

where $\mathcal{H}_t^{\pi_\theta} = -\sum_a \pi(a|s_t)\log\pi(a|s_t)$ is the policy entropy to encourage the exploration for finding better potential action, and α is a discounted factor. $V_t^n = \gamma^n V_\theta(s_{t+n}) + \sum_{m=0}^{n-1}\gamma^m r_{t+m}$ is the n-step discounted accumulative state value to represent how good the state s_t is. In our experiment, the data is the episodic experience therefore the n is defined as the number of steps in each episode.

Offline A2C and the combination of online/offline A2C

(Oh et al., 2018) is inspired by PER (Schaul et al., 2016) and proposes an offline actor-critic loss, hence PER can be easily integrated into the actor-critic based algorithms. The offline actor-critic loss is defined by

$$L^{offline} = \mathbb{E}_{s,a\sim\pi_\theta}(L_{policy}^{offline} + \beta^{offline}L_{value}^{offline}) \tag{11}$$

where $\beta^{offline}$ is a discounted factor. $L_{policy}^{offline}$ and $L_{value}^{offline}$ are defined accordingly by

$$L_{policy}^{offline} = -\log\pi_\theta(a_t|s_t)(R_t - V_\theta(s_t))_+ \tag{12}$$

$$L_{value}^{offline} = \frac{1}{2}\|(R_t - V_\theta(s_t))_+\|^2 \tag{13}$$

where $R_t = \sum_k^\infty \gamma^{k-t} r_k$ is the *Monte-Carlo return* instead of the accumulative return V_t^n, and $(\cdot)_+ = \max(\cdot, 0)$.

To make the best of offline A2C to improve the convergence speed of online A2C, two problems should be solved: (1) *selection of useful data*; (2) *different way of weight update* caused by different data distribution in the online learning and batch learning. Online learning updates its weight in a stochastic way. Its dataset is unnecessary to fit any distribution therefore the way of weight update is unbiased. Dataset in the batch learning, however, is expected to meet the *independently identical distribution* (IID). Therefore, weight update in batch learning mismatches that in the online learning, hence introducing the bias if online learning and batch learning are simply combined.

The PER better solves these two problems by setting the *priority* of data and applying the *importance sampling weigh* in the batch learning. The priority is defined by the *temporal difference* (TD)-error δ:

$$p_i = |\delta| + \varepsilon \tag{14}$$

where ε is a small positive constant, while the priority in the offline A2C is set to be the lower bound of TD-error:

$$p_i = (\delta)_+,\ \delta = R - V_\theta(s). \tag{15}$$

Hence, the probability to sample the experience is defined by

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{16}$$

where α is the discount of priority, k the transitions in one episode and $i \epsilon k$. The bias caused by batch learning is reduced by the importance sampling weight that is defined by

$$w_i = \frac{(N \cdot P(i))^{-\beta}}{max_i w_i} \tag{17}$$

where $\frac{1}{max_i w_i}$ is a weight for normalization to stabilize the weight update, N the size of samples and $\beta \in [0, 1]$ is a compensation of $P(i)$. Hence the way of weight update in the offline/batch learning turns into

$$\Delta \leftarrow \Delta + w_i \cdot \nabla_\theta(L_{policy}^{offline} + \beta^{offline}L_{value}^{offline}) \tag{18}$$

LSTM and AW encoders

LSTM encoder uses LSTM to encode the obstacles near the robot, hence forming the description of environmental state S^{lstm} which includes the features of the robot and obstacles

$$S^{lstm} = [s_r,\ S_o^{lstm}] \tag{19}$$

where S_o^{lstm} denotes the features of the obstacle in the environment. Obstacles are encoded by LSTM according to their distances (Everett et al., 2018) to the robot:

$$S_o^{lstm} = LSTM\Big([e_{d_{min}}^{lstm}, \ldots, e_{d_{max}}^{lstm}]\Big),\ [e_{d_{min}}^{lstm}, \ldots, e_{d_{max}}^{lstm}] \leftarrow rank(d_i),\ i \in N \tag{20}$$

where N denotes the number of obstacles. d_i denotes the distance of obstacle o_i to the robot. $e_{d_{min}}^{lstm}$ and $e_{d_{max}}^{lstm}$ denote the *pairwise features* of the obstacles with shortest and largest distances to the robot. The pairwise feature is defined as the combined feature of the robot and one of the obstacles:

$$e_i^{lstm} = [s_r,\ o_i] \tag{21}$$

AW encoder is based on the attention weight (Chen et al., 2019b; Lin et al., 2017). It also defines the environmental

state as the feature combination of the robot and obstacles:

$$S^{aw} = [s_r,\ S_o^{aw}] \tag{22}$$

where S_o^{aw} denotes the features of the obstacle encoded by AW, and it is defined by

$$S_o^{aw} = \sum\nolimits_{i=1}^{n} [softmax(\alpha_i)] \cdot h_i \tag{23}$$

where α_i and h_i denote the *attention score* and the *interaction feature* of the robot and obstacle o_i respectively. The interaction feature is defined as

$$h_i = f_h(e_i;\ w_h) \tag{24}$$

where $f_h(\cdot)$ and w_h denote the neural network and its weight. e_i denotes the *embedded feature* of the robot and obstacle o_i. The attention score is defined by

$$\alpha_i = f_\alpha(e_i,\ e_{mean};\ w_a) \tag{25}$$

where $f_\alpha(\cdot)$ and w_a denote the neural network and its weight. e_{mean} denotes the mean of all embedded features. The embedded feature and e_{mean} are defined by

$$e_i = f_e(s_r,\ o_i,\ M_i;\ w_e),\ i \in N \tag{26}$$

$$e_{mean} = \frac{1}{n} \sum\nolimits_{i=1}^{N} e_i \tag{27}$$

where $f_e(\cdot)$ and w_e denote the neural network and its weigh. M_i denotes the occupancy map of obstacle o_i and it is defined by

$$\begin{aligned} M_i(a,\ b,\ :) &= \sum\nolimits_{j \in \mathcal{N}_i} \delta_{ab}[x_j - x_i,\ y_j - y_i] \cdot w_j^{'}, \\ w_j^{'} &= (v_{xj},\ v_{yj},\ 1) \end{aligned} \tag{28}$$

where $w_j^{'}$ is a local state vector of obstacle o_j. $\mathcal{N}_i$ denotes other obstacles near the obstacle o_i. The *indicator function* $\delta_{ab}\left[x_j - x_i,\ y_j - y_i\right] = 1$ if $(x_j - x_i,\ y_j - y_i) \in (a,\ b)$ where $(a,\ b)$ is a two-dimension cell.

AW-PER-based A2C and its training strategies

This part elaborates our proposed algorithm. It features the new encoder (AW) and combined online and offline (batch) learning approach. We first recall and clarify the definition of environmental description, and then the proposed algorithm is given.

The definition of environmental description (environmental state)

Before introducing our algorithm, it is necessary to clarify all sorts of descriptions of state. Let's first recall the definitions of state in the section of Problem Formulation:

$$\begin{cases} s_{agent} = \left[s^{obs},\ s^h\right],\ s \in R^9 \\ s^{obs} = \left[p_x,\ p_y,\ v_x,\ v_y,\ r\right],\ s^o \in R^5 \\ s^h = \left[p_{gx},\ p_{gy},\ v_{pref},\ \theta\right],\ s^h \in R^4 \end{cases} \tag{29}$$

where s_{agent} denotes the state of agent (obstacles or robot). It consists of its observable state s^{obs} and hidden state s^h. Hence, *the source state of the robot in the environment* is descripted by

$$s_{robot}^{source} = [s_{robot},\ \left\{\widehat{s}_0^{obs} \ldots \widehat{s}_N^{obs}\right\}] \tag{30}$$

where $\widehat{s}_N^{obs}$ denotes the observable state of N-th obstacles. However, the source state cannot be directly used for the training therefore it is replaced by *the robot-centric states* which is transformed from the source state of the robot. The robot-centric states are defined by

$$s_r = \left[d_g,\ v_{pref},\ \theta,\ r,\ v_x,\ v_y\right],\ s_r \in R^6 \tag{31}$$

$$o_i = \left[p_x,\ p_y,\ v_{xi},\ v_{yi},\ r_i,\ d_i,\ r_i + r\right],\ o_i \in R^7 \tag{32}$$

where s_r denotes new state of the robot while o_i denotes robot-centric observable state of i-th obstacle. Note that i-th denotes the obstacle's order which is generated randomly in the simulator when one episode of the experiment starts. In the definition of robot-centric observable state o_i, the $r_i + r$ denotes the collision constraint of each obstacle to the robot. The collision constraint varies among obstacles. To some degree it represents how "danger" the obstacle is. The robot is easy to learn how to keep a safe distance to each obstacle with its collision constraint. Otherwise, more data is required for the robot to learn the *safe distance strategy* in the trial and error. Finally, the state of the robot in the environment (*environmental description*) from training is defined by

$$s = [s_r,\ \{o_0 \ldots o_N\}] \tag{33}$$

Note that o_i in $\{o_0 \ldots o_N\}$ is the source description of the obstacles. It is not encoded by any encoders.

AW-PER-based A2C and its training strategies

Our algorithm features the improvements in the description of the environmental state and the convergence speed by applying the AW encoder and PER to A2C algorithm.

Hence, the robot in the dense and dynamic scenario can better understand the obstacles nearby via a robust environmental state which is encoded by AW. Our AW-based A2C also improves the convergence speed of the baseline algorithm LSTM-based A2C in the early-stage training. The PER further improves the convergence speed of the AW-based A2C by combining the online learning and batch learning to learn from data generated by the AW-based A2C itself. Therefore, we introduce the AW-PER-based A2C which converges steeply to a better reward without the expert experience. Our algorithm is shown in detail in the *Algorithm 1*, while the *Algorithms 2–4* are its subfunctions.

First, episodic actions are executed via the policy network $\pi(a_t|s_t;\theta)$ to generate experiences $< s_t, a_t, r_t, S_{t+1} >$ (lines 4–6). Second, Monte-Carlo returns R_t are computed according to the stored reward r_t from this episode . Episodic experiences (s_t, a_t, R_t) are stored in $\mathcal{E}$ which are also stored in the prioritized replay buffer $\mathcal{D}$ if $R_t > 0$ (lines 7–12). Third, the AW-based A2C is trained in a combined manner via the online learning and batch learning based on PER (lines 13–17). Fourth, these three steps repeat until that the network gets rid of the first-stage training (lines 3–18). Fifth, the weight of the first-stage training is saved for the network initialization of second-stage training, in which the AW-based A2C is only trained in an online manner (lines 20–32). Third-stage training is almost the same with the second stage-training. Their difference is that the third-stage training uses a smaller step size (learning rate) which encourages a stable convergence of network. Note that the state s_t here refers to the source state $s = [s_r, \{o_0 \dots o_N\}]$ which does not use any encoders to encode the state of obstacles nearby.

Algorithm 1: AW-PER-based A2C

1. Initialize the prioritized replay buffer $\mathcal{D}$
2. Initialize the AW-based A2C network θ
// First-stage training (single thread)
3. **For** episode $i < N_1$ **do**
4. **For** $s_t \neq S_{terminal}$ in episode *i* **do**
5. Execute action to generate the experience: $<s_t, a_t, r_t, S_{t+1} > \sim \pi(a_t|s_t;\theta)$
6. **end for**
7. **If** $s_t = s_{terminal}$ **then**
8. Compute Monte-Carlo returns: $R_t = \sum_k^{\infty} \gamma^{k-t} r_k$ for all *t* in episode *i*
9. Store the episodic experiences: $\mathcal{E} \leftarrow \mathcal{E} \cup (s_t, a_t, R_t)$
10 Update the replay buffer with prioritized experiences: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{E}$ if $R_t > 0$
11. Clear the episodic experiences: $\mathcal{E} \leftarrow \phi$
12. **end if**
13. Train online AW-based A2C: $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$
14. **If** length $(\mathcal{D})$ > K **do**
15. Train offline AW-based A2C: $\boldsymbol{Train\_A2C_{offline}}(\overline{\boldsymbol{\mathcal{D}}})$
16. **Break**
17. **end if**
18. **end for**
19. Save the model of first-stage training: $\theta_1 \leftarrow \theta$

// Second-stage or third-stage training
20. Initialize the networks by θ_1
21. **For** episode $i < N_2$ **do**
22. **For** $s_t \neq s_{terminal}$ in episode *i* **do**
23. Execute action: $<s_t, a_t, r_t, s_{t+1} > \sim \pi(a_t|s_t;\theta)$
24. **end for**
25. **If** $s_t = s_{terminal}$ **then**
26. Compute Monte-Carlo returns: $R_t = \sum_k^{\infty} \gamma^{k-t} r_k$ for all *t* in episode *i*
27. Store the episodic experiences: $\mathcal{E} \leftarrow \mathcal{E} \cup (s_t, a_t, R_t)$
28. Clear the episodic experiences: $\mathcal{E} \leftarrow \phi$
29. **end if**
30. Train online AW-based A2C: $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$
31. **end for**
32. Save the model of 2nd-stage/3rd-stage training: θ_2 or $\theta_3 \leftarrow \theta$

The subfunction $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$ first computes the probability distribution of the actions and values by the subfunction $\boldsymbol{NN_{a2c}}(\boldsymbol{s})$ (line 1). Second, TD-error δ is obtained by $\delta = R_t - v(\cdot;\theta)$ which is used for computing the policy loss and value loss (line 2–4). Third, the weight of AW-based A2C θ is updated according to the gradient of value loss and the policy loss (line 5 where η is the learning rate).

Algorithm 2: $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$

//training of AW-based A2C

1. Compute value and probability distribution: $\pi(\cdot;\theta), v(\cdot;\theta) \leftarrow \boldsymbol{NN_{a2c}}(\boldsymbol{s})$
2. Compute TD-error: $\delta = R_t - v(\cdot;\theta)$ for all t in $\mathcal{E}$
3. compute the policy loss: $L_{policy}^{online} = -\log \pi(\cdot;\theta)\cdot\delta - \alpha\mathcal{H}_t^{\pi_\theta}$
4. compute the value loss: $L_{value}^{online} = \frac{1}{2}\|-\delta\|^2$
5. update the weight AW-based A2C: $\theta \leftarrow \theta + \eta\cdot\nabla_\theta(L_{policy}^{online} + \beta^{online}L_{value}^{online})$

Unlike $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$ in which the network is trained once in a manner of online learning, the network in the subfunction $\boldsymbol{Train\_A2C_{offline}}(\overline{\mathcal{D}})$ can be trained for M times (line 1) to further update the network intensively in a manner of batch learning. $\boldsymbol{Train\_A2C_{offline}}(\overline{\mathcal{D}})$ cannot work independently. It must work with $\boldsymbol{Train\_A2C_{online}}(\mathcal{E})$ as its *supplement*. First, a batch of experience $\overline{\mathcal{D}}$ is sampled from the prioritized replay buffer $\mathcal{D}$ according to the probability $P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$ (line 2). Second, the importance-sampling weight can be obtained accordingly by $w_j = \frac{(N\cdot P(j))^{-\beta}}{max_k w_k}$ where N denotes the number of experiences $<s_j, a_j, R_j, p_j, idx>$ in the $\overline{\mathcal{D}}$ (line 3). Third, the policy distribution and value are obtained by executing the subfunction $\boldsymbol{NN_{a2c}}(\boldsymbol{s})$. The TD-errors δ_j and new priorities $(\delta_j)_+$ are obtained accordingly by $\delta_j = (R_j - v(\cdot;\theta))$ and $(\cdot)_+ = \max(\cdot, 0)$ (lines 4–6). Fourth, the policy loss and value loss in the batch learning of AW-based A2C are obtained. They are used for updating the network θ (lines 7–9). Fifth, the priority of experiences is updated for the next training (line 10).

Algorithm 3: $\boldsymbol{Train\_A2C_{offline}}(\overline{\boldsymbol{\mathcal{D}}})$

1. **For** $i < M$ do
2. Sample a batch of experience $\overline{\mathcal{D}} <s_j, a_j, R_j, p_j, idx>$ from $\mathcal{D}$: $\overline{\mathcal{D}}_j \sim P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$
3. Compute importance-sampling weight: $w_j = \frac{(N\cdot P(j))^{-\beta}}{max_k w_k}$
4. Compute the policy distribution and value: $\pi(\cdot;\theta), v(\cdot;\theta) \leftarrow \boldsymbol{NN_{a2c}}(\boldsymbol{s})$
5. Compute TD-error: $\delta_j = (R_j - v(\cdot;\theta))$ for all j in $\overline{\mathcal{D}}$
6. Compute new priority: $(\delta_j)_+$ for all j in $\overline{\mathcal{D}}$
7. Compute policy loss: $L_{policy}^{offline} = -\log \pi(\cdot;\theta)\cdot(\delta_j)_+\cdot\overline{p}_j - \alpha\mathcal{H}_t^{\pi_\theta}$
8. Compute value loss: $L_{value}^{offline} = v(\cdot;\theta)\cdot(-\delta_j)\cdot\overline{p}_j$
9. Update the network: $\theta \leftarrow \theta + \eta\cdot w_j\cdot\nabla_\theta(L_{policy}^{offline} + \beta^{offline}L_{value}^{offline})$
10. Update priority: $p_j \leftarrow (\delta_j)_+$
11. **end for**

The subfunction $NN_{a2c}(s)$ is about the forward propagation of AW-based A2C network which consists of two parts: *AW network* and *A2C network*. First, the AW network computes the embedded feature e_i and interaction feature h_i by MLP layers f_e and f_h without the activation function (lines 1–2). Second, the mean weight of embedded feature e_{mean} is obtained according to all embedded features (line 3). Third, the attention score of the robot to the obstacle o_i and its surrounded obstacles is obtained by a MLP layer f_α (line 4). Fourth, the description of surrounding obstacles S_o^{aw} is obtained by $S_o^{aw} = \sum_{i=1}^{n}[softmax(\alpha_i)] \cdot h_i$ (line 5). Fifth, the state of robot and the description of surrounding obstacles form the full description of environmental state S^{aw} (line 6). Finally, the policy distribution of actions and the state value are obtained (line 7) by A2C network f_{a2c} that needs to be designed to find the optimal configurations in the experiment for better performance of the AW-based A2C.

Algorithm 4: $NN_{a2c}(s)$

// Prepare the AW-based joint environmental state S^{aw}

1. Compute the embedded feature: $e_i = f_e(s_r, o_i, M_i; w_e)$
2. Compute the interaction feature: $h_i = f_h(e_i; w_h)$
3. Compute the mean weight of embedded feature: $e_{mean} = \frac{1}{n}\sum_{i=1}^{n} e_i$
4. Compute the attention score: $\alpha_i = f_\alpha(e_i, e_{mean}; w_a)$
5. Compute the description of surrounding obstacles: $S_o^{aw} = \sum_{i=1}^{n}[softmax(\alpha_i)] \cdot h_i$
6. Compute the environmental state: $S^{aw} = [s_r, S_o^{aw}]$

// Forward propagation of A2C network

7. Compute the policy distribution and state value: $\pi, v \leftarrow f_{a2c}(S^{aw}; \theta_{a2c})$

Experiments and results

Experimental environment

The experiments are conducted in the simulators with one robot and multiple obstacles. The behaviors of robot are controlled by the trained policies, while the behaviors of obstacles are controlled by ORCA algorithm. The policies of robot are trained in the circle-crossing simulator. They are tested in the square-crossing and circle-crossing simulators simultaneously (Fig. 3). Other environment settings: (1) action space: action space consists of 81 actions that include *no action* and *5 speed choices* in each direction from *16 moving directions*; (2) time limit and time cost in each step: time limit of each episode is set to 25 s and one step/action costs 0.25 s.

Design of network architecture

Configurations of layer and node

Before designing the network architecture, we first introduce our expectations to reach our goals of algorithm training: (1) as less training data as possible; (2) as short training time as possible. Note that these goals are achieved under the condition that the accuracy and reliability of initial algorithms are not affected. The number of layers should be kept in a reasonable slot. The network with large number of hidden layers is not feasible for our task because the large network requires more data to converge. Its training process is also time-consuming. According to principles for the configuration of the layer and node in each layer, the network with two hidden layers is sufficient to solve any non-linear classification or regression problems. Hence the shortcoming of learning inefficiency caused by one hidden layer is better avoided (Brownlee, 2018; Goodfellow et al., 2016; Hornik et al., 1989; Lippmann, 1988; Reed & MarksII, 1999). However, the learning efficiency may be improved with the increase of number of hidden layers, but the consumed training time may increase accordingly.

Given the number of node in each layer reported in literatures (Everett et al., 2018), the *grid test* is used to find the optimal or near-optimal number of layer and number of node in each layer. Let N_{layer} denotes the number of the *hidden layer*, and the grid test is conducted under $N_{layer} \in \{1, 2\}$. Let N_{node} denotes the number of nodes in each layer, and the grid test is conducted under $N_{node} \in \{64, 128, 256\}$. Activation function is required to activate the hidden layers, and *Tanh functions* is used in this experiment. Hence, two types of architecture for A2C are obtained, and they are shown in Fig. 4. Note that the actor network and critic network in these two architectures only share the same input and output. Parameters in the layers cannot be shared in these

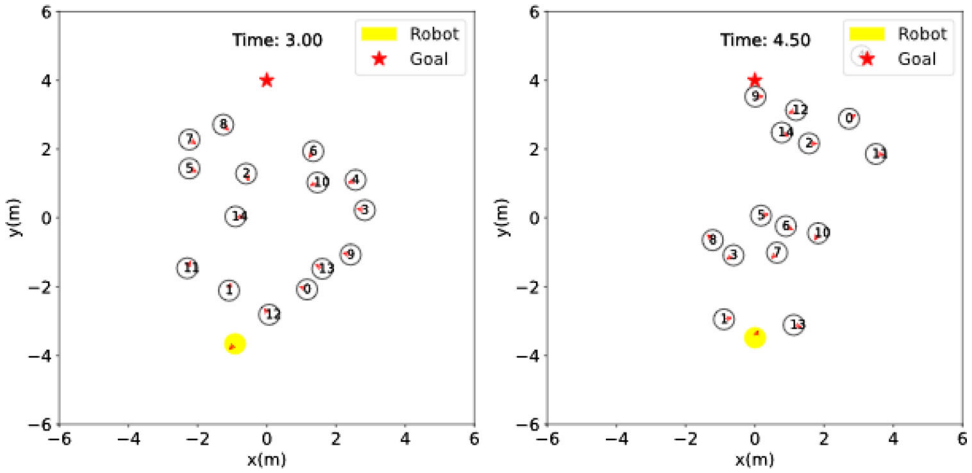
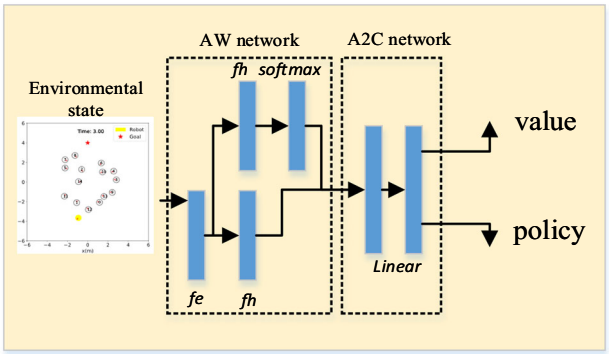
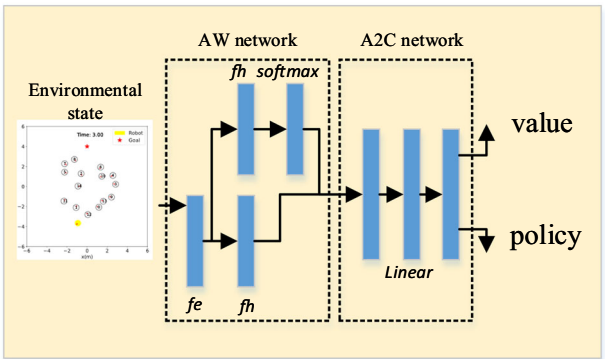


Fig. 3 Circle-crossing and square-crossing simulators. Left and right figure denotes the circle-crossing and square-crossing simulators respectively with the random attribute in the experiments. Random attribute denotes the random generation of the starting points and destinations, but the agents will still follow the basic circle-crossing or square-crossing rules, which refers to that the agents move from one side of circle (in circle-crossing simulator) or square (in square-crossing simulator) to their opposite side



(a)



(b)

Fig. 4 Two possible architectures of AW-based A2C network. AW-based A2C network with three hidden layers (**b**) is used as the comparison of two-hidden-layer case (**a**) for testing the claim that two hidden layers are sufficient and efficient for creating any classification regions of desired shape

architectures because networks cannot converge once the task is to solve the continuous control problem (Schulman et al., 2017). Recall that AW-based A2C consists of AW network and A2C network. We attempt to keep the configurations of AW network in (Chen et al., 2019b), and then find the near-optimal configurations of A2C network. That means the configurations of AW networks are set to [150, 100], [100, 50] and [100, 100] respectively for MLP layers that consist of the layer of embedded feature f_e, interaction feature f_h and attention score f_a. The AW network is included in the propagation of AW-based A2C network. In other words, AW network provides two layers (one input layer and one hidden layer) to the AW-based A2C network. Hence A2C network just need two layers (one hidden layer and one output layer) theoretically according to the claim that two hidden layers are *sufficient and efficient* for creating any classification regions of desired shape (Goodfellow et al., 2016; Reed & MarksII, 1999).

Grid test (grid search)

To better select the optimal or near-optimal architecture for A2C algorithm, we consider seven factors (Table 1) in which *value coefficient, entropy coefficient, optimizer,* and *discount factor* are set to be stable. *The learning rate, input dimension* (S^{aw} dimension or the output dimension of AW network), and *multiple obstacle* (complex feature) are set to different values. Given $N_{node} \in \{64, 128, 256\}$, the first architecture (Fig. 4a) has *three configurations,* while the second architecture (Fig. 4b) has *nine configurations* (Table 2). The grid test is expected to involve 96($12 \times 2 \times 2 \times 2$) experiments. It is time-consuming to train and evaluate the performance of all configurations. Hence, we first chose a simple one-robot one-obstacle case to test the efficiency and efficacy of each possible configurations before the grid test. Then we select

Table 1 The factors considered in the grid test

Learning rate	Input dimension	Multiple obstacles (complex feature)	Value coefficient	Entropy coefficient	Optimizer	Discount factor
Default/ 3e−4	13/56	1/3	0.5	0.001	Adam	0.95

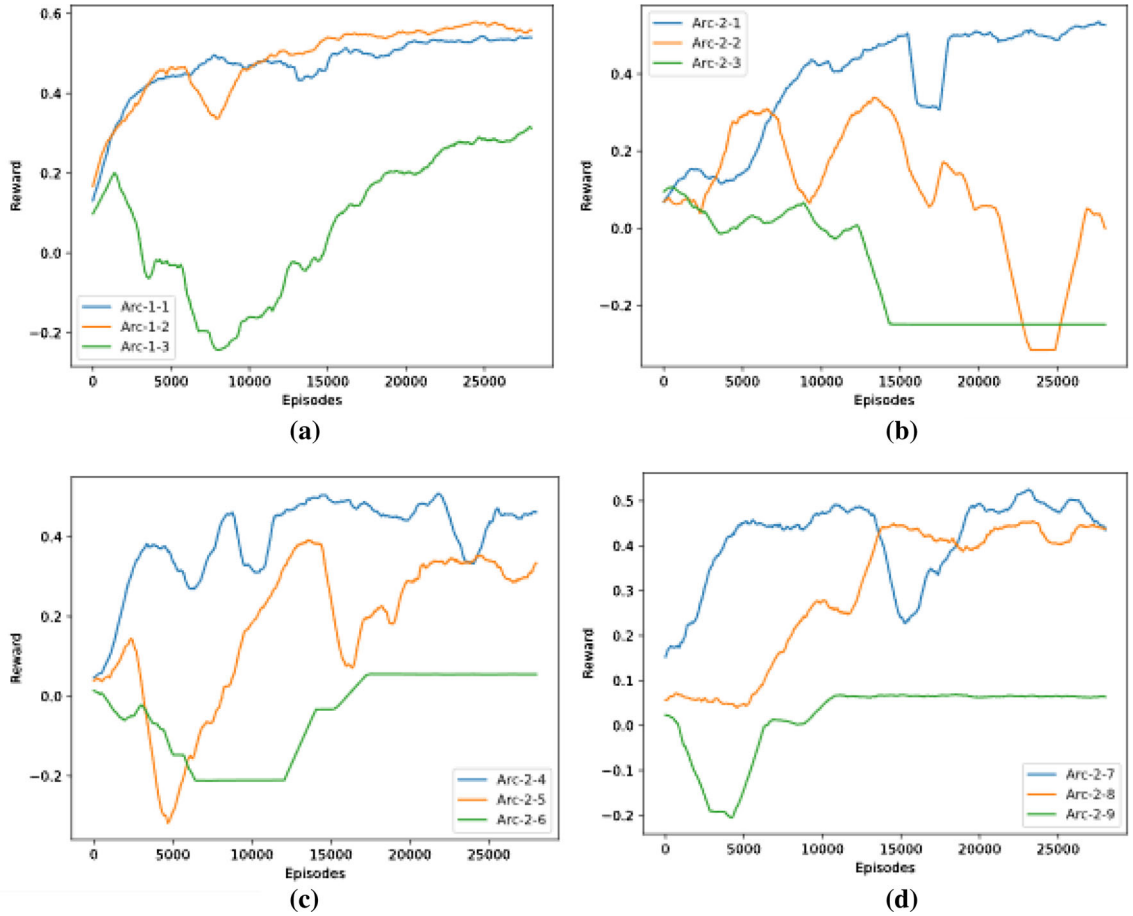


Fig. 5 Initial training results of one-obstacle case without any encoders before the grid tests. **a** Denotes the training curves of Architecture-1-1, Architecture-1-2 and Architecture-1-3. **b** Denotes the training curves of Architecture-2-1, Architecture-2-2 and Architecture-2-3. **c** Denotes the training curves of Architecture-2-4, Architecture-2-5 and Architecture-2-6. **d** Denotes the training curves of Architecture-2-7, Architecture-2-8 and Architecture-2-9. All architectures are tested in the same learning rate (default = 0.01)

the best configurations for the further tests. The results of training are shown in Fig. 5.

The first grid test (impact from the learning rate): the learning rate is the most important hyper-parameters in the training (Goodfellow et al., 2016). A larger learning rate may lead to a fast convergence of network to the sub-optimal and the weight fluctuation of network, while a smaller learning rate makes the training slow relatively. Hence, it is necessary to tune the learning rate to find the best trade-off for the training. However, instead of finding the best learning rate for training, here our aim is about figuring out the impact of learning rate to the performance of network convergence by changing the learning rate from the default larger one to the smaller one (3e−4). The test results with smaller learning rate are shown in Fig. 6b, while Fig. 6a denotes the results with a larger learning rate. The second grid test (impact from the input dimension): performances of configuration are further tested by changing the input dimensions from 13 to 56 for one-robot one-obstacle case by applying the AW encoder (AW network) to describe the environmental state. Figure 6b works as the benchmark in which its input dimension and learning rate are 13 and 3e−4 respectively. Figure 6c denotes the test results of configurations with input dimension 56, and its learning rate keeps 3e−4. The third grid test (impact from the complex feature): number of obstacles changes from 1 to 3 to add the complexity of the environment. Figure 6d denotes the test

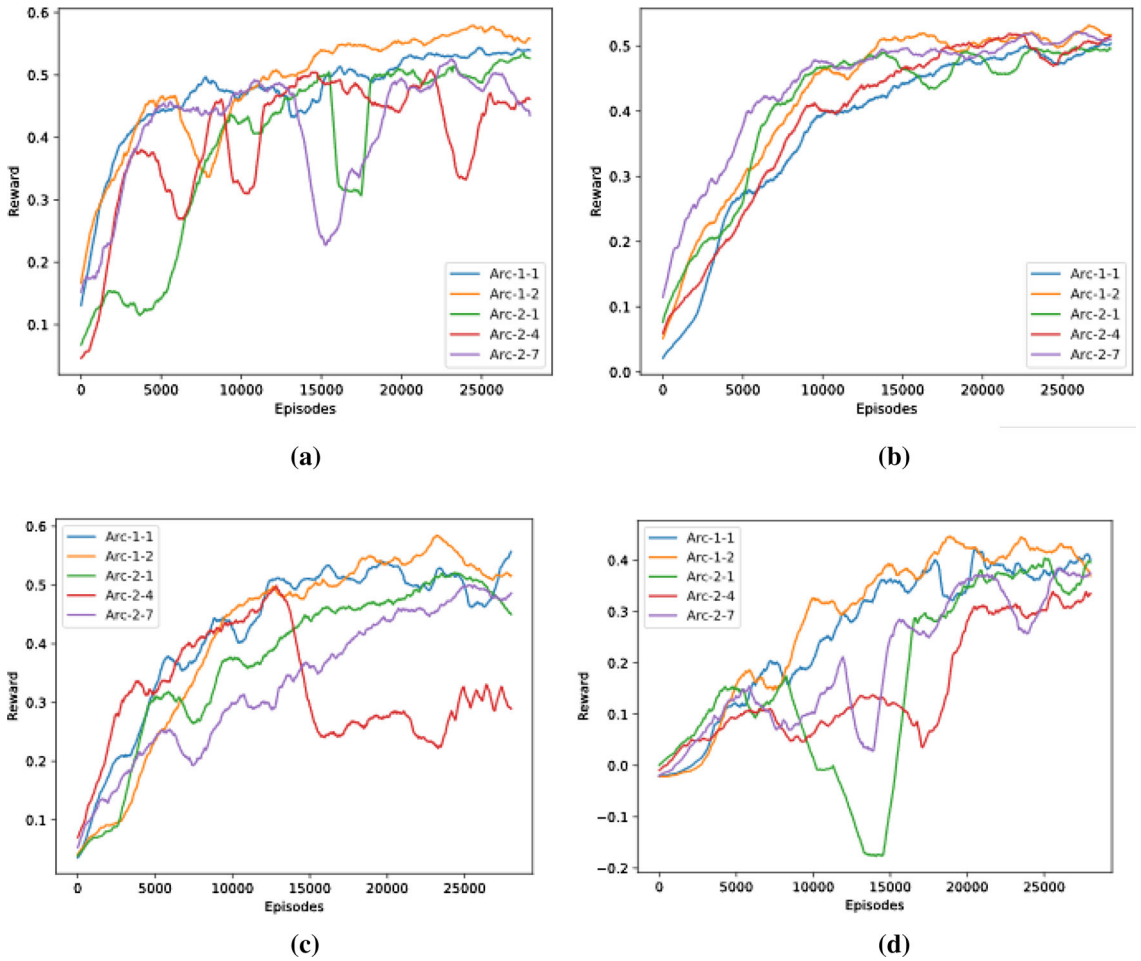


Fig. 6 Results of grid test. **a** Denotes the results with default learning rate, 13 input dimensions and one-obstacle case. **b** Denotes the results with learning rate 3e−4, 13 input dimension and one-obstacle case. **c** Denotes the results with learning rate 3e−4, 56 input dimensions and one-obstacle case. **d** Denotes the results with learning rate 3e−4, 56 input dimensions and three-obstacle case

results of these configurations, and their learning rates also keep 3e−4.

Efficacy and efficiency analysis of possible architectures

According to Fig. 6, all configurations in the candidate architectures are qualified to reach the goal of convergence with high rewards, except for Arc-2-4 in Fig. 6c. Arc-1-2 outperforms other configurations in the efficiency of convergence and the value of converged reward, regardless of the learning rate, input dimensions and complexity of feature. Hence, the Arc-1-2 is selected as the network configuration for the further experiments. Detailed observations of test results include: (1) small learning rate slows down the convergence speed slightly but leads to a stable convergence simultaneously; (2) large input dimension shows less impact to cases with less hidden layer (Arc-1-1 and Arc-1-2), while the convergence speed in cases with more hidden layer (Arc-2-1, Arc-2-4, and Arc-2-7) are slightly slowed down, because a large input dimension causes a larger number of *the learning unit* in configurations, especially for networks with more layers; (3) a complex feature causes a slow and fluctuated convergence speed to the configurations with more layers in the early-stage training, while cases with less layer are more robust when the number of obstacles in the environment increases.

Model training

Basic settings of training

The behaviors of robot are controlled by the trained policies, while the behaviors of obstacles are controlled by ORCA policy in the training. The simulator for model training is the circle-crossing simulator, while the square-crossing simulator is for model evaluation. All algorithms in the training include *LSTM-based A2C*, *AW-based A2C* and *AW-PER-based A2C*. We first implement LSTM-based A2C and

Table 2 Possible configurations in two architectures

Name	Configurations
Arc-1-1	[input dim, 64] + Tanh + [64, output dim]
Arc-1-2	[input dim, 128] + Tanh + [128, output dim]
Arc-1-3	[input dim, 256] + Tanh + [256, output dim]
Arc-2-1	[input dim, 64] + Tanh + [64, 64] + Tanh + [64, output dim]
Arc-2-2	[input dim, 64] + Tanh + [64, 128] + Tanh + [128, output dim]
Arc-2-3	[input dim, 64] + Tanh + [64, 256] + Tanh + [256, output dim]
Arc-2-4	[input dim, 128] + Tanh + [128, 64] + Tanh + [64, output dim]
Arc-2-5	[input dim, 128] + Tanh + [128, 128] + Tanh + [128, output dim]
Arc-2-6	[input dim, 128] + Tanh + [128, 256] + Tanh + [256, output dim]
Arc-2-7	[input dim, 256] + Tanh + [256, 64] + Tanh + [64, output dim]
Arc-2-8	[input dim, 256] + Tanh + [256, 128] + Tanh + [128, output dim]
Arc-2-9	[input dim, 256] + Tanh + [256, 256] + Tanh + [256, output dim]

then optimize it by applying the AW encoder to describe the environmental information for the improvements in the convergence speed and reduction of over-fitting. Finally, the convergence speed of *AW-based A2C* is further accelerated by applying the batch learning with PER and new training strategy.

LSTM-based A2C

Existing LSTM-based algorithms to tackle the complex indoor motion planning problem are too dependent on the expert experience in the network initialization (Chen et al., 2019b; Everett et al., 2018). Otherwise, their networks cannot converge even in one-obstacle case because of their incompetent reward function which is designed for networks initialized by the imitation learning. We first implement the online A2C with the reward function reported in (Everett et al., 2018; Chen et al., 2017, 2019b). In the implementation, the states of the agent are encoded by LSTM encoder to form a new description of the environment. Then this LSTM-based description of environmental state is concatenated with the state of the robot as the input of A2C algorithm (Fig. 7). As a result, the reported reward function leads to the difficulty in network convergence, hence the reported reward function is replaced by a new reward function (Eq. 34). The reward received by the robot is 1 if the robot reaches the goal ($p_{current} = p_g$ where $p_{current}$ denotes the position of the robot while p_g denotes the position of the goal) within

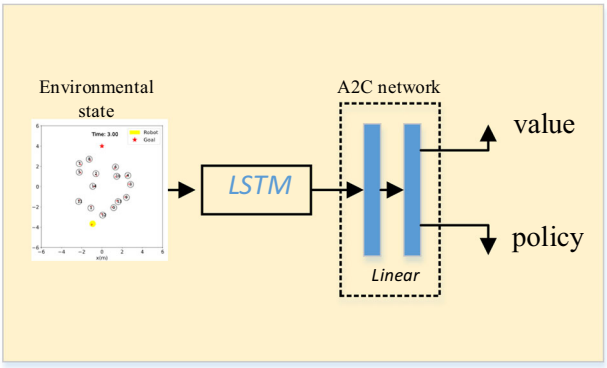


Fig. 7 The architecture of LSTM-based A2C network in which the A2C network is the same as that of the AW-based A2C in Fig. 4. Number of recurrent layers in LSTM is set to the default value (default = 1). The hidden dimension of LSTM is set to 50

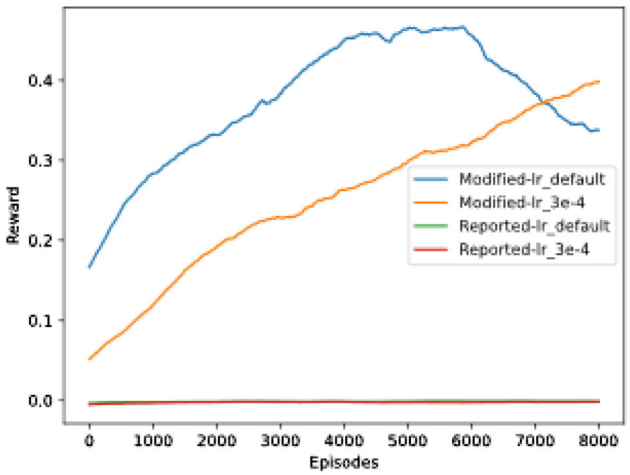


Fig. 8 Results of training for the LSTM-based A2C with reported reward function and modified reward function. Modified reward function contributes to the convergence of network regardless of learning rate, while the reported reward function cannot learn anything because the robot cannot find the goal (success reward) in the early-stage training

the time limit. If $0 < d_{min} < 0.2$ where d_{min} denotes the minimum distance of the robot and obstacles, the reward received by the robot is set to $-0.1 + \frac{d_{min}}{2}$. If the robot collides with the obstacles, the reward is set to -0.25. If the experimental time reaches the time limit $t = t_{max}$ and the robot does not reach the goal $p_t \neq p_g$, the reward is set to $\frac{d_{start\_to\_goal} - (p_g - p_{current})}{d_{start\_to\_goal}} \cdot 0.5$ where $d_{start\_to\_goal}$ denotes the distance of the start to the goal. New reward function accelerates the convergence speed by attaching a reward to *the final position of the robot* that are with shorter distance to the goal. The training results of LSTM-based A2C with reported and modified reward functions in the case with one obstacle are shown in Fig. 8.

Note that in current reinforcement learning or deep learning, the setting of constant or the setting of reward is decided

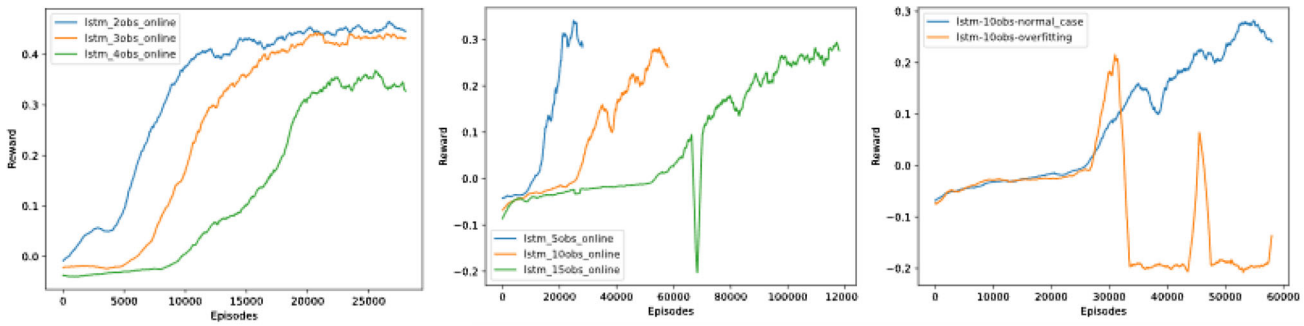


Fig. 9 Training curves of LSTM-based A2C in multiple-obstacle cases. **a** Denotes the training in cases with two, three and four obstacles. **b** Denotes the training in cases with five, ten and fifteen obstacles. **c** Denotes a normal training case with ten obstacles and an outlier case in the overfitting (yellow curve)

Table 3 The parameters/hyper-parameters considered in trainings of LSTM-based and AW-based A2C

Learning rate	Input dimension	Multiple obstacles	Value coefficient	Entropy coefficient	Optimizer	Discount factor
3e−4	56(6 + 50)	2/3/4/5/10/15	0.5	0.001	Adam	0.95

according to the *intuition* and *trial and error*. Then a parameter which leads to better performance will be selected. Here parameters of reward (e.g., 1, − 0.25 and 0) is the feedback to the robot when the robot interacts with the environment. The setting of reward function is from the trial and error. The rewards will be used in the backpropagation indirectly to make the neural network update towards the direction of convergence (global optimum). Otherwise, the network may not converge or converges towards the local optimum. The safe distance 0.2 m is set artificially according to the requirement of tasks (e.g., case with high-speed obstacles). The safe distance can be 0.1 m if obstacles move slow.

$$R(s,a) = \begin{cases} 1 \; if \; \mathrm{p_{current}} = \mathrm{p_g} \\ -0.1 + \frac{d_{min}}{2} \; if \; 0 < \mathrm{d_{min}} < 0.2 \\ -0.25 \; if \; \mathrm{d_{min}} < 0 \\ \frac{d_{start\_to\_goal} - (p_g - p_{current})}{d_{start\_to\_goal}} \cdot 0.5 \; if \; \mathrm{t} = \mathrm{t_{max}} \; and \\ \mathrm{p_t} \neq \mathrm{p_g} \\ 0 \quad otherwise \end{cases} \tag{34}$$

The experiments of LSTM-based A2C algorithm are extended to the multiple-obstacle cases in which the number of obstacles is $N \in \{2, 3, 4, 5, 10, 15\}$ (Table 3). Motion of obstacles is controlled by ORCA policy. The circle-crossing simulator is used in the training. LSTM-based A2C algorithm can successfully accomplish the motion planning tasks in multiple-obstacle cases. However, the training of robot's policy still suffers *the suboptimal reward* and *slower convergence speed* with the increase of obstacles nearby as Fig. 9a, b. Moreover, the LSTM-based A2C is likely to fall into the problem of the *over-fitting* because LSTM encodes the states of the agent according to distances of the obstacle to the robot. This distance-based encoding method causes the result that the closer obstacles have larger impacts on the robot. However, it cannot work always, e.g., a failed training case in Fig. 9c, because the distance-based encoding method describes the environmental state *partially*. Other factors like the speed and moving direction of obstacles should be considered as well. Hence, a robust description of environmental state is needed to better solve the *over-fitting*, *slow convergence speed* and *suboptimal reward* problems. This is achieved by applying the AW encoder to replace the LSTM encoder.

AW-based online A2C

New environmental state is descripted or interpreted by the AW encoder that consists of four versions: *basic version*, *global state version*, *occupancy map version* and *full version*. Note that the global feature $e_{mean} = \frac{1}{n}\sum_{k=1}^{n} e_k$ and the occupancy map $M_i(a, b, :)$ are not included in the basic version, while the full version includes them simultaneously. The performances of four versions of AW-based A2C in the multiple-obstacle training are shown in Fig. 10, in which (a–d) represent the training results of full version, occupancy map version, global state version and basic version respectively.

According to their training results, they all follow the same trend: there are less differences in the convergence speed and converged reward in lesser obstacle cases for four versions of

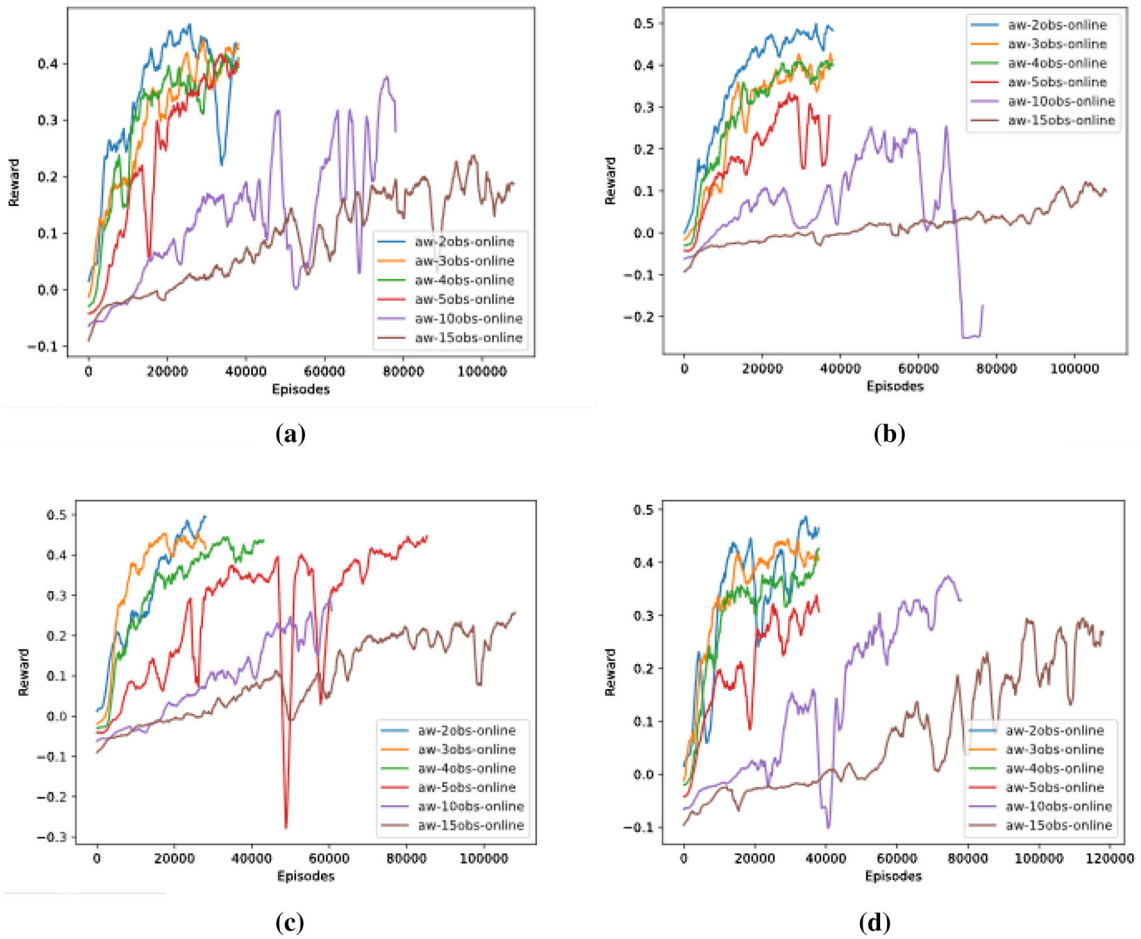


Fig. 10 Comparisons of AW-based A2C in multiple-obstacle cases. **a**–**d** denote the training results of the full version, occupancy map version, global state version and basic version of AW encoder respectively

AW. Their convergence speeds are fast, and rewards obtained are high. However, the convergence speeds slow down and rewards obtained become smaller with the increase of obstacles.

The training curves of AW-based A2C are also compared with that of LSTM-based A2C (Fig. 11). The criterions for comparisons are the *number of episodes used for early-stage training* and the *peak reward within the same number of episodes*. Note that the parameters/hyper-parameters considered in the training of AW-based A2C is the same with that of LSTM-based A2C in Table 3. The early-stage training here is defined as the *moment at which the robot can find its goal constantly*. This means the average accumulative reward and success rate reach around 0.1 and 40% respectively. Detailed comparisons of these two algorithms are listed in Table 4.

According to the comparisons, near all versions of AW-based A2C largely outperform the LSTM-based A2C to get rid of the early-stage training, not only in cases with less obstacles but also the cases with dense obstacles. The global state version works better in cases with dense obstacles, while the occupancy map version did not take obvious effect when comparing with the basic version. However, obvious improvements are easy to be noticed once the global state and occupation map work together to form the full version of AW-based A2C. The training curves and peak reward of AW-based A2C are also slightly better than that of LSTM-based A2C. The full version of AW-based A2C performs robust than the rest versions of AW-based A2C. However, AW-based A2C still converges slowly in cases with dense obstacles, although it outperforms the convergence speed of LSTM-based A2C. The convergence speed of AW-based A2C can be further improved by applying the batch learning with PER to make the best of experiences discarded in the online learning.

AW-PER-based A2C and its training strategy

The batch learning with PER steeply improves the convergence speed of AW-based A2C in the early-stage training, but the network of AW-based A2C eventually converges to a suboptimal reward since the distribution of experiences collected from the dense and dynamic environment for the

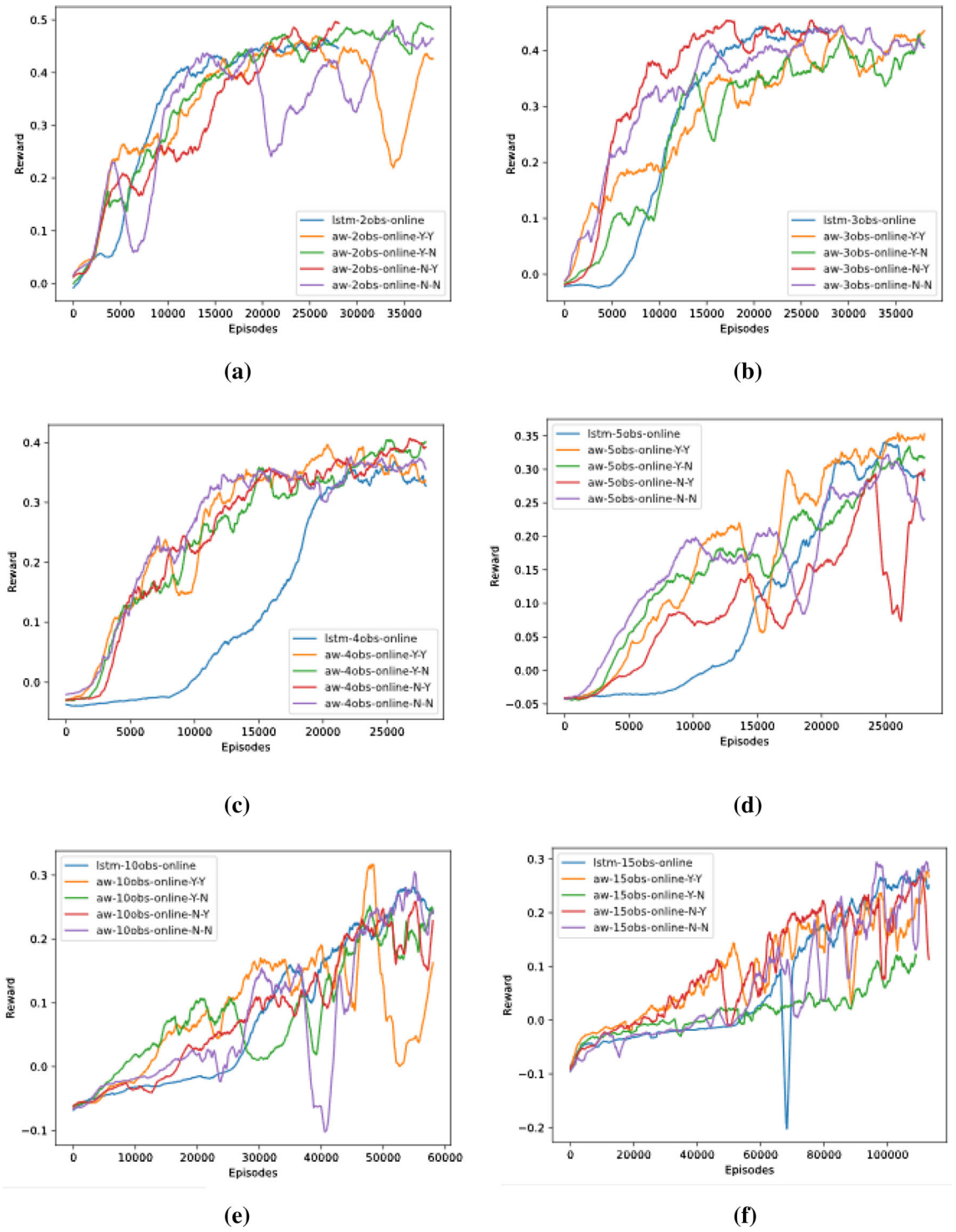


Fig. 11 LSTM-based A2C and AW-based A2C in multiple-obstacle cases. **a**–**f** Denote their comparisons in cases with 2, 3, 4, 5, 10 and 15 obstacles

Table 4 Training analysis of AW-based A2C and LSTM-based A2C

Algorithm	Episode used for early-stage training in 2/3/4/5/10/15 obstacle cases	Peak reward
LSTM-based A2C	5000/8000/15000/15000/33000/70000	4.5/4.4/3.6/3.4/2.7/2.7
Basic version	2500/2800/3500/5500/30000/65000	4.8/4.4/3.7/3.2/3.1/2.9
Global state version	2500/3000/4000/13000/20000/45000	4.9/4.5/4.1/3.0/2.5/2.7
Occupancy map version	2500/5000/3500/6500/30000/95000	4.9/4.2/4.1/3.3/2.5/1.2
Full version	**2500/2000/3000/**7500/25000/**45000**	4.7/4.4/3.9/**3.6/3.2/**2.7

Bold values indicate the best performance

batch learning changes in the training. This may cause the result that the final converged policy or network is slightly different from that of online AW-based A2C, and some unexpected collisions follow. Hence, we choose a training strategy that consists of three training stages: (1) AW-based A2C is trained in a manner of online learning and batch learning in the early-stage training; (2) the model from the first-stage training is trained in an online learning manner; (3) the model from the second-stage training continues to be trained online with a smaller learning rate. A suboptimal model is obtained from the first-stage training with less dataset. The model then converges from the suboptimal to near-optimal in the second-stage training. The performance of model can be further improved slightly in the third-stage training by applying a smaller learning rate.

The experiments of AW-PER-based A2C include cases with 5 and 10 obstacles to represent two levels of density of obstacles: *normal density*, and *high density*. The parameters/hyper-parameters considered in the trainings of AW-PER-based A2C are shown in Table 5.

The results of the first and second-stage trainings with 10 obstacles are shown in Fig. 12. We find that the batch learning is sensitive to the large learning rate (1e−2) in the training, hence contributing less to the convergence speed. This problem is better solved by applying a smaller learning rate (1e−4) in the training (Fig. 12a). According to the training results, AW-PER-based A2C costs near 10,000 episodes to get rid of the early-stage training. However, online AW-based A2C and LSTM-based A2C spend around 25,000 and 32,000 episodes respectively in the first-stage training. For the second-stage training of AW-PER-based A2C in which the model is initialized by a model trained with 10,000 episode, 15,000 episodes are used to reach a near-optimal reward. However, AW-based A2C and LSTM-based A2C cost 47,000 and 53,000 episodes respectively to reach that level.

The results of first and second-stage trainings with 5 obstacles are shown in Fig. 13. According to that, it is obvious to notice that the AW-PER-based A2C takes near 2000 episodes to get rid of the early-stage training, while AW-based A2C and LSTM-based A2C take 8000 and 15,000 episodes respectively to achieve that result. The model of AW-PER-based A2C trained with 2000 episodes continues to be trained in an online manner. A near-optimal model is obtained after 18,000 episodes. However, AW-based A2C and LSTM-based A2C cost around 30,000 episodes respectively to reach almost the same result.

The results of third-stage training in cases with 5 and 10 obstacles are shown in Fig. 14. The performance of AW-PER-based A2C can be further improved by applying a smaller learning rate in the training. The performance of model improved a lot in the case with 5 obstacles in the third-stage training (near 0.06 in the reward), while the reward increased a little in the case with 10 obstacles (around 0.03). It is hard to see further improvement after 10,000 episodes for cases with 10 and 5 obstacles, hence the third-stage models after 10,000 episodes are saved for the model evaluations that will be shown in the next section.

Model evaluations

We selected five evaluation indicators that consist of training evaluations, quantitative evaluations, qualitative evaluations, computational evaluations, and robustness evaluations to evaluate the performance of algorithms. These indicators are widely used in reinforcement learning. Many works just select one or two to evaluate their algorithms. It is not enough. Hence, we summarize mentioned indicators from many reinforcement learning works. Then, these indicators are used to evaluate our algorithms to give readers a comprehensive understanding.

This section first summarizes the results of training (training evaluations). Then the trained models are evaluated from four perspectives: (1) *quantitative evaluations*; (2) *qualitive evaluations*; (3) *computational evaluations*; (4) *robustness evaluations*. Note that our experiments did not involve many fine-tunings of three algorithms in the training. We believe that there still some spaces for further enhancement in the performance of three algorithms.

Table 5 The parameters/hyper-parameters considered in the trainings of AW-PER-based A2C

Parameters/hyper-parameters of online learning							
Learning rate	Input dimension	Multiple obstacles	Value coefficient	Entropy coefficient	Optimizer	Discount factor	–
3e−4 3e−5 (2nd) 3e−6 (3rd)	56(50 + 6)	5/10	0.5	0.001 1e−5 (2nd) 1e−6 (3rd)	Adam	0.95	–

Parameters/hyper-parameters of batch learning							
Batch size	Priority coefficient	Importance-sampling coefficient	Training batches	Learning rate	Value coefficient	Value weight coefficient	Entropy coefficient
200	0.6	0.1	2	1e−4	0.1	0.01	1e−5

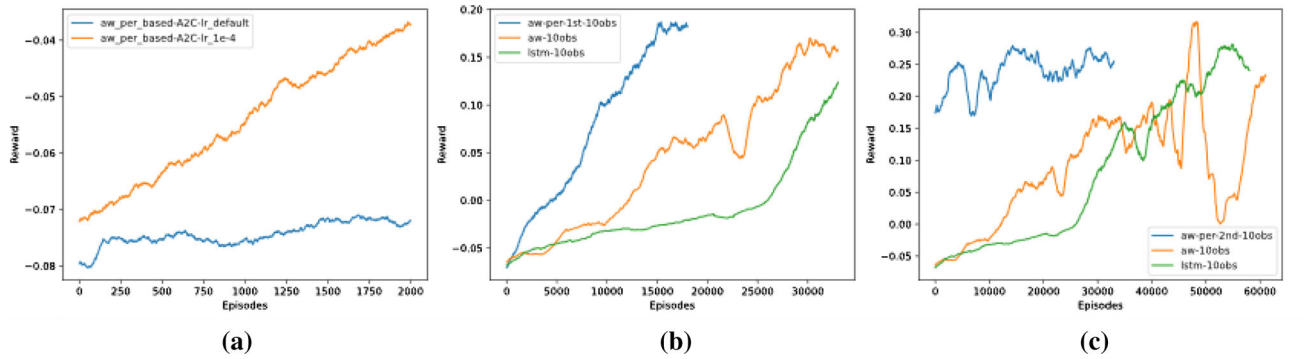


Fig. 12 The first and second-stage training of AW-PER-based A2C in the cycle-crossing simulator with 10 obstacles. **a** Denotes the worse impact of a larger learning rate to the convergence speed. **b**, **c** Denote the results of first and second stage respectively

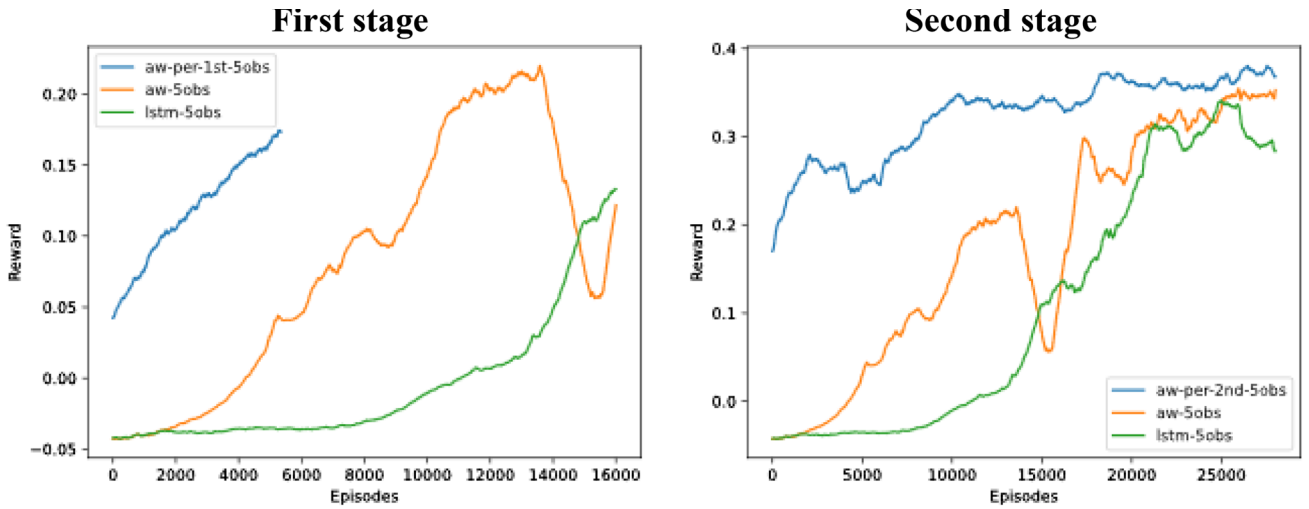


Fig. 13 The first and second-stage training of AW-PER-based A2C in the cycle-crossing simulator with five obstacles

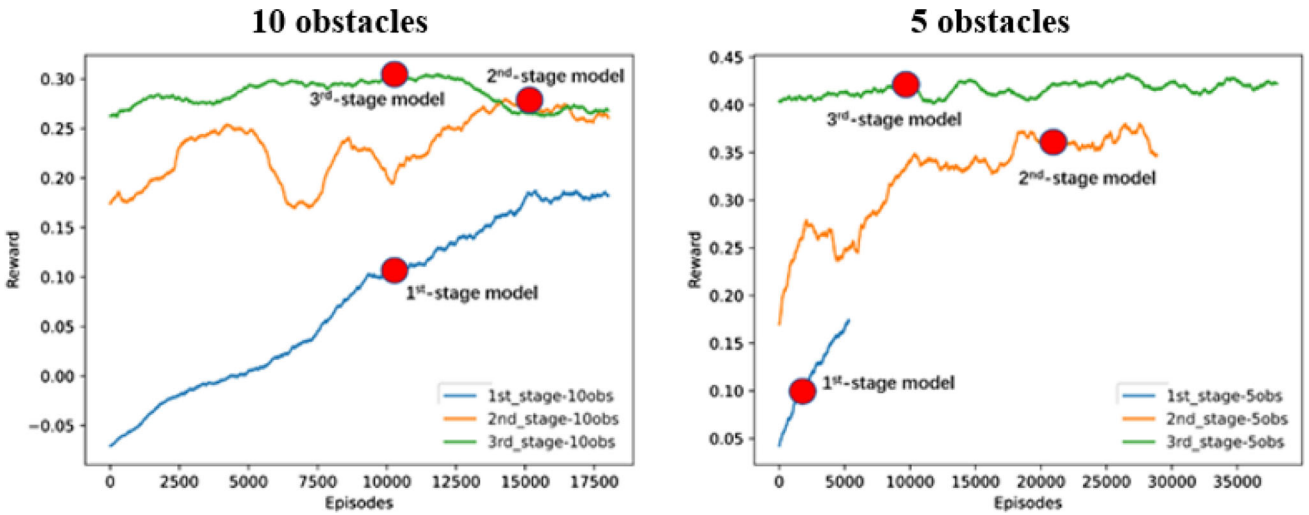


Fig. 14 All-stage training of AW-PER-based A2C in the cycle-crossing simulator with 10/5 obstacles. Left figure denotes the training results of the case with 10 obstacles, while right figure denotes that of the case with 5 obstacles

Table 6 Comparisons of three algorithms in the training

algorithms	Episode cost (1st stage, 10/5 obs.)	Reward (1st stage, 10/5 obs.)	Episode cost (2nd stage, 10/5 obs.)	Reward (2nd stage, 10/5 obs.)	Episode cost (3rd stage, 10/5 obs.)	Reward (3rd stage, 10/5 obs.)	Overall episode cost (10/5 obs.)
LSTM-based A2C	32,000/15000	0.1/0.1	28,000/15000	0.23/0.32	15,000/15000	0.31/0.36	75,000/45000
Our AW-based A2C	25,000/8000	0.1/0.1	35,000/22000	0.23/0.35	15,000/15000	**0.33**/0.38	75,000/45000
Our AW-PER-based A2C	**10,000/2000**	**0.1/0.1**	**15,000/**18000	**0.27/0.36**	**10,000/10000**	0.30/**0.42**	**35,000/30000**

Bold values indicate the best performance

Training evaluations

This part considers the converged average accumulative reward and the number of episodes spent in three training stages. Note that LSTM-based A2C and AW-based A2C don't have 2nd-stage training. They are first trained with 60,000 episodes. Then they are retrained with 15,000 episodes for a stable convergence. However, we still use the 2nd-stage training in these two algorithms for clear comparisons with AW-PER-based A2C. Detailed compassions of three algorithms are shown in Table 6. According to the results, our AW-based A2C and AW-PER-based A2C outperform the LSTM-based A2C, especially the AW-PER-based A2C which costs merely around 30%/15% of data (10,000/2000 episodes) to get rid of the 1st-stage training when comparing to that of LSTM-based A2C (32,000/15000 episodes). In the 2nd-stage training, AW-based A2C converges slightly slower than LSTM-based A2C, but the convergence speed of AW-PER-based A2C is faster than that of LSTM-based A2C, especially in cases with 10 obstacles. In the 3rd-stage training, AW-PER-based A2C outperforms the rest two algorithms not only in the convergence speed but also in the converged reward in cases with 5 obstacles, while its converged reward (0.30) is slightly lower than that of the rest two algorithms in cases with 10 obstacles (0.31 and 0.33). Overall, AW-PER-based A2C takes near 50%/30% of data (35,000/30000) to converge. Its converged reward is almost the same with that of the rest two algorithms in cases with 10 obstacles, but it is better than that of the rest two algorithms in cases with 5 obstacles.

Settings for model evaluations

Settings of the experiment for model evaluations are simply listed in Table 7. Models of three algorithms are evaluated in simulators with 10 and 5 obstacles. All the experiments for evaluations are conducted in the circle-crossing simulator, except for the experiments of robust evaluations which are conducted in the square-crossing simulator. Obstacles in the simulators are controlled by the ORCA policy, while the robot is controlled by the trained policies of three algorithms. Total number of the test for each algorithm is set to 500 episodes.

Quantitative evaluations

Six criterions are selected for the quantitative comparisons of three algorithms. These criterions include the *success rate*, *average time to goal*, *collisions rate*, *timeout rate*, *mean distance to obstacles*, and *accumulative discounted rewards*. Detailed comparisons are shown in Table 8. According to the comparisons, the accumulative rewards received by these three algorithms are almost the same as that in the training. Robots based on these three algorithms learnt how to keep a safe distance to the obstacles (in success cases). The AW-based A2C and AW-PER-based A2C outperform the LSTM-based A2C in the success rate and timeout rate. This means these two algorithms are easy to find their goals within the time limit (25 s), but AW-PER-based A2C seems likely to suffer more collisions in cases with 10 obstacles. There is less difference for the robots based on these three algorithms on the time spent to reach their goal in cases with 10 obstacles, while the robot controlled by AW-PER-based A2C outperforms the robots based on the rest two algorithms on the time to goal in cases with 5 obstacles.

Qualitative evaluations

The qualitative evaluation is about *how good the policy is* or *what strategies the robot learnt*. This is achieved by observing the trajectories of the robot. The policies based on AW-PER-based A2C are shown Fig. 15, while Fig. 16 presents the trajectory comparisons of LSTM-based A2C, AW-based A2C and AW-PER-based A2C.

According to Fig. 15, the robot is likely to learn a *"Recede-Wait-Forward"* strategy in the environment with *high-density* obstacles: the robot first moves out of the crowded obstacles and keeps a distance to them [Fig. 15a (a1–a5)]; the

Table 7 Environmental settings in the model evaluations

Algorithms	Number of obstacles	Simulators	Policy	Number of episodes
LSTM-based A2C (Everett et al., 2018; Mnih et al., 2016) Our AW-based A2C Our AW-PER-based A2C	10/5	Circle-crossing/square-crossing	ORCA (obstacles)/trained policy (robot)	500

Table 8 Comparisons of three algorithms in the test according to six criterions

Algorithms	Success rate (10/5 obs.)	Time to goal (10/5 obs.)	Collision rate (10/5 obs.)	Timeout rate (10/5 obs.)	Mean distance to obs. (10/5 obs.)	Rewards (10/5 obs.)
LSTM-based A2C	0.77/0.88	18.25 s/17.05 s	0.04/0.05	0.19/0.07	0.14 m/0.13 m	0.3068/0.3588
Our AW-based A2C	**0.84**/0.90	**18.17 s**/15.65 s	**0.03/0.03**	0.13/0.07	0.13 m/0.13 m	**0.3288**/0.3844
Our AW-PER-based A2C	**0.84/0.93**	18.28 s/**15.09 s**	0.09/0.04	**0.07/0.03**	0.13 m/0.14 m	0.2941/**0.4205**

Bold values indicate the best performance

robot then waits for a while until the obstacles start to move away [Fig. 15a (a6–a7)]. The last, the robot moves right forward towards the goal with the highest speed [Fig. 15a (a8–a10)]. However, the robot in the *normal-density* environment is expected to learn a *"Wait-Forward"* strategy: the robot first waits for a short timeslot until the obstacles starts to move away [Fig. 15b (b1–b4)]. Then it directly forwards to its goal with the fastest speed [Fig. 15b (b5–b10)]. More relevant experiments are conducted to further prove the strategies learnt by the robot, and the results are shown in Fig. 15c, d, in which learnt strategies are illustrated by the trajectory of the robot instead of the separate states of the environment.

According to Fig. 16, in cases with 10 obstacles, the robot controlled by AW-PER-based A2C learnt a fixed policy (Recede-Wait-Forward) as that shown in Fig. 15. However, this fixed policy did not appear on the robot based on AW-based A2C and LSTM-based A2C. AW-based A2C and LSTM-based A2C seem to generate flexible policies, in which most of them are good except for few cases like the second trajectory in Fig. 16a. It is hard for the robot based on LSTM-based A2C to handle the obstacle right approaching the robot. Then the robot gets stuck at the beginning. However, the robot based on AW-based A2C behaves smarter by performing a "Forward-Avoid-Forward" policy (the second trajectory in Fig. 16b) to reach a high reward. In cases with 5 obstacles, AW-PER-based A2C generates a "Wait-Forward" policy as that shown in Fig. 15. However, AW-based A2C and LSTM-based A2C generate different "Recede-Wait-Forward" policies. Their difference is that the robot based on LSTM-based A2C recedes *too far*, while the robot based on AW-based A2C just recedes a short distance. Hence, the robot based on AW-based A2C is expected to achieve higher reward than that of the robot based on LSTM-based A2C.

Computational evaluations

This part considers the *time cost in the training*. Detailed comparisons of time cost (hour) are shown in Table 9. Note that just *single thread* is used in these three algorithms for data collection. According to the results, AW-PER-based algorithm converges great faster (0.1 h) than the rest two algorithms (near 1 h) in the 1st-stage training, while LSTM-based A2C is more time-efficient than the rest two algorithms in the 2nd and 3rd-stage training. LSTM-based A2C performs almost the same as the AW-PER-based A2C in the time cost of entire training process, but it costs more than 2/1.5 times episodes to converge in cases with 10/5 obstacles when comparing with that of AW-PER-based A2C. AW-based A2C costs the most time in its training either for the case of 10 obstacles or for the case of 5 obstacle, when comparing with the rest two algorithms.

Robustness evaluations (extreme tests)

This part considers evaluating the performance of three algorithms in different environment (the square-crossing environment) to test the robustness of these algorithms. Note that the behaviors of the obstacle in the square-crossing environment are far different from that in the circle-crossing environment. Much noise is also added to further increase the random attribution of the obstacle in the square-crossing environment. Policies of three algorithms are trained in the circle-crossing environment. These policies are tested in the

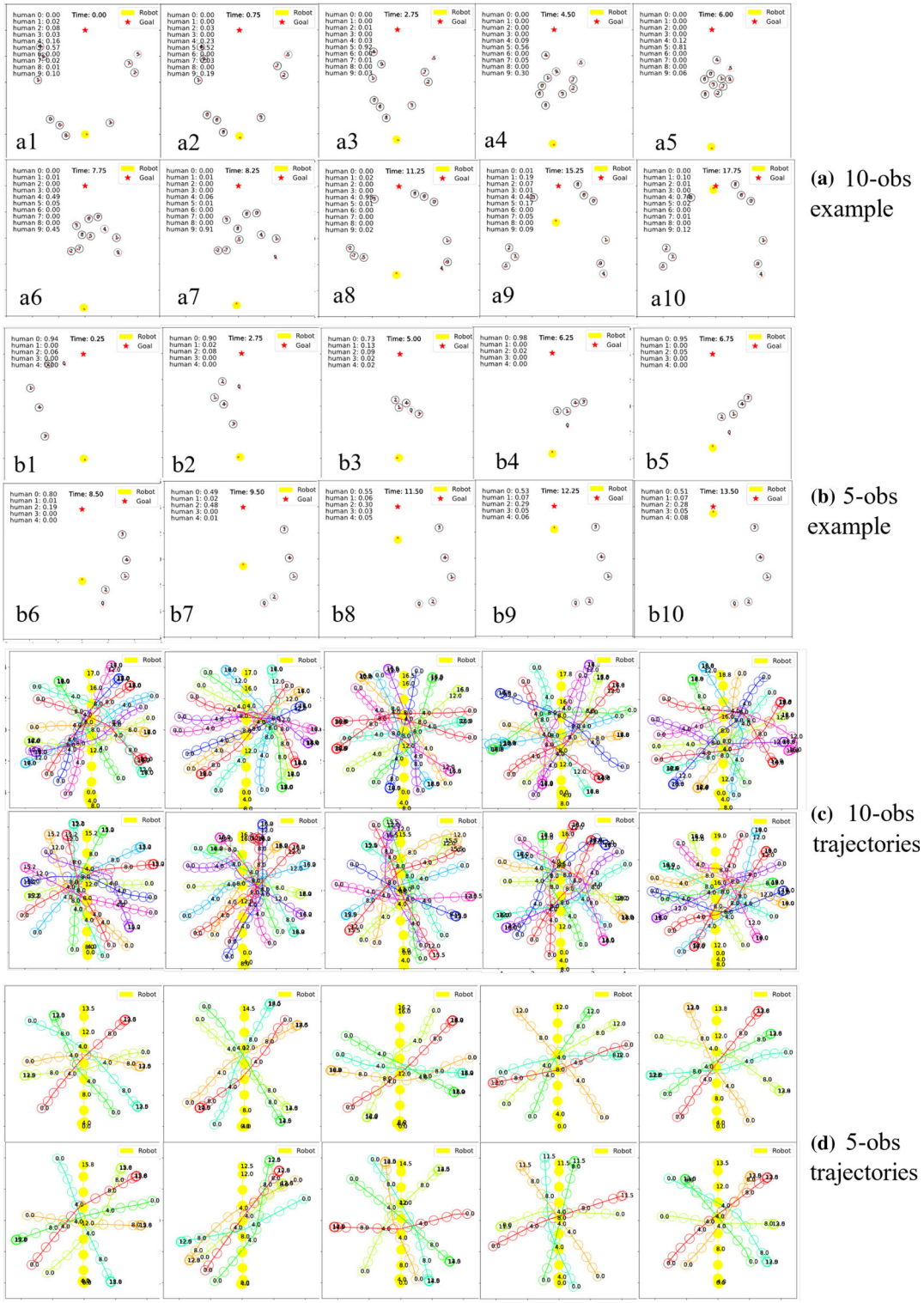


Fig. 15 The strategies the robot learnt. **a**, **b** show examples of AW-PER-based A2C in one episode, while **c**, **d** present trajectories of the robot based on AW-PER-based A2C in different test cases with 10/5 obstacles. Note that distances of robot-obstacle are assumed to be significantly larger than the radius of the robot and obstacles. Obstacles refer to walking pedestrians in indoor scenario of real world. All environment parameters (e.g., radius and speed of obstacles and the robot, the distance of the start to goal) are set according to the real values or real geometry of the robot and obstacles in indoor scenario. Hence, the trained model is easy to transfer to real world

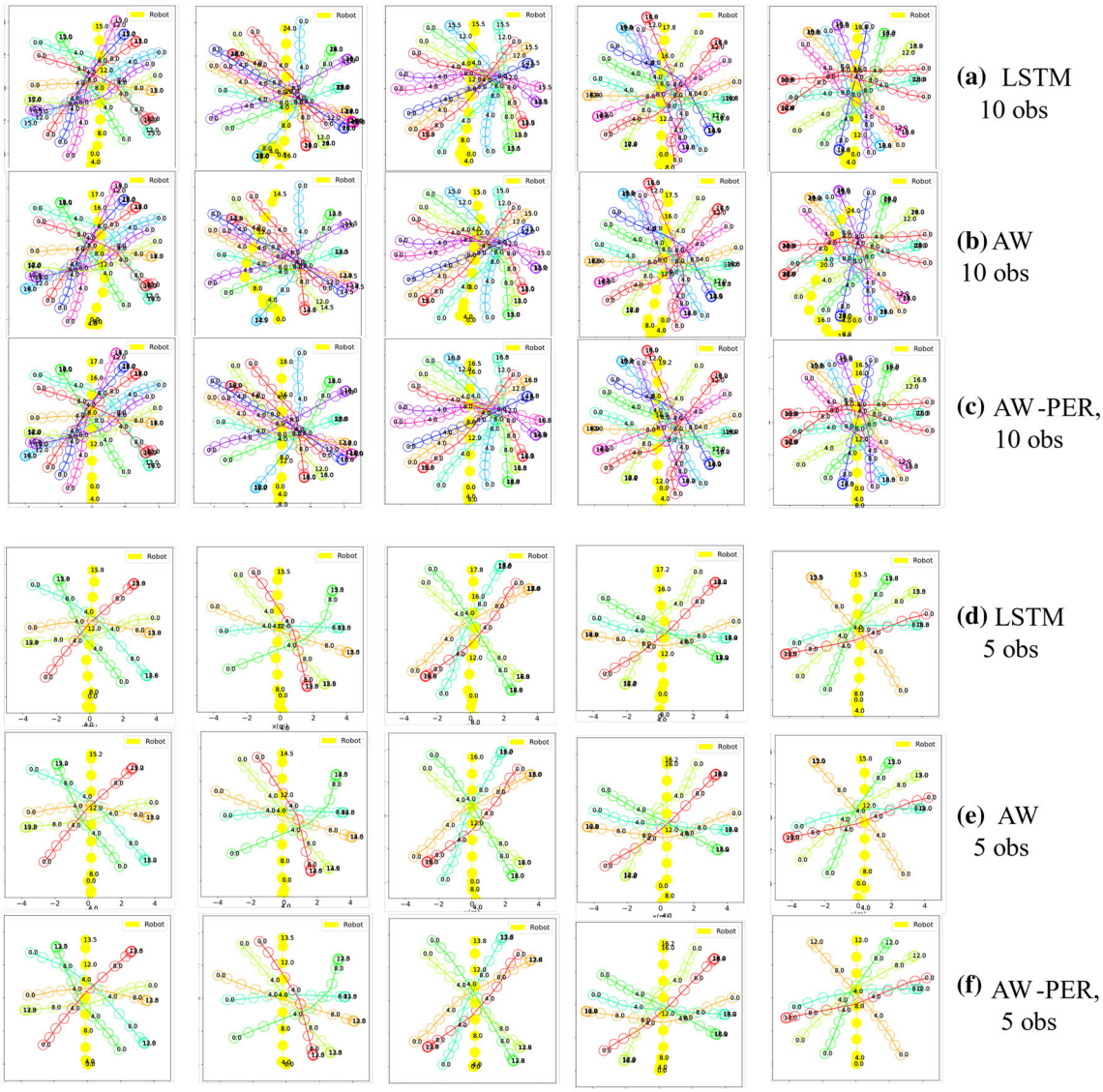


Fig. 16 Trajectory comparisons of three algorithms in the same test cases. **a**–**c** Denotes the cases with 10 obstacles, while **d**–**f** denote the cases with 5 obstacles

Table 9 Time cost of three algorithms in the training for the case of single thread

Algorithms	1st stage (10/5 obs.)	2nd stage (10/5 obs.)	3rd stage (10/5 obs.)	Overall (10/5 obs.)
LSTM-based A2C	2.20/0.65	1.80/0.55	0.80/0.5	4.80/1.70
Our AW-based A2C	3.40/0.55	4.70/1.45	2.00/0.95	10.10/2.95
Our AW-PER-based A2C	**1.25/0.10**	1.90/1.15	1.25/0.70	**4.40**/1.95

Bold values indicate the best performance

square-crossing environment. Six criterions are considered in the robustness evaluations of three algorithms. They consist of the success rate, time to goal, collision rate, timeout rate, mean distance to obstacles and received accumulative reward, like that in the quantitative evaluations. Detailed evaluations are shown in Table 10. According to the results, AW-based A2C outperforms the rest two algorithms on near all criterions, except for the time to goal and timeout rate which are slightly worse than that of AW-PER-based A2C. The performance of AW-PER-based A2C drops faster than the rest two algorithms in the success rate and collision rate. This also causes the subsequent low accumulative rewards

Table 10 Comparison of three algorithms in the square-crossing simulator (the extreme tests)

Algorithm	Success rate (10/5 obs.)	Time to goal (10/5 obs.)	Collision rate (10/5 obs.)	Timeout rate (10/5 obs.)	Mean distance to obs. (10/5 obs.)	Rewards (10/5 obs.)
LSTM-based A2C	0.29/0.49	16.65 s/15.51 s	0.48/0.39	0.23/0.12	0.12 m/0.11 m	0.0478/0.1473
Our AW-based A2C	**0.39/0.54**	**16.20 s/13.16 s**	**0.39/0.34**	0.22/**0.12**	0.12 m/0.11 m	**0.1362/0.1562**
Our AW-PER-based A2C	0.21/0.41	**15.39 s/**13.33 s	0.63/0.46	**0.16/**0.13	0.11 m/0.11 m	0.0187/0.0687

Bold values indicate the best performance

Table 11 Experience cost and the reward received in the 2nd-stage training

Algorithms	Episode cost (2nd-stage, 10/5 obs.)	Reward (2nd-stage, 10/5 obs.)
LSTM-based A2C	28,000/15000	0.23/0.32
Our AW-based A2C	35,000/22000	0.23/0.35
Our AW-PER-based A2C	15,000/18000	0.27/0.36

received. The relatively lower robustness of AW-PER-based A2C is caused by the PER. The PER requires less data to make the A2C converged, but it also leads to the less explorations for finding more potential actions. These actions may be useless to improve the performance in the circle-crossing environment but useful in the square-crossing environment, hence resulting in a relatively low performance of the robot in a completely new environment.

Discussion and future work

This section first analyzes the problems found in the experiment. Then some future research directions will be discussed. Some problems came up during the experiments, for example, the slow convergence speed of AW-based A2C in the 2nd-stage training and the distribution drift caused by PER.

Problem 1

Let's recall some results of 2nd-stage training from Table 6 that is partially shown in Table 11. LSTM-based A2C and AW-based A2C reach the same reward 0.23 (0.1 to 0.23) in cases with 10 obstacles. LSTM-based A2C spends 28,000 episodes to reach that, while AW-based A2C costs more episodes (35,000). Experiments of LSTM-based A2C and AW-based A2C in cases with 5 obstacles also follow the same trend. We guess this problem may relate to the architecture of AW network which consists of four layers with a complex connection among layers (Fig. 4), while LSTM network consists of merely one recurrent layer. More layers with complex connections in the neural network sometimes cause the *vanishing gradient* problem in the backpropagation process (Kolbusz et al., 2017). This means the early layers next to the input layer are expected to receive less gradient to update their weight, hence causing the slow update or even zero update of weight. The more layers are, the less the gradient is received for the early layers in the backpropagation process. However, AW-PER-based A2C better reduces the impact from the vanishing gradient problem by providing a pretrained model which leads to a high success rate (near 0.4) in the motion planning of the robot in the 2nd-stage training. A better success rate means a higher reward and gradient, therefore a higher gradient is received in the early layers of AW-based A2C, although these layers are slightly impacted by the vanishing gradient.

Problem 2

Distribution drift problem is reflected in the way of weight update which causes three consequences: (1) weight update is sensitive to a larger learning rate; (2) the final converged policy is slightly unpredictable; (3) relatively low robustness of AW-PER-based A2C.

The first consequence is shown in Fig. 17, in which a pretrained model is retrained in the 2nd-stage training. A lager learning rate (3e−4) is applied in this period of training. However, the AW-PER-based model (green line) converges from a worse point (reward = − 0.9), instead of the training result of pretrained model (reward = 0.1). The larger learning rate is then replaced by a smaller one (3e−5), therefore this problem is better solved.

The second consequence is shown in the 3rd-stage reward received by AW-based A2C and AW-PER-based A2C in Table 6 that is also partially shown in Table 12. In the 2nd and 3rd-stage training, AW-PER-based A2C updates its weight in an online manner like that of AW-based A2C. Intuitively, these two algorithms may converge to a same reward. However, results of 3rd-stage training show that their converged rewards are slightly different. Learnt policies by the robot

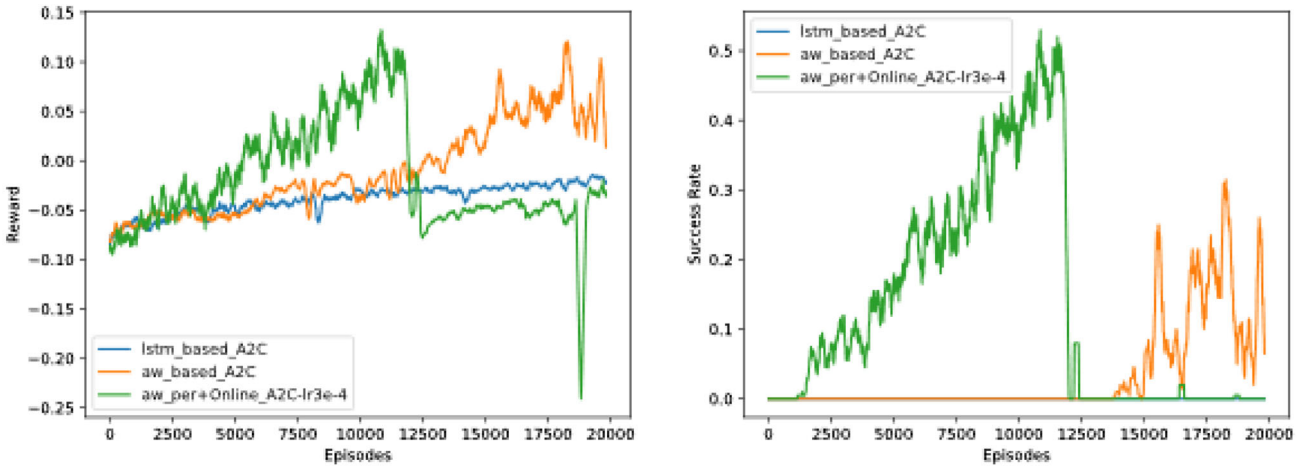


Fig. 17 The effect of the distribution drift on the received reward (left) and success rate (right)

Table 12 The reward received by the robot in 3rd-stage training

Algorithms	Reward (3rd-stage, 10/5 obs.)
Our AW-based A2C	0.33/0.38
Our AW-PER-based A2C	0.30/0.42

Table 13 The success rate and collision rate in the extreme test

Algorithms	Success rate (10/5 obs.)	Collision rate (10/5 obs.)
LSTM-based A2C	0.29/0.49	0.48/0.39
Our AW-based A2C	0.39/0.54	0.39/0.34
Our AW-PER-based A2C	0.21/0.41	0.63/0.46

differ slightly as well. For example, the robot using AW-PER-based A2C is expected to choose the "Recede" and "Wait" strategies in the early-stage of motion planning with 10 obstacles. However, the robot based on AW-based A2C is apt to select "Follow" and "Wait" strategies. In cases with 5 obstacles, the robot based on AW-PER-based A2C is likely to choose the "Wait" strategy at the beginning, while the robot using AW-based A2C likes "Recede" strategy. Higher reward is expected to be received once the "Follow" strategy is selected by the robot, while it is hard to obtain better reward if the robot recedes at the beginning.

The third consequence is shown in Table 10 which is partially shown in Table 13. The performance of AW-PER-based A2C drops slightly faster than the AW-based A2C and LSTM-based A2C in the success rate and collision rate of robustness evaluations (extreme tests). We find that these three consequences are caused by PER which introduces the bias to the process of weight update in the training by changing the data distribution, therefore changing the solution or policy the network should converge to in the online learning (Schaul et al., 2016). The importance-sampling weight reduces the impact caused by the distribution drift, but it cannot eliminate it, therefore slight changes are found in the converged policies of AW-based A2C and AW-PER-based A2C. Moreover, the AW-PER-based A2C costs less data (35,000/30000 episodes) to converge, while AW-based A2C and LSTM-based A2C spend 75,000/45000 episodes to reach the convergence. This means less explorations are done in the AW-PER-based A2C when comparing with the rest two algorithms, therefore the converged policy of AW-PER-based A2C slightly lacks flexibility and robustness in the qualitative evaluations and robustness evaluations respectively. In summary, algorithms based on PER converge faster and less data is consumed. However, the flexibility and robustness of their converged policies drop slightly especially in challenging scenarios. We believe all algorithms based on the PER in other fields share the same consequence caused by the PER in the flexibility and robustness of the policy. Hence, how to find a better trade-off between the advantage and disadvantage of PER matters.

Future research directions

Future works are expected to focus on four aspects: (1) methods for better feature interpretation; (2) methods to reduce the distribution drift; (3) combination of global path planning algorithms with local motion planning algorithms; (4) 3-dimension real-world implementation.

Recent relation graph performs robust as well in the description of environmental states (Chen et al., 2019a). Impact from the distribution drift problem is likely to be further reduced by changing the way to store and sample

the data in the replay buffer (Bu & Chang, 2020). However, algorithms discussed in this paper are better to solve the local motion planning problems. It cannot replace the global path planning algorithms for the outdoor path planning tasks. It would be interesting to combine our local motion planning with global path planning algorithms for more challenging tasks, such as path/motion planning in large and complex airport like Daxin airport in Beijing. Our work does not consider the complex 3-dimention case. 3-dimention implementation would be more challenging because of the irregular or complex shapes and geometries of obstacles. If our algorithm is applied into 3-dimention case directly, this will cause many potential problems. For instance, collision detection. Our work simplifies the obstacle shape as the circle with dynamic radius. The constraint of collision detection is obtained by computing the minimum distance of two agents ($d_{min} = r_{robot} + r_{obstacle}$). If the distance of the robot and obstacle $D < d_{min}$, the robot and obstacle collide. It is not enough in the 3-dimention case. Our future work in 3-dimention implementation will consider fusing other method (Redon et al., 2005; Husty et al., 2007; Barton et al., 2009; Choi et al., 2009) to ensure an accurate collision detection constraint. In real-world implementation, unexpected fault, influence of disturbances, and other uncertainties in real system are challenges that cannot be ignored. Future work will also consider the uncertainties of real-world implementation, especially the fault detections (Cheng et al., 2021; Dong et al., 2020).

Conclusion

This paper first implements the LSTM-based A2C without the expert experience by making the modification on the reward function, but LSTM-based A2C suffers slow convergence speed and over-fitting in the training. It is followed by applying the AW encoder to replace the LSTM encoder to better describe the environmental state, hence the problems in LSTM-based A2C are reduced. The convergence speed of AW-based A2C is then further improved by combining the online learning and batch learning which is based on the PER. As the results, AW-PER-based A2C takes only near 15% and 30% of data to get rid of the early-stage training. It spends around 45% and 65% of data to reach the convergence when comparing with LSTM-based A2C in cases with 10 and 5 obstacles. AW-PER-based A2C converges to almost the same reward as that of LSTM-based A2C and AW-based A2C in cases with 10 and 5 obstacles (even better in cases with 5 obstacles) at the expenses of slightly sacrificing its robustness in the extreme test (robustness evaluations). Our AW-PER-based A2C and AW-based A2C are easy to be applied into the real motion planning tasks once the features of the agent (the robot and obstacles) are acquired by the sensors [e.g., *light detection and ranging* (liDAR), depth camera and encoder]. For example, position, velocity and moving direction are obtained by the liDAR and encoder. Radius is obtained by the depth camera, while goal's position and preferred velocity are set artificially.

Author contributions CZ, PF, HH, BH: conceptualization; CZ: methodology; CZ: formal analysis and investigation; CZ: writing—original draft preparation; PF, BH, HH: writing—review and editing; CZ, HH, BH, PF: funding acquisition, resources; PF, BH: Supervision.

Funding Open access funding provided by University of Eastern Finland (UEF) including Kuopio University Hospital. The authors did not receive support from any organization for the submitted work.

Data availability Our source code is available on the website (https://github.com/CHUENGMINCHOU /AW-PER-A2C).

Declarations

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Consent to participate Not applicable.

Consent to publish Not applicable.

Ethical approval Not applicable.

References

Bai, Z., Cai, B., Shangguan, W., & Chai, L. (2019). Deep learning based motion planning for autonomous vehicle using spatiotemporal LSTM network. In *Proceedings 2018 Chinese Automation Congress, CAC 2018* (pp. 1610–1614). https://doi.org/10.1109/CAC.2018.8623233

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. *Machine Learning Proceedings, 1995*, 30–37. https://doi.org/10.1016/b978-1-55860-377-6.50013-x

Barton, M., Shragai, N., & Elber, G. (2009). Kinematic simulation of planar and spatial mechanisms using a polynomial constraints solver. *Computer-Aided Design and Applications, 6*(1), 115–123. https://doi.org/10.3722/cadaps.2009.115-123

Bas, E. (2019). An introduction to Markov chains. *Basics of Probability and Stochastic Processes*. https://doi.org/10.1007/978-3-030-32323-3_12

Brownlee, J. (2018). Better deep learning: train faster, reduce overfitting, and make better predictions. *Machine Learning Mastery*. Retrieved from https://machinelearningmastery.com/better-deep-learning/.

Bry, A. & Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proceedings—IEEE international conference on robotics and automation*. https://doi.org/10.1109/ICRA.2011.5980508.

Bu, F. & Chang, D. E. (2020). Double prioritized state recycled experience replay. In *2020 IEEE international conference on consumer electronics—Asia, ICCE-Asia 2020*. https://doi.org/10.1109/ICCE-Asia49877.2020.9276975.

Chen, Y. F., Liu, M., Everett, M. & How, J. P. (2017). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *Proceedings—IEEE international conference on robotics and automation* (pp. 285–292). https://doi.org/10.1109/ICRA.2017.7989037.

Chen, C., Hu, S., Nikdel, P., Mori, G. & Savva, M. (2019a). Relational graph learning for crowd navigation. *ArXiv*. http://arxiv.org/abs/1909.13165.

Chen, C., Liu, Y., Kreiss, S., & Alahi, A. (2019b). Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *Proceedings—IEEE international conference on robotics and automation* (pp. 6015–6022). https://doi.org/10.1109/ICRA.2019.8794134

Cheng, P., Wang, H., Stojanovic, V., He, S., Shi, K., Luan, X., Liu, F., & Sun, C. (2021). Asynchronous fault detection observer for 2-D Markov jump systems. *IEEE Transactions on Cybernetics*. https://doi.org/10.1109/TCYB.2021.3112699

Choi, Y. K., Chang, J. W., Wang, W., Kim, M. S., & Elber, G. (2009). Continuous collision detection for ellipsoids. *IEEE Transactions on Visualization and Computer Graphics, 15*(2), 311–324. https://doi.org/10.1109/TVCG.2008.80

Dong, X., He, S., & Stojanovic, V. (2020). Robust fault detection filter design for a class of discrete-time conic-type non-linear Markov jump systems with jump fault signals. *IET Control Theory & Applications, 14*(14), 1912–1919. https://doi.org/10.1049/iet-cta.2019.1316

Everett, M., Chen, Y. F., & How, J. P. (2018). Motion planning among dynamic, decision-making agents with deep reinforcement learning. *IEEE International Conference on Intelligent Robots and Systems, Iii*. https://doi.org/10.1109/IROS.2018.8593871

Farouki, R. T., & Sakkalis, T. (1994). Pythagorean-hodograph space curves. *Advances in Computational Mathematics, 2*(1), 41–66. https://doi.org/10.1007/BF02519035

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine, 4*(1), 23–33. https://doi.org/10.1109/100.580977

Funke, J., Theodosis, P., Hindiyeh, R., Stanek, G., Kritatakirana, K., Gerdes, C., Langer, D., Hernandez, M., Müller-Bessler, B., & Huhnke, B. (2012). Up to the limits: Autonomous Audi TTS. *IEEE Intelligent Vehicles Symposium, 2012*, 541–547. https://doi.org/10.1109/IVS.2012.6232212

González, D., Pérez, J., Lattarulo, R., Milanés, V. & Nashashibi, F. (2014). Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. In *2014 17th IEEE international conference on intelligent transportation systems, ITSC 2014* (pp. 1430–1435). https://doi.org/10.1109/ITSC.2014.6957887.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT press, Cambridge, MA. Retrieved from http://www.deeplearningbook.org.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Husty, M. L., Pfurner, M., & Schröcker, H.-P. (2007). A new and efficient algorithm for the inverse kinematics of a general serial 6R manipulator. *Mechanism and Machine Theory, 42*(1), 66–81. https://doi.org/10.1016/j.mechmachtheory.2006.02.001

Inoue, M., Yamashita, T., & Nishida, T. (2019). Robot path planning by LSTM network under changing environment. *Advances in Intelligent Systems and Computing, 759*, 317–329. https://doi.org/10.1007/978-981-13-0341-8_29

Jiang, M., Grefenstette, E. & Rocktäschel, T. (2020). Prioritized level replay. *ArXiv*. http://arxiv.org/abs/2010.03934.

Kolbusz, J., Rozycki, P., & Wilamowski, B. M. (2017). The study of architecture MLP with linear neurons in order to eliminate the "vanishing gradient" problem BT—artificial intelligence and soft computing. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, & J. M. Zurada (Eds.), *International conference on artificial intelligence and soft computing 2017: Artificial intelligence and soft computing* (pp. 97–106). Springer. https://doi.org/10.1007/978-3-319-59063-9_9

Konda, V. R. & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014).

Li, A. A., Lu, Z. & Miao, C. (2021). Revisiting prioritized experience replay: A value perspective. *ArXiv*. http://arxiv.org/abs/2102.03261.

Lin, Z., Feng, M., Dos Santos, C. N., Yu, M., Xiang, B., Zhou, B. & Bengio, Y. (2017). A structured self-attentive sentence embedding. In *5th international conference on learning representations, ICLR 2017—conference track proceedings* (pp. 1–15).

Lippmann, R. P. (1988). An introduction to computing with neural nets. *ACM SIGARCH Computer Architecture News, 16*(1), 7–25. https://doi.org/10.1145/44571.44572

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing atari with deep reinforcement learning, pp. 1–9. *ArXiv*, http://arxiv.org/abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proeedings of machine learning research* (Vol. 48, pp. 1928–1937). PMLR. http://proceedings.mlr.press/v48/mniha16.pdf

Munos, R., Stepleton, T., Harutyunyan, A., & Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. *Advances in Neural Information Processing Systems, Nips, 26*, 1054–1062.

Oh, J., Guo, Y., Singh, S., & Lee, H. (2018). Self-Imitation Learning. *35th International Conference on Machine Learning, ICML, 9*(2), 6214–6223.

Redon, S., Lin, M. C., Manocha, D., & Kim, Y. J. (2005). Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering, 5*(2), 126–137. https://doi.org/10.1115/1.1884133

Reed, R. & MarksII, R. J. (1999). *Neural smithing: Supervised learning in feedforward artificial neural networks*. MIT Press. Retrieved from https://mitpress.mit.edu/books/neural-smithing.

Reeds, J. A., & Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics, 145*(2), 367–393. https://doi.org/10.2140/pjm.1990.145.367

Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2016). Prioritized experience replay. In *4th international conference on learning representations, ICLR 2016—conference track proceedings* (pp. 1–21).

Schulman, J., Levine, S., Moritz, P., Jordan, M. & Abbeel, P. (2015). Trust region policy optimization. In *32nd international conference on machine learning, ICML 2015*, (vol. 3, pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. (pp. 1–12) *ArXiv*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *31st international conference on machine learning, ICML 2014*, (vol. 1, pp. 605–619).

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing, 30*(8), 2133–2147. https://doi.org/10.1080/01431160802549278

Van Den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. *Proceedings—IEEE International Conference on Robotics and Automation, 2*(4), 100–107. https://doi.org/10.1109/ROBOT.2008.4543489

Van Hasselt, H., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double Q-Learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, (pp. 2094–2100).

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Frcitas, N. (2016). Dueling network architectures for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016, 4*(9), 2939–2947.

Wang, Z., Mnih, V., Bapst, V., Munos, R., Heess, N., Kavukcuoglu, K. & De Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *5th international conference on learning representations, ICLR 2017—conference track proceedings*, *2016*. Retrieved from https://static.aminer.cn/upload/pdf/239/1521/964/58d82fc8d649053542fd5854.pdf.

Xu, W., Wei, J., Dolan, J. M., Zhao, H., & Zha, H. (2012). A real-time motion planner with trajectory optimization for autonomous vehicles. *IEEE International Conference on Robotics and Automation, 2012*, 2061–2067. https://doi.org/10.1109/ICRA.2012.6225063

Zha, D., Lai, K. H., Zhou, K. & Hu, X. (2019). Experience replay optimization. In *IJCAI international joint conference on artificial intelligence* (Vols. 2019-Augus). https://doi.org/10.24963/ijcai.2019/589