Representation Learning and Reinforcement Learning for Dynamic Complex Motion Planning System

Chengmin Zhou, *Student Member, IEEE*, Bingding Huang, and Pasi Fränti, *Senior Member, IEEE*

***Abstract*— Indoor motion planning challenges researchers because of the high density and unpredictability of moving obstacles. Classical algorithms work well in the case of static obstacles but suffer from collisions in the case of dense and dynamic obstacles. Recent reinforcement learning (RL) algorithms provide safe solutions for multiagent robotic motion planning systems. However, these algorithms face challenges in convergence: slow convergence speed and suboptimal converged result. Inspired by RL and representation learning, we introduced the ALN-DSAC: a hybrid motion planning algorithm where attention-based long short-term memory (LSTM) and novel data replay combine with discrete soft actor–critic (SAC). First, we implemented a discrete SAC algorithm, which is the SAC in the setting of discrete action space. Second, we optimized existing distance-based LSTM encoding by attention-based encoding to improve the data quality. Third, we introduced a novel data replay method by combining the online learning and offline learning to improve the efficacy of data replay. The convergence of our ALN-DSAC outperforms that of the trainable state of the arts. Evaluations demonstrate that our algorithm achieves nearly 100% success with less time to reach the goal in motion planning tasks when compared to the state of the arts. The test code is available at https://github.com/CHUENGMINCHOU/ALN-DSAC.**

***Index Terms*— Intelligent robot, motion planning, reinforcement learning (RL), representation learning.**

I. Introduction

INDOOR service robots appeared in airports and restaurants to provide services to visitors (e.g., luggage delivery and food delivery). However, these robots suffer poor motion planning performance in scenarios with dense and dynamic obstacles (pedestrians) because of obstacle's motion unpredictability. This barricades robot's further commercial use. Some robot's motions are controlled by classical path planning algorithms, such as the graph search algorithm (e.g., A* [1]), sample-based algorithm (e.g., the rapidly exploring random tree (RRT) [2]), and interpolating curve algorithms [3], [4], [5], [6], [7]. These algorithms work well in static and low-speed less-obstacle scenarios. However, they make robots suffer many collisions in complex cases because they generate the motions or paths online. Online motion depends on map update that requires much computing resource. Reaction-based algorithms, such as dynamic window approach (DWA) [8] and optimal reciprocal collision avoidance (ORCA) [5], perform fast to handle obstacle's unpredictability. This enables the robot to fast avoid slow-speed obstacles. However, these algorithms require environment update in which consumed time must be considered. Deep learning (DL) generates robot's motion by performing trained models in which consumed time can be ignored. Classical DL, such as convolutional neural network (CNN) [9], generates instant motion, which is a one-step prediction and does not consider task goal, therefore obtaining suboptimal trajectories. Recent progress in deep reinforcement learning (RL), such as optimal value RL (e.g., dueling deep Q network (DQN) [10]) and policy gradient RL (e.g., asynchronous advantage actor–critic (A3C) algorithm [11]), enables robot to consider task goal and instant obstacle avoidance simultaneously. These algorithms provide near-optimal solutions to handle complex cases. However, training of RL algorithms faces many challenges: slow convergence speed and suboptimal converged result caused by high bias and variance.

In this article, we conclude that data quality, efficacy of data replay strategies, and optimality of RL algorithms are the main aspects, which decide the performance of RL-based motion planning.

Data Quality (Representation Methods): Bai et al. [9] and Long et al. [12] learned obstacle features directly from source images, which are with poor data quality, because of background noise or unnecessary information. Representation learning [13] partly alleviates this problem by interpretating and encoding the feature of the robot and obstacles to improve data quality. Representation methods that fit motion planning tasks are the long short-term memory (LSTM) [14], [15], attention weight (AW) [16], [17], and some graph representation learning methods [18] such as relation graph (RG) [19].

Efficacy of Data Replay: The data replay strategy makes use of saved data to improve the convergence speed dramatically when it is compared to online learning [14], which learns episodic data with online feature. Typical data replay strategy consists of the experience replay (ER) [20] and prioritized experience replay (PER) [21], [22]. ER introduces less bias and variance when the algorithm learns from a random batch

Manuscript received 5 October 2022; revised 30 December 2022; accepted 13 February 2023. *(Corresponding authors: Pasi Fränti; Bingding Huang.)*

Chengmin Zhou is with the Machine Learning Group, School of Computing, University of Eastern Finland, 80100 Joensuu, Finland, and also with the College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China (e-mail: zhou@cs.uef.fi).

Bingding Huang is with the College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China (e-mail: huangbingding@sztu.edu.cn).

Pasi Fränti is with the Machine Learning Group, School of Computing, University of Eastern Finland, 80100 Joensuu, Finland (e-mail: franti@cs.uef.fi).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2023.3247160.

Digital Object Identifier 10.1109/TNNLS.2023.3247160

of data because sampled data for training have the same data distribution, independently identical distribution (i.i.d.).

Optimality of RL: RL algorithms basically include the optimal value RL and policy gradient RL. They are primarily tested in games such as Atari game. Their representatives are DQN [20] and actor–critic algorithm [23]. Then, many variants follow. DQN evolves into double DQN [24], dueling DQN [10], and soft Q-learning [25], while the actor–critic algorithm basically evolves from three directions: multithread direction, deterministic direction, and monotonous direction. Multithread direction denotes using a multiple-thread method and policy entropy to accelerate the convergence speed. Examples are A3C and A2C [11]. Deterministic direction proves that the policy to select actions is stable or deterministic in one state s and actions are directly decided by this state $a \leftarrow \mu_\theta(s)$, while its counterpart, the stochastic policy, selects actions by the possibility $a \leftarrow \pi_\theta(a \mid s)$. Examples are deterministic policy gradient (DPG) [26] and deep DPG (DDPG) [27]. Monotonous direction introduces the trust region constraint, surrogate, and adaptive penalty to ensure the monotonous update of policy. Examples are trust region policy optimization (TRPO) [28] and proximal policy optimization (PPO) [29]. Currently, the double Q-learning actor–critic is the most efficient architecture. It is applied to delayed DDPG (TD3) [30] and soft actor–critic (SAC) algorithm [31], [32], [33], [34].

Technical Difficulties: The mentioned methods above have many weaknesses.

1) In representation methods, LSTM suffers a suboptimal encoding strategy when encoding obstacle features. Poor and suboptimal orders (e.g., random order and the order by distance of robot and obstacle [14]) cause suboptimal converged result. Networks of AW and RG suffer from slow convergence.

2) In the data replay, learning from stored batch data in ER ignores the online feature of episodic data and importance of each data [35]. PER also lacks the consideration of online feature of episodic data, but it considers the importance of each data by the data prioritization. Prioritization in PER finds a better tradeoff between stochastic sampling and greedy sampling. However, this prioritization changes data distribution. Hence, the bias is introduced in PER, although importance sampling weight (ISW) [21] is applied to partly alleviate this problem.

3) In RL algorithms, much bias and variance are introduced even in a deterministic case such as DDPG, although many tricks are used to reduce them, for instance, double Q network [24] and advantage architecture [10] to reduce the overestimation of Q value, and one-step/multistep actor–critic architecture to reduce the variance. Moreover, many RL algorithms, such as A3C and A2C, are not data-efficient. They are on-policy algorithms, which require more data for training.

Existing trainable motion planning algorithms have many weaknesses in mentioned three aspects. Trainable motion planning methods include RG [19], PPO with multiple robots [12], [29], CADRL [36], LSTM-A2C [11], LSTM-RL [14], and SARL [16]. RG is the combination of RG and DQN. Relation graph and DQN face the problems of data quality and optimality of RL. It is hard to train the graph network although RG can represent robot–obstacle relationship. DQN brings high bias and variance, which causes slow convergence. PPO with multiple robots faces problems of data quality because it learns obstacle features from source images with much noise. CADRL learns the pairwise feature of the robot and one obstacle by DQN. Then, the trained model is applied to multiple-obstacle case [36]. It faces the problems of data quality and optimality of RL because it is myopic and uses the closest obstacle feature for training instead of all obstacle features. DQN in CADRL also brings high bias and variance. LSTM-A2C and LSTM-RL face three problems simultaneously because LSTM encodes the obstacle features by distance-based order [14], which partly represents robot–obstacle relationship. A2C/A3C lack data replay. A2C/A3C and DQN bring high bias and variance. SARL consists of AW and DQN where the attention network interprets robot–obstacle features to efficacious neural network weight [16]. However, it faces the problem in RL optimality because DQN brings high bias and variance.

Motivation: To address mentioned problems.

1) We first implemented discrete SAC (DSAC) algorithm to improve the optimality of RL. The architecture of DSAC is the double Q-learning actor–critic, which is the most efficient architecture currently.

2) We optimized existing distance-based LSTM encoding to improve the data quality. Distance-based LSTM encodes pairwise robot–obstacle features by a distance of the robot and obstacle. It represents the robot–obstacle relationship partly; therefore, it is improved by attention-based encoding, which computes the attention-based importance of obstacles to decide the way of encoding robot–obstacle features.

3) We introduced a novel data replay method by combining online learning and offline learning to improve the efficacy of data replay. Existing RL algorithms are fed with either online experience or offline experience. We attempted to fuse them to create a new experience. Moreover, online learning and offline learning coexist in our novel data replay method. RL is fed with new experience when RL learns online, while it is fed experience sampled from replay buffer when RL learns offline.

Contribution and Benchmarks: In short, our contribution is ALN-DSAC (Fig. 1), which features: 1) the implementation of DSAC; 2) attention-based LSTM encoding; and 3) novel data replay method. The benchmarks include trainable motion planning algorithms CADRL [36], LSTM-A2C [11], LSTM-DQN (LSTMRL) [14], and SARL [16], as well as classical algorithm ORCA, which relies on the relative position and velocity of the robot and obstacles to compute the possible velocity of the robot [5], [37], [38].

Interpretability: ALN-DSAC is interpretable or explainable because it fits human intuitions to learn and make decisions on nonlinear motion planning tasks. This contributes to better hidden features of dynamic robot and obstacles. Features are then learned by explainable RL (DSAC). This means that.

1) Attention network is derived from human attention mechanism; therefore, it is understandable to human. Attention-based LSTM encoding better describes the robot–obstacle relationship by attention-based obstacle importance.

2) Human has better learning performance if it learns from complete experience. Novel data replay better keeps complete, time-sequential, and episodic online experience. This provides high-quality inputs. These two factors contribute to better hidden features.

3) RL is explainable because it derives from the learning process of humans or other creatures. It is a basic reward/punishment-action process, which is understandable to humans.

II. Preliminaries

A. Preliminary of RL

Markov decision process (MDP) is the sequential decision process based on the Markov chain [39]. Markov Chain is defined by a variable set $X = \{X_n : n > 0\}$, where the probability $p(X_{t+1} \mid X_t, \ldots, X_1) = p(X_{t+1} \mid X_t)$. This means that 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 $\langle S, A, P, R \rangle$. S denotes the state, and here, it refers to the state of robot and obstacles. A denotes an action taken by the robot. Action $A = [\theta, v]$ is selected from action space. In this article, actions directions $\theta \in \{0, (\pi/8), \ldots, 2\pi\}$. Speed of each direction $v \in \{0.2, 0.4, \ldots, 1\}$. Hence, the action space consists of 81 actions, including a stop action. P denotes the possibility to transit from one state to the next state. R denotes the reward or punishment received by the robot after executing actions. The reward function in this article is defined by

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

where p_{current} denotes the position of the robot, p_g denotes the position of the goal, $d_{\min}$ denotes the minimum distance of the robot and obstacles, and $d_{\text{start\_to\_goal}}$ denotes the distance of the start to the goal. Our reward function (1) is modified from [16], which cannot work without imitation learning. Equation (1) accelerates the convergence speed by attaching a reward to the final position of the robot. This encourages the robot to approach the goal. Other crucial terms include value, policy, value function, and policy function. The value denotes how good one state is or how good one action is in one state. The value consists of state value (V value) and state–action value (Q value). A value is defined by the expectation of accumulative rewards $V(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+1} + \cdots + \gamma^{T-1} R_T \mid s_t]$ or $Q(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+1} + \cdots + \gamma^{T-1} R_T \mid (s_t, a_t)]$, where γ is a discounted factor. The policy denotes the way to select actions. In the function approximation case, policy is represented by networks. The value function in RL scope is represented by networks to estimate the value of environmental state via the function approximation [40]. The policy function is also represented by neural networks. Actions are selected in an indirect way (e.g., $a \leftarrow \operatorname{argmax}_a R(s,a) + Q(s,a;\theta)$ in DQN [20], [41]) or direct way (e.g., $\pi_\theta : s \to a$ in actor–critic algorithm [23]).

B. Problem Formulation

ORCA introduced a competent simulator that includes dynamic robot and obstacles in a fixed-size 2-D indoor area. The robot and obstacles move toward their goals simultaneously and avoid collisions with each other (Fig. 2). This simulator creates circle- and square-crossing scenarios that add predictable complexity to tasks. Let s represent the state of robot. Let a and v represent the action and velocity of

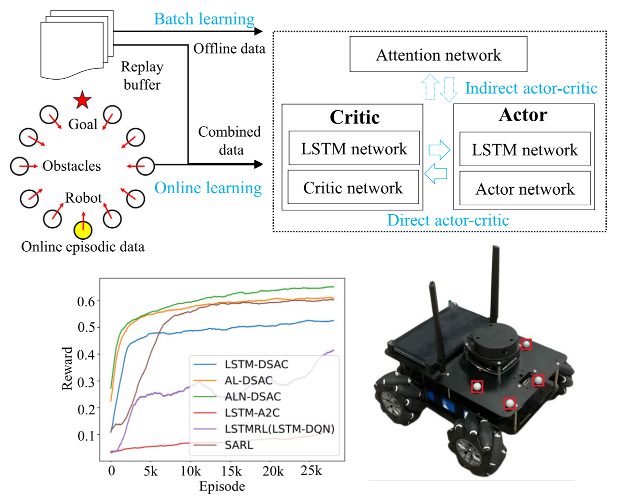

Fig. 1. Workflow of ALN-DSAC. LSTM/AL-DSAC denotes the DSAC with LSTM or attention-based LSTM encoders.

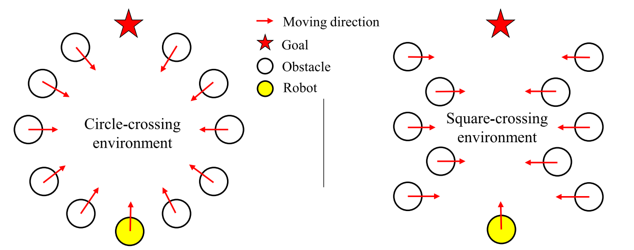

Fig. 2. Circle- and square-crossing simulators. Obstacles are randomly generated near the brink of the circle in a circle-crossing environment. In a square-crossing environment, obstacles are randomly generated on the left side or right side.

robot, respectively, and $a = v = [v_x, v_y]$. Let $p = [p_x, p_y]$ represent the position of robot. Let s_t represent the state of robot at time step t. s_t consists of observable and hidden parts $s_t = [s_t^{\text{obs}}, s_t^h]$, $s_t \in R^9$. Observable part refers to factors that can be measured or observed by others. It consists of position, velocity, and radius $s^{\text{obs}} = [p_x, p_y, v_x, v_y, r]$, $s^{\text{obs}} \in R^5$. The hidden part refers to factors that cannot be seen by others. It consists of planned goal position, preferred speed, and heading angle $s^h = [p_{gx}, p_{gy}, v_{\text{pref}}, \theta]$, $s^h \in R^4$. The state, position, and radius of obstacles are described by $\hat{s}$, $\hat{p}$, and $\hat{r}$, respectively

$$\text{minimize } E\left[t_g \vee s_0, \hat{s}_0^{\text{obs}}, \pi, \hat{\pi}\right] \tag{2}$$

$$\text{s.t. } \|p_t - \hat{p}_t\|_2 \geq r + \hat{r} \quad \forall t \tag{3}$$

$$p_{tg} = p_g \tag{4}$$

$$p_t = p_{t-1} + \Delta t \cdot \pi : \left(s_{0:t}, \hat{s}_{0:t}^{\text{obs}}\right) \tag{5}$$

$$\hat{p}_t = \hat{p}_{t-1} + \Delta t \cdot \hat{\pi} : \left(\hat{s}_{0:t}, s_{0:t}^{\text{obs}}\right). \tag{6}$$

The robot plans its motion by obeying policy : $(s_{0:t}, \hat{s}_{0:t}^{\text{obs}}) \to a_t$, while obstacles obey $\hat{\pi}:(\hat{s}_{0:t}, s_{0:t}^{\text{obs}}) \to a_t$. Robot policy $\pi:(s_{0:t}, \hat{\pi}_{0:t}^{\text{obs}}) \to a_t$ denotes that the algorithm is based on the robot state $s_{0:t}$ and the observable obstacle state $\hat{s}_{0:t}^{\text{obs}}$ to obtain policy π, which outputs action a_t at time step t. Obstacle policy $\hat{\pi}:(\hat{s}_{0:t}, s_{0:t}^{\text{obs}}) \to a_t$ denotes that the algorithm is based on the obstacle state $\hat{s}_{0:t}$ and the observable robot state $s_{0:t}^{\text{obs}}$ to obtain policy $\hat{\pi}$, which outputs action a_t at time step t. The objective of robot is to minimize the time to its goal $E[t_g]$ (2) under the policy π without collisions to obstacles. Constraints of robot's motion planning can be formulated via (3)–(6) that represent collision avoidance constraint, goal constraint, kinematics of robot, and kinematics of obstacle, respectively. Collision avoidance constraint denotes that the distance of robot and obstacles $\|p_t - \hat{p}_t\|_2$ should be greater than or equal to the radius sum of robot and obstacles $r + \hat{r}$. Goal constraint denotes that the position of the robot p_{tg} should be equal to the goal position p_g if robot reaches its goal. Kinematics of robot denotes the position of robot p_t that is equal to the sum of the robot position p_{t-1} and the change of robot position $\Delta t \cdot \pi : (s_{0:t}, \hat{s}^{\text{obs}}_{0:t})$. $\pi : (s_{0:t}, \hat{s}^{\text{obs}}_{0:t})$ is a velocity decided by the policy π. Kinematics of obstacles is the same as that of the robot.

III. Methods

We first present the mechanisms of LSTM, AW, and DSAC. They are fundamental knowledge. Then, we introduce LSTM-DSAC and AL-DSAC that work as the ablations for validating the contributions of DSAC, attention-based LSTM, and novel data replay. Finally, AL-DSAC is optimized by integrating novel data replay to form ALN-DSAC.

A. LSTM, AW, and DSAC

1) LSTM: It encodes pairwise robot–obstacle feature $E^{\text{lstm}} = [e_1^{\text{lstm}}, \ldots, e_N^{\text{lstm}}]$ to form the description of obstacle state S_o^{lstm}. E^{lstm} can be encoded in different orders in LSTM, such as the order by maximum distance, the order by minimum distance [14], and the random order

$$E^{\text{lstm}} = \begin{cases} \left[e^{\text{lstm}}_{d_{\min}}, \ldots, e^{\text{lstm}}_{d_{\max}}\right] \leftarrow \text{rank}_{\min}(d_i), & i \in N \\ \left[e^{\text{lstm}}_{d_{\max}}, \ldots, e^{\text{lstm}}_{d_{\min}}\right] \leftarrow \text{rank}_{\max}(d_i), & i \in N \\ \left[e^{\text{lstm}}_{d_{\text{random}}}, \ldots, e^{\text{lstm}}_{d_{\text{random}}}\right] \leftarrow \text{random}(d_i), & i \in N \end{cases} \quad (7)$$

where d_i denotes the distance of the robot and obstacle, $\text{rank}_{\min}(d_i)$ denotes that E^{lstm} is ordered by minimum distance of the robot and obstacles, and $\text{rank}_{\max}(d_i)$ and $\text{random}(d_i)$ denote that E^{lstm} is ordered by maximum and random distances, respectively. Environment state for training S^{lstm} is defined as the combination of the robot state s_r and obstacle state S_o^{lstm}. S^{lstm} and S_o^{lstm} are defined by

$$S^{\text{lstm}} = \left[s_r, S_o^{\text{lstm}}\right], \quad S_o^{\text{lstm}} = \text{LSTM}\left(E^{\text{lstm}}\right) \quad (8)$$

where the pairwise robot–obstacle feature E^{lstm} consists of pairwise robot–obstacle features e_i^{lstm}. The pairwise robot–obstacle feature is defined as the combination of robot state s_r and the state of each obstacle o_i

$$e_i^{\text{lstm}} = [s_r, o_i], \quad i \in N \quad (9)$$

where N denotes the number of obstacles. Note that s_r and o_i here are the robot-centric states, which are simply transformed from the states described in Section II-B. The robot-centric states are defined by

$$s_r = \left[d_g, v_{\text{pref}}, \theta, r, v_x, v_y\right], \quad s_r \in R^6 \quad (10)$$

$$o_i = \left[p_x, p_y, v_{xi}, v_{yi}, r_i, d_i, r_i + r\right], \quad o_i \in R^7 \quad (11)$$

where d_g denotes the distance of the robot to its goal and o_i denotes the robot-centric observable state of the ith obstacle. Note that the ith denotes the obstacle's order, which is generated randomly in the simulator when an episode of the experiment starts. In o_i, radius sum $r_i + r$ denotes the collision constraint of each obstacle to robot. Radius (r and r_i) and radius sum in the feature definition enable the robot to be fast aware of the safe distance to each obstacle. This contributes to the convergence.

2) Attention Weight: AW-based [16], [42] obstacle feature S_o^{aw} combines with the robot feature s_r to form the environmental state S^{aw} for training

$$S^{\text{aw}} = \left[s_r, S_o^{\text{aw}}\right] \quad (12)$$

where S_o^{aw} is defined by

$$S_o^{\text{aw}} = \sum_{i=1}^{n} [\text{softmax}(\alpha_i)] \cdot h_i \quad (13)$$

where α_i and h_i denote the attention score and the interaction feature of robot and obstacle o_i, respectively, and n denotes the number of obstacles. The interaction feature is defined by

$$h_i = f_h(e_i; w_h) \quad (14)$$

where $f_h(\cdot)$ and w_h denote the multiple-layer perceptron (MLP) and its weight, respectively. e_i denotes the embedded feature obtained from the pairwise robot–obstacle feature $[s_r, o_i]$ or $[s_r, o_i, M_i]$. The attention score is defined by

$$\alpha_i = f_\alpha(e_i, e_{\text{mean}}; w_a) \quad (15)$$

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

$$e_i = f_e(s_r, o_i, M_i; w_e), \quad i \in N \quad (16)$$

$$e_{\text{mean}} = \frac{1}{n}\sum_{i=1}^{n} e_i \quad (17)$$

where $f_e(\cdot)$ and w_e denote MLP and its weight, respectively. M_i denotes the occupancy map of obstacle o_i and it is defined by

$$M_i(a, b, :) = \sum_{j \in \mathcal{N}_i} \delta_{ab}\left[x_j - x_i, y_j - y_i\right] \cdot w'_j$$
$$w'_j = \left(v_{xj}, v_{yj}, 1\right) \quad (18)$$

where w'_j is a local state vector of obstacle o_j and N_i denotes other obstacles near the obstacle o_i. The indicator function $\delta_{ab}[x_j - x_i, y_j - y_i] = 1$ if $(x_j - x_i, y_j - y_i) \in (a, b)$ where (a, b) is a 2-D cell [16]. The mechanism of AW is shown in Fig. 3.

3) DSAC: The policy of classical RL algorithm is obtained by maximizing the objective $\sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi}[r(s_t, a_t)]$. SAC considers the reward and entropy simultaneously. The objective of SAC is defined as the maximum entropy objective

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi}[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t))], \mathcal{H}(\pi(\cdot \mid s_t))$$
$$= -\log \pi(\cdot \mid s_t) \quad (19)$$

where $\mathcal{H}(\pi(\cdot \mid s_t))$ denotes the entropy. α is the temperature parameter, which decides the importance of the entropy versus the reward for controlling the stochasticity of optimal policy. This means that the temperature further encourages the explorations to find promising avenues and give up unpromising avenues. In objective maximization, the SAC policy converges to optimal policy certainly by the soft policy iteration, which consists of policy evaluation and policy improvement. The optimal policy is obtained by repeatedly applying policy evaluation and policy improvement. Policy evaluation [32] proves that if $Q^{k+1} = \mathcal{T}^\pi(Q^k)$, Q^k will converge to the soft Q

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

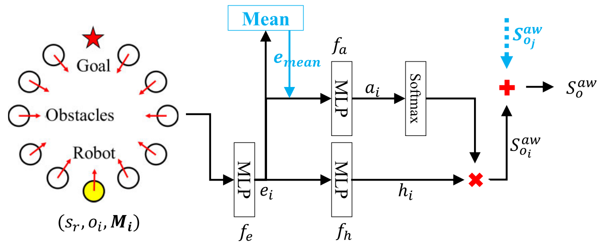

Fig. 3. Mechanism of AW. Here, we present the mechanism to compute the attention-based feature of one obstacle $S_{o_i}^{\text{aw}}$. It combines with the features of other obstacle $S_{o_j}^{\text{aw}}$, $j \in N$, to form the feature of obstacles S_o^{aw}.

value of π when $k \to \infty$. $\mathcal{T}^\pi$ is a modified Bellman backup operator given by

$$\mathcal{T}^\pi(Q)(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}[V(s_{t+1})] \quad (20)$$

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log \pi(a_t|s_t)]. \quad (21)$$

Applying $\mathcal{T}^\pi$ to Q value will bring Q value closer to Q^π. This means that $Q(s_t, a_t) \leq \mathcal{T}^\pi(Q)(s_t, a_t) \leq Q^\pi(s_t, a_t)$. Policy improvement [32] proves that $Q^{\pi_{\text{new}}} \geq Q^{\pi_{\text{old}}}$ in objective maximization. π_{new} is defined by

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{KL}\left(\pi'(\cdot \mid s_t) \parallel \frac{\exp\left(Q^{\pi_{\text{old}}}(s_t, \cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)}\right) \quad (22)$$

where $Z^{\pi_{\text{old}}}(s_t)$ is the partition function for distribution normalization. It can be ignored because it does not contribute to the gradient of new policy. $Q^{\pi_{\text{old}}}$ guides the policy update to ensure an improved new policy. New policy is constrained to a parameterized family of distribution $\pi' \in \Pi$ like Gaussians to ensure the tractable and optimal new policy. Given the repeated application of policy evaluation and improvement, policy π eventually converges to the optimal policy π^*, $Q^{\pi^*} \geq Q^\pi$, $\pi \in \Pi$.

SAC is the combination of soft policy iteration and function approximation. In (19), temperature α is either a fixed value or an adaptive value. In function approximation, networks θ and ϕ are used to approximate the action value and policy value. The action value objective and its gradient are obtained by

$$\begin{cases} J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q(s_t, a_t; \theta) - \bar{Q}(s_t, a_t)\right)^2\right] \\ \bar{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}\left[V\left(s_{t+1}; \bar{\theta}\right)\right] \\ \nabla_\theta J(\theta) = \nabla_\theta Q(s_t, a_t; \theta) \\ \qquad \cdot \left(Q(s_t, a_t; \theta) - r(s_t, a_t) + \gamma V\left(s_{t+1}; \bar{\theta}\right) \right. \\ \qquad \qquad \left. -\alpha \log \pi_\phi(a_{t+1} \mid s_{t+1})\right) \end{cases} \quad (23)$$

where $\bar{Q}$ is the target action value. A state value $V(s_{t+1}; \bar{\theta})$ is approximated by a target action value network $\bar{\theta}$. γ is a discount factor. Policy objective and its gradient are obtained by

$$\begin{cases} J(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{KL}\left(\pi_\phi(\cdot \mid s_t) \parallel \frac{\exp(Q(s_t,\cdot;\theta))}{Z_\theta(s_t)}\right)\right] \\ \qquad = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\phi}\left[\alpha \log \pi_\phi(a_t \mid s_t) - Q(s_t, a_t; \theta)\right]\right] \\ \nabla_\phi J(\phi) = \nabla_\phi \alpha \log \pi_\phi(a_t \mid s_t) \\ \qquad + \nabla_\phi f_\phi(\epsilon_t; s_t) \\ \qquad \cdot \left(\nabla_{a_t} \alpha \log \pi_\phi(a_t \mid s_t) - \nabla_{a_t} Q(s_t, a_t)\right) \\ a_t = f_\phi(\epsilon_t; s_t) \end{cases} \quad (24)$$

where $f_\phi(\epsilon_t; s_t)$ is the network transformation, in which ϵ_t is an input noise vector sampled from fixed distribution like spherical Gaussian. The temperature objective is defined by

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t}\left[-\alpha \log \pi_t(a_t \mid s_t) - \alpha \bar{\mathcal{H}}\right] \quad (25)$$

where $\bar{\mathcal{H}}$ is the target entropy. Temperature objective gradient is obtained by approximating dual gradient descent [43]. Eventually, the networks and temperature are updated by

$$\begin{cases} \theta \leftarrow \theta - \gamma_\theta \nabla_\theta J(\theta) \\ \phi \leftarrow \phi - \gamma_\phi \nabla_\phi J(\phi) \\ \alpha \leftarrow \alpha - \gamma_\alpha \nabla_\alpha J(\alpha) \\ \bar{\theta} \leftarrow \tau\theta + (1 - \tau)\bar{\theta} \end{cases} \quad (26)$$

where the discount factor $\tau \in (0, 1)$.

SAC is used in tasks with continuous action space. However, the action space in this article is discrete. Hence, SAC should be modified to suit our task. Some modifications [33] should be made. They are summarized as follows.

1) The Q function should be moved from $Q : S \times A \to \mathbb{R}$ to

$$Q : S \times A \to \mathbb{R}^{|A|} \quad (27)$$

where Q values of all possible actions should be outputted, instead of a Q value of the action taken by the robot.

2) The outputted policy should be the action distribution

$$\pi : S \to [0, 1]^{|A|} \quad (28)$$

instead of mean and covariance of action distribution of SAC $\pi : S \to \mathbb{R}^{2|A|}$.

3) In (25), its expectation $\mathbb{E}_{a_t\sim\pi_t}[\cdot]$ is obtained by the Monte Carlo estimation, which involves taking an expectation over action distribution [33]. In discrete action space, expectation should be calculated directly, instead of Monte Carlo estimation. Hence, the temperature objective changes into

$$J(\alpha) = \pi(s_t)^T\left[-\alpha \log \pi_t(s_t) - \alpha \bar{\mathcal{H}}\right]. \quad (29)$$

Similarly, the policy objective changes into

$$J(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\pi(s_t)^T\left[\alpha \log \pi_\phi(s_t) - Q(s_t; \theta)\right]\right]. \quad (30)$$

B. LSTM-DSAC and AL-DSAC

LSTM-DSAC: When designing the architecture of LSTM-DSAC, it is worthwhile to consider how LSTM connects with networks. Two options include: 1) one LSTM means that actor and critic networks share one LSTM and this option is used in [14] and 2) two LSTMs means that actor network and critic network have different LSTM. One LSTM encoder indicates that the actor–critic architecture does not contribute to the convergence of LSTM. LSTM updates its networks according to its own gradient

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{k} \frac{\partial E_i}{\partial W} \quad (31)$$

where k denotes the batch length and W denotes the network weights in LSTM. E denotes the prediction error, which is defined by

$$E_i = h(i) - h(i-1) \quad (32)$$

where $h(\cdot)$ is the prediction vector (hidden state). In one LSTM case, LSTM converges according to the gradient of itself.

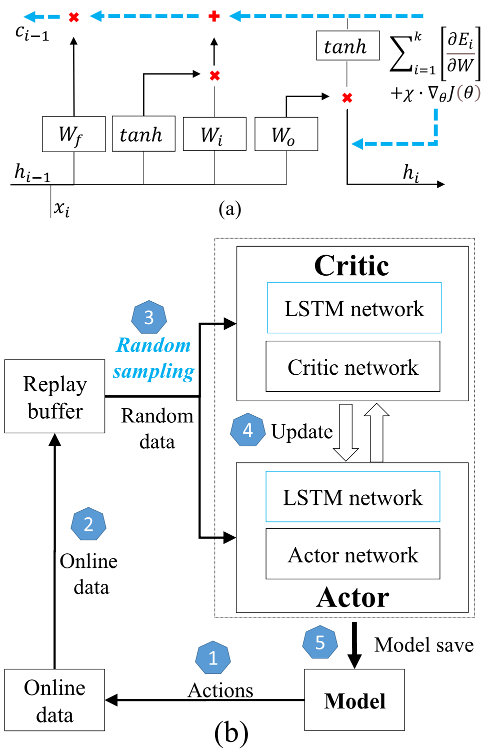

Fig. 4. Architecture of LSTM-DSAC. (a) LSTM backpropagation. (b) LSTM-DSAC architecture.

Moreover, there is no constraint here to regulate the gradient of LSTM. Hence, the gradient of LSTM may be small or unstable. The consequence is that the LSTM encoder may converge slowly and unstable. There may be large differences in the convergence among LSTM encoder, actor network, and critic network; therefore, overall convergence may be poor.

In two LSTM cases, LSTM combines critic network or actor network to form two integrated networks [Fig. 4(b)]. Integrated networks form new actor and critic, which are different from the original actor network and critic network. These new actor and critic (with LSTM) are in an actor–critic relationship, which contributes to the convergence of LSTM according to the chain rule in the backpropagation process [Fig. 4(a)]. This means that the gradient received by LSTM encoders changes from (33) to

$$\begin{cases} \dfrac{\partial E_{\text{lstm\_critic}}}{\partial W} = \sum_{i=1}^{k}\left[\dfrac{\partial E_{i\_\text{critic}}}{\partial W}\right] + \chi \cdot \nabla_\theta J(\theta) \\ \dfrac{\partial E_{\text{lstm\_actor}}}{\partial W} = \sum_{i=1}^{k}\left[\dfrac{\partial E_{i_{\text{actor}}}}{\partial W}\right] + \chi \cdot \nabla_\phi J(\phi) \end{cases} \tag{33}$$

where χ is a discount factor, which decides the portion of gradient contributed by actor network or critic network. θ and ϕ denote the critic network and actor network in function approximation. The gradient of actor or critic networks works as the constraint to accelerate and stabilize the convergence of LSTM. Two LSTMs converge in different pace, but two encoders fit their connected actor network or critic network. This contributes to the overall convergence of all networks. Hence, two LSTMs are used in the design of LSTM-DSAC. Finally, LSTM-DSAC architecture is designed in Fig. 4(b).

AL-DSAC: [14] shows that the first encoded feature has a large impact on LSTM gates, while the rear encoded feature has less impact. This means that the LSTM learns more front encoded features and forget more rear features when encoding a state with pairwise robot–obstacle features. In LSTM-DSAC, the pairwise robot–obstacle features are encoded in minimum-distance order (7). This indicates that the obstacle, which is close to the robot, has larger importance [14]. It is not always true because the obstacle importance also depends on speed and moving direction of obstacle according to human intuitions, instead of the distance to robot only.

AW [16] addresses this problem well by introducing attention score to evaluate the obstacle importance, instead of the distance. Pairwise obstacle features are ranked by attention-based importance (attention score). Then, ranked pairwise obstacle features are encoded by LSTM.

In existing work [16] and LSTM-DSAC, LSTM connects with actor or critic network to form new actor or critic. Intuitively, AW can connect with actor or critic network to form the AW-based DSAC (AW-DSAC). Then, AW is updated indirectly by the loss gradient. However, this makes the attention network receive small gradient in backpropagation, and slow convergence follows.

To solve this problem, we optimize the indirect update of AW to a direct way by separating AW from actor or critic network [Fig. 5(a)]. Separate AW, actor, and critic form a double actor–critic architecture, which means: 1) direct actor–critic relationship of actor and critic and 2) indirect actor–critic relationship of attention network and actor/critic [Fig. 5(b)]. As a result, the actor and critic contribute to the convergence of each other. The loss gradient of actor contributes to the convergence of AW. AW also contributes to the convergence of actor/critic. To be specific, the importance of each pairwise robot–obstacle feature is computed by AW, and therefore, pairwise robot–obstacle features are ranked or reordered by their importance. Then, LSTM encodes reordered pairwise robot–obstacle feature. Hence, LSTM remembers more robot–obstacle features with large importance and forgets some features with small importance. This improves the data quality and indirectly contributes to overall convergence. Finally, the AL-DSAC architecture is designed in Fig. 5(c).

C. ALN-DSAC

Work [35] indicated that episodic data with online feature contribute more to the convergence than offline data stored in the replay buffer. Note that the online feature denotes the priority or weight of environment state in different time step. This weight is represented by the accumulative reward received in each state. AL-DSAC considers merely learning from offline data in replay buffer. It does not make the best of episodic data with online feature (online weighted data); therefore, there is space for further improvement in convergence.

Inspired by [35] and [44], which learns online data and offline data simultaneously to improve the convergence of RL, we propose ALN-DSAC to learn online weighted data and offline data simultaneously. This is achieved by the novel PER, which prepares the online combined data and offline prioritized data for training.

To prepare online combined data [Fig. 6(a)], online episodic data cannot be used for batch learning directly because its data distribution is different from that of batch data. Data with different data distribution will cause overfitting in training. To solve this problem, we first prepare prioritized data-1 (size n). It then combines with online weighted episodic data

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

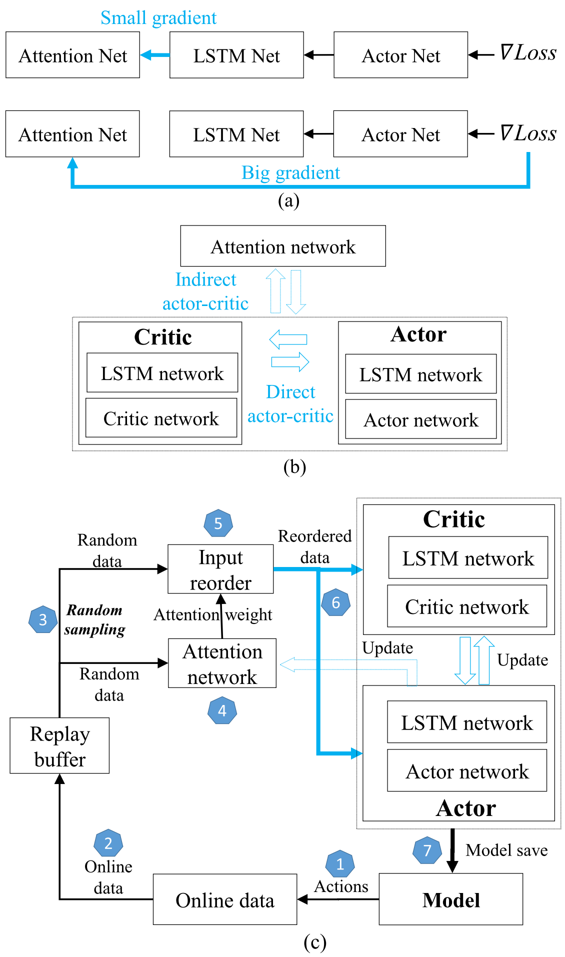

Fig. 5. Mechanism of AL-DSAC. (a) Mechanism for fast convergence of attention network. (b) Double actor–critic mechanism. (c) Architecture of AL-DSAC.

(size m) to form the combined data. Prioritized data-1 is selected from sampled batch data by the weight or priority, which is the accumulative reward (Monte Carlo return) defined by

$$P_t = \sum_{k}^{\infty} \gamma^{k-t} r_k. \tag{34}$$

To prepare offline prioritized data [Fig. 6(b)], K-batch experiences are sampled from replay buffer. Offline prioritized data are then selected from concatenated K experiences by the weight defined by (34). However, sampled K-batch experiences share the similarity to each other. This makes offline prioritized data lack diversity in training, and slow convergence speed and suboptimal converged result follow. To solve the problem of large similarity in sampled K experiences, we first apply cosine difference [44] to describe the similarity of sampled K experiences. Then, if the similarity of offline prioritized data is smaller than the threshold $\xi < \xi_{\text{threshold}}$, offline prioritized data are selected for training. Otherwise, a new batch data is randomly resampled from replay buffer to replace offline prioritized data for training. Similarity ξ is defined by

$$\xi = 1 - \cos^{-1}\left(\frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|}\right) \tag{35}$$

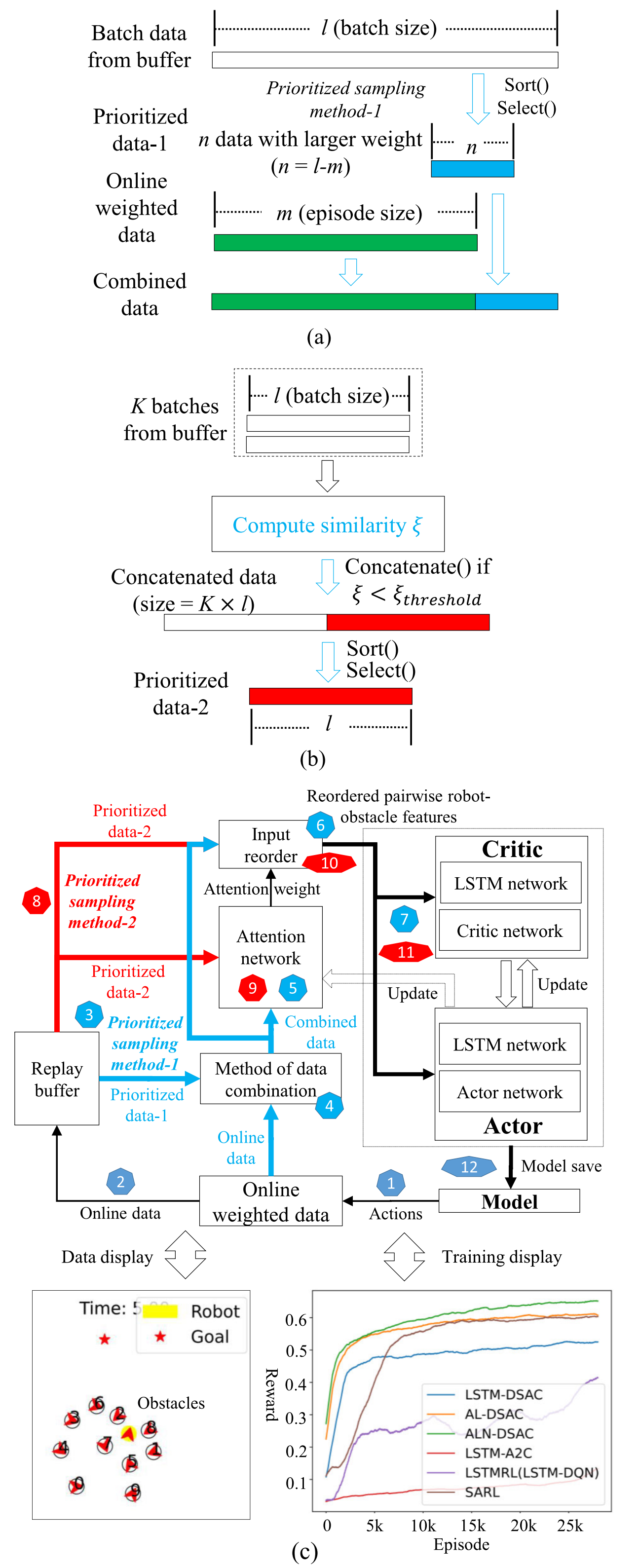

Fig. 6. Architecture of ALN-DSAC. (a) Prioritized sampling method-1 and data combination method to prepare the combined data. (b) Prioritized sampling method-2 to prepare the prioritized data-2. (c) Architecture of ALN-DSAC.

where v_1 and v_2 denote priority vectors of two sampled experiences and $\cos^{-1}((v_1 \cdot v_2)/(\|v_1\|\|v_2\|))$ denotes the cosine difference.

The ALN-DSAC architecture is shown in Fig. 6(c). Actions are selected to generate online weighted data, which is saved

Algorithm 1 ALN-DSAC

1.Initialize the replay buffer $\mathcal{D}$
2.Initialize attention network θ_{att}, critic networks θ_{c1} and θ_{c2}, and policy network θ_p
3.Initialize target critic networks $\bar{\theta}_{c1}$ and $\bar{\theta}_{c2}$: $\bar{\theta}_{c1} \leftarrow \theta_{c1}$, $\bar{\theta}_{c2} \leftarrow \theta_{c2}$
4. **For** episode $i < N$ **do**
5. **For** $t \neq T_{terminal}$ in episode i **do**
6. Execute action: $\langle s_t, a_t, r_t, S_{t+1}\rangle \sim \pi(a_t|s_t;\theta_p)$
7. **Off-Train** If length (D) < batch size l
8. Compute priorities of data in entire episode:

$$P_t = \sum\nolimits_k^\infty \gamma^{k-t} r_k$$

9. Store M episodes data $\mathcal{E}_{M\_delay}$
10. Update replay buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{E}_{M\_delay}$ if $i\%M = 0$
11. **On-Off-Train**
12. $i = i + 1$
13.Save models: θ_{att}, θ_{c1}, θ_{c2} and θ_p

Algorithm 2 Off-Train

1.// Prepare prioritized data-2
2.Sample K batches $\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_K\}$ randomly from replay buffer $\mathcal{D}$
3.Compute data similarity ξ
4. **If** $\xi < \xi_{threshold}$
5. Concatenate K-batch data
6. Sort data by their priorities
7. Select largest-priority data $\mathcal{E}_{prioritized-data-2}$ with the length l
8. **else**
9. Resample one batch data $\mathcal{E}_{random}\langle s, a, r, S', P\rangle$ from $\mathcal{D}$
10. **Forward-Backpropagation**

Algorithm 3 On-Off-Train

1.// Prepare prioritized data-1 and combined data
2.Sample one batch data $\mathcal{E}_{random}$ from replay buffe $\mathcal{D}$
3.Sort the data $\mathcal{E}_{random}$ by its priority
4.Select the largest-priority data $\mathcal{E}_{prioritized-data-1}$ with length $n, n = l - m$
5.Concatenate the online and offline data:

$$\mathcal{E}_{combined}\langle s, a, r, S', P\rangle = \mathcal{E}_i + \mathcal{E}_{prioritized-data-1}$$

6. **Forward-Backpropagation**

in replay buffer (steps 1 and 2). For learning from online data, at the end of each episode, prioritized sampling method-1 is used to generate prioritized data-1 (step 3), which combines with current online weighted data to form combined data (step 4). In training, combined data are fed to an attention network to generate AW, which is used to reorder the pairwise robot–obstacle features in each state of combined data (steps 5 and 6). Actor and critic (combination of LSTM and critic/actor network) then learn from reordered data (step 7). For learning from batch data, at each state, prioritized sampling method-2 is used to generate prioritized data-2 for training (step 8). Training with batch data (steps 9–11) is the same as training with combined data. Previous steps repeat until the convergence of networks. Eventually, models (attention model, actor model, and critic model) are saved for evaluation (step 12).

Algorithm (Pseudocode): ALN-DSAC is described in Algorithm 1, while Algorithms 2–4 are its subalgorithms. In Algorithm 1, replay buffer, networks are first initialized. These networks include the attention network θ_{att}, critic networks (θ_{c1} and θ_{c2}), target critic networks ($\bar{\theta}_{c1}$ and $\bar{\theta}_{c2}$), and policy network θ_p. Here, double critic architecture is used to reduce the overestimation of Q value. Then, the experience of each state $\langle s_t, a_t, r_t, S_{t+1}\rangle$ is obtained by

$$\langle s_t, a_t, r_t, S_{t+1}\rangle \sim \pi\left(a_t|s_t;\theta_p\right). \tag{36}$$

If length ($\mathcal{D}$) < batch size l, networks learn from the prioritized data-2, which is obtained by subalgorithm *Off-Train*. Once the robot reaches the terminal state (finding the goal, collision, and timeout), priorities of episodic experience are obtained by $P_t = \sum_k^\infty \gamma^{k-t} r_k$.

However, saving only one episodic experience to replay buffer at the end of every episode will cause poor data diversity in early stage training. This means that experiences sampled via subalgorithms *Off-Train* in each step for batch learning are the same or almost the same. Networks trained with these similar experiences converge slowly. Delayed infusion of recent experiences [44] is adopted to solve this problem. This means that M episodic experiences (format $\langle s_t, a_t, r_t, S_{t+1}, P_t\rangle$) are saved in every M episodes, instead of current one episodic experience $\mathcal{E}_i$. This is achieved by

$$\mathcal{E}_{M\_\text{delay}} = \{\mathcal{E}_i, \mathcal{E}_{i-1}, \ldots, E_{i-M+1}\}. \tag{37}$$

Current episodic experience $\mathcal{E}_i$ then combines the prioritized data-1 to form the combined data, which is fed to subalgorithm *On-Off-Train*. Previous steps repeat until the convergence of networks. Then, networks are saved for evaluations.

Algorithm 2 (network training based on offline data) is executed in every step of an episode. K-batch experiences $\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_K\}$ are randomly sampled from replay buffer. Experience is in the format with priorities $< s_{l'}, a_{l'}, r_{l'}, S_{l'}, P_{l'} >, l' \in l$ and its similarity ξ is computed by $\xi = 1 - \cos^{-1}((v_1 \cdot v_2)/(\|v_1\|\|v_2\|))$. If $\xi < \xi_{\text{threshold}}$, these K experiences are concatenated and sorted by their priorities to prepare the prioritized data-2 $\mathcal{E}_{\text{prioritized-data-2}}$. Otherwise, new batch experience $\mathcal{E}_{\text{random}} < s, a, r, S', P >$ is randomly sampled from replay buffer. Finally, networks are updated via feedforward and backpropagation processes *Forward-Backpropagation*.

Algorithm 3 (network training based on combined data) is executed at the end of an episode. One batch experience $\mathcal{E}_{\text{random}}$ is first sampled from replay buffer randomly. $\mathcal{E}_{\text{random}}$ is sorted or reordered by its priority to prepare the prioritized data-1 $\mathcal{E}_{\text{prioritized-data-1}}$. Current online experience $\mathcal{E}_i$ combines $\mathcal{E}_{\text{prioritized-data-1}}$ to form a combined experience $\mathcal{E}_{\text{combined}} < s, a, r, S', P >$. Finally, networks are updated using combined experience in the feedforward and backpropagation processes *Forward-Backpropagation*.

Algorithm 4 denotes *network feedforward and backpropagation*. The input $\mathcal{E}$ can be one of $\mathcal{E}_{\text{random}}$, $\mathcal{E}_{\text{prioritized-data-2}}$, and $\mathcal{E}_{\text{combined}}$. These experiences have the same length l. The feedforward process consists of computing four loss values: 1) critic loss; 2) policy loss; 3) attention loss, and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Algorithm 4 Forward-Backpropagation

1. // Calculate critic loss

2.Calculate *value* of next attention weight:
$AW_{next} \leftarrow f_{\theta_{att}}(S')$

3.Rank next state according to AW_{next}:

$$S'_{ranked} \leftarrow rank_{\max\_weight}(S')$$

4.Calculate next probability distribution and its log value:

$$p_{next}, \log p_{next} \leftarrow f_{\theta_p}(S'_{ranked})$$

5.Calculate next target Q values:

$$\bar{q}_{next\_1}, \bar{q}_{next\_2} \leftarrow f_{\bar{\theta}_{c1/c2}}(S'_{ranked})$$

6.Calculate expectation of next state value:

$$\mathbb{E}_{S'\sim p}[V(S')] = \sum [p_{next} \cdot \min(\bar{q}_{next\_1}, \bar{q}_{next\_2}) - \alpha \cdot \log p_{next}]$$

7.Calculate discounted next target Q value:

$$\bar{Q}_{next} = r(s, a) + \gamma \mathbb{E}_{S'\sim p}[V(S')]$$

8.Calculate *value* of current attention weight:
$AW \leftarrow f_{\theta_{att}}(s)$

9.Rank current state according to AW:

$$s_{ranked} \leftarrow rank_{\max\_weight}(s)$$

10.Calculate current Q values: $q_1, q_2 \leftarrow f_{\theta_{c1/c2}}(s_{ranked})$

11.Calculate critic loss:
$\mathcal{L}_{critic} = MSE(q_1, \bar{Q}_{next}) + MSE(q_2, \bar{Q}_{next})$

12.// Calculate policy loss

13.Calculate current probability distribution and its log value:

$$p, \log p \leftarrow f_{\theta_p}(s_{ranked})$$

14.Calculate current critic values:

$$q_{1\_policy}, q_{2\_policy} \leftarrow f_{\theta_{c1/c2}}(s_{ranked})$$

15.Calculate current entropy:

$$\mathcal{H}(\pi(\cdot\,|\,s)) = -log\pi(\cdot\,|\,s) = -\sum p \cdot \log p$$

16.Calculate expectation of current Q value:

$$\mathbb{E}_{s\sim p}[Q(s)] = \sum \min[(q_{1_{policy}}, q_{2_{policy}}) \cdot p]$$

17.Calculate policy loss:

$$\mathcal{L}_{policy} = -mean[\mathbb{E}_{s\sim p}[Q(s)] + \alpha \cdot \mathcal{H}(\pi(\cdot\,|\,s))]$$

18.// Calculate attention loss

19.Calculate policy loss:

$$\mathcal{L}_{attention} = L_{policy} = -mean[\mathbb{E}_{s\sim p}[Q(s)] + \alpha \cdot \mathcal{H}(\pi(\cdot\,|\,s))]$$

20.// Calculate temperature loss

21. Calculate temperature loss: $\mathcal{L}_\alpha = -\min[log\alpha \cdot (\bar{\mathcal{H}} - \mathcal{H})]$

22.// Update networks and temperature (backpropagation)

23. Update the critic network:
$\theta_{ci} \leftarrow \theta_{ci} - \gamma \nabla_{\theta_{ci}} \mathcal{L}_{critic}, i \in 1, 2$

Algorithm *(Continue.)* Forward-Backpropagation

24. Update the policy network:
$\theta_p \leftarrow \theta_p - \gamma \nabla_{\theta_p} \mathcal{L}_{policy}$

25. Update the attention network:
$\theta_{att} \leftarrow \theta_{att} - \gamma \nabla_{\theta_{att}} \mathcal{L}_{attention}$

26. Update the temperature:
$\alpha \leftarrow \alpha - \gamma \nabla_\alpha \mathcal{L}_\alpha, \alpha \leftarrow e^\alpha$

4) temperature loss. Backpropagation consists of four updates: 1) critic networks; 2) policy network; 3) attention network; and 4) temperature parameter.

Compute Critic Loss: The next target Q value and the current Q value are required to compute the critic loss. To compute the next target Q value, the next state S' is first fed to the attention network θ_{att} to generate its AW by

$$AW_{\text{next}} \leftarrow f_{\theta_{\text{att}}}(S'). \tag{38}$$

Pairwise robot–obstacle features in S' are reordered by its AW to obtain the ranked next state by

$$S'_{\text{ranked}} \leftarrow \text{rank}_{\text{max\_weight}}(S'). \tag{39}$$

S'_{ranked} is fed to the policy network to obtain the next probability distribution and its log value by

$$p_{\text{next}}, \log p_{\text{next}} \leftarrow f_{\theta_p}(S'_{\text{ranked}}). \tag{40}$$

Note that here, the policy network is combined with LSTM to form a new actor. S'_{ranked} is also fed to double target critic networks $\bar{\theta}_{c1/c2}$ to obtain the next target Q values by

$$\bar{q}_{\text{next\_1}}, \bar{q}_{\text{next\_2}} \leftarrow f_{\bar{\theta}_{c1/c2}}(S'_{\text{ranked}}). \tag{41}$$

However, the next target Q values cannot be used to compute the critic loss directly. They should combine with entropy to form the discounted next target Q value. This is achieved by computing the expectation of next state value via

$$\mathbb{E}_{S'\sim p}[V(S')] = \sum [p_{\text{next}} \cdot \min(\bar{q}_{\text{next\_1}}, \bar{q}_{\text{next\_2}}) - \alpha \cdot \log p_{\text{next}}] \tag{42}$$

where α is the temperature parameter. Note that the Q value here is in format $Q : S \times A \rightarrow \mathbb{R}^{|A|}$; hence, the next state value is the sum of Q values from every action, instead of Q value of an action taken by the robot. Thus, the next target Q value is obtained by

$$\bar{Q}_{\text{next}} = r(s, a) + \gamma \mathbb{E}_{S'\sim p}[V(S')]. \tag{43}$$

To calculate the current Q value, current state s is first fed to attention network θ_{att} to generate its AW by

$$\text{AW} \leftarrow f_{\theta_{\text{att}}}(s). \tag{44}$$

Pairwise robot–obstacle features in s are then reordered by obtained AW via

$$s_{\text{ranked}} \leftarrow \text{rank}_{\text{max\_weight}}(s). \tag{45}$$

Thus, the current Q value is computed by feeding reordered pairwise robot–obstacle features s_{ranked} to double critic networks $\theta_{c1/c2}$ to obtain current critic values via

$$q_1, q_2 \leftarrow f_{\theta_{c1/c2}}(s_{\text{ranked}}). \tag{46}$$

Note that here, critic networks are combined with LSTM to form a new critic. Finally, the critic loss is obtained by summing the mean square error (mse) of q_1/q_2 and $\bar{Q}_{\text{next}}$ via

$$\mathcal{L}_{\text{critic}} = \text{mse}(q_1, \bar{Q}_{\text{next}}) + \text{mse}(q_2, \bar{Q}_{\text{next}}). \tag{47}$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Compute Policy Loss: The current ranked state s_{ranked} is first fed to the policy network θ_p to obtain the current probability distribution and its log value by

$$p, \log p \leftarrow f_{\theta_p}(s_{\text{ranked}}). \tag{48}$$

Simultaneously, s_{ranked} is also fed to double critic networks $\theta_{c1/c2}$ to obtain current critic values by

$$q_{1\_\text{policy}}, q_{2\_\text{policy}} \leftarrow f_{\theta_{c1/c2}}(s_{\text{ranked}}). \tag{49}$$

The current entropy $\mathcal{H}(\pi(\cdot \mid s))$ is computed by

$$\mathcal{H}(\pi(\cdot \mid s)) = -\log\pi(\cdot \mid s) = -\sum p \cdot \log p. \tag{50}$$

The expectation of current Q value $\mathbb{E}_{s\sim p}[Q(s)]$ is computed by

$$\mathbb{E}_{s\sim p}[Q(s)] = \sum \min\left[\left(q_{1_{\text{policy}}}, q_{2_{\text{policy}}}\right) \cdot p\right]. \tag{51}$$

Finally, the policy loss is computed by

$$\mathcal{L}_{\text{policy}} = -\text{mean}\left[\mathbb{E}_{s\sim p}[Q(s)] + \alpha \cdot \mathcal{H}(\pi(\cdot \mid s))\right]. \tag{52}$$

Compute Attention Loss: The attention network and policy network share the same loss by

$$\mathcal{L}_{\text{attention}} = \mathcal{L}_{\text{policy}} = -\text{mean}\left[\mathbb{E}_{s\sim p}[Q(s)] + \alpha \cdot \mathcal{H}(\pi(\cdot \mid s))\right]. \tag{53}$$

Compute Temperature Loss: The temperature loss is computed by minimizing (30) via

$$\mathcal{L}_{\alpha} = -\min\left[\log\alpha \cdot (\bar{\mathcal{H}} - \mathcal{H})\right]. \tag{54}$$

Update of Networks: Critic networks, policy network, attention network, and temperature parameter are updated by gradient ascent via

$$\theta_{ci} \leftarrow \theta_{ci} - \gamma \nabla_{\theta_{ci}} \mathcal{L}_{\text{critic}}, \quad i \in 1, 2 \tag{55}$$

$$\theta_p \leftarrow \theta_p - \gamma \nabla_{\theta_p} \mathcal{L}_{\text{policy}} \tag{56}$$

$$\theta_{\text{att}} \leftarrow \theta_{\text{att}} - \gamma \nabla_{\theta_{\text{att}}} \mathcal{L}_{\text{attention}} \tag{57}$$

$$\alpha \leftarrow \alpha - \gamma \nabla_{\alpha} \mathcal{L}_{\alpha}, \quad \alpha \leftarrow e^{\alpha} \tag{58}$$

where γ is a discount factor. Note that the update of temperature parameter has extra process after the gradient ascent, that is, $\alpha \leftarrow e^{\alpha}$, which contributes to the convergence of temperature.

IV. Experiments

A. Model Frameworks for the Experiment

Model frameworks of LSTM-DSAC, AL-DSAC, and ALN-DSAC are designed in Fig. 7. In the LSTM-DSAC framework, double critic networks are used to reduce the overestimation of Q value. Each critic network has three linear layers. Double critic networks are also used in the framework of AL-DSAC and ALN-DSAC. The difference of these two frameworks is the attention network, which consists of one softmax layer and three MLPs (f_e, f_h, and f_a). The output is the AW value, which is used to reorder the pairwise robot–obstacle features in each state. The attention network takes a policy loss gradient to update its weight in backpropagation. Configurations (parameters) of frameworks are shown in Table I.

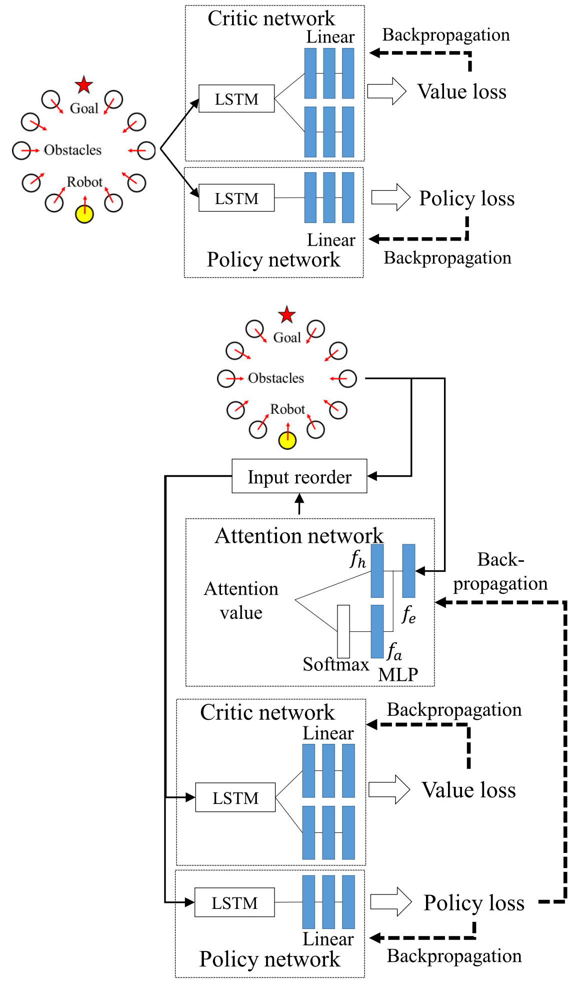

Fig. 7. Network frameworks. The first framework is for LSTM-DSAC, while the second is for AL-DSAC and ALN-DSAC.

B. Model Training

We first implement LSTM-DQN (LSTMRL), LSTM-DSAC, AW-DSAC, AL-DSAC, and ALN-DSAC as the ablation experiments to check the contributions of DSAC, attention-based LSTM encoding, and novel data replay method in convergence. Then, experiments are extended to cases with two and ten obstacles. Finally, LSTM-DSAC, AL-DSAC, and ALN-DSAC are compared with the state of the arts, which include CADRL, LSTM-DQN (LSTMRL), SARL, and A2C-LSTM.

Fig. 8(a) shows that: 1) DSAC converges faster than DQN by comparing the training of LSTM-DQN and LSTM-DSAC; 2) separate two LSTMs converge faster than one shared LSTM in LSTM-DSAC; and 3) LSTM encoding contributes more to convergence than that of AW pooling by comparing LSTM-DSAC and AW-DSAC. Fig. 8(b) shows that: 1) attention-based LSTM encoding contributes more to convergence than that of LSTM encoding by comparing the training of AL-DSAC and LSTM-DSAC and 2) novel data replay method contributes to the convergence of early stage training by comparing the training of AL-DSAC and ALN-DSAC. Fig. 8(c) presents the ALN-DSAC with different configurations in the novel data replay. Comparisons show that our novel data replay method with five delayed experiences is the most efficient method to speed up the convergence.

Fig. 8(d) shows the convergence of LSTM-DSAC, AL-DSAC, and ALN-DSAC in a two-obstacle case. The con-

TABLE I
PARAMETERS OF ALN-DSAC

Parameters/Hyper-parameters	Values of Parameters/Hyper-parameters
Similarity threshold $\xi_{\text{threshold}}$	0
Number of delayed episodes	5/10 (5/10 obstacles)
Frequency of network update (Online data case)	Per episode
LSTM hidden size	50
Number of MLP layer	3
ReLU layer after MLP	Yes (First MLP layer)
MLP input/output size (interaction layer and embedded layer)	[150, 100]-[100, 50]
MLP input/output size (attention layers)	[100, 100]-[100, 1]
Reward	Source reward
Gamma	0.95
Tau	0.005
Learning rate	3e-4
Alpha	0.2
Frequency of network update (Offline data case)	Per step
Automatic entropy tuning	True
Batch size	128
Input layer size (SAC network)	6+50
Hidden layer size (SAC network)	128
Output size of policy network (SAC network)	81

vergences of these algorithms are almost the same, although AL-DSAC performs slightly better than the rest algorithms. Their differences are small (reward difference <0.1). Fig. 8(e) shows the convergence of these three algorithms in the five-obstacle case. AL-DSAC and ALN-DSAC perform almost the same, but better than LSTM-DSAC. Fig. 8(f) shows the convergence of these three algorithms in the ten-obstacle case. ALN-DSAC performs the best and, then, the AL-DSAC and LSTM-DSAC. Fig. 8(d)–(f) also shows that the number of delayed experiences matters. When obstacles are less (e.g., two and five obstacles), small, delayed experience number (e.g., 5) works well. Large, delayed experience number (e.g., 10) works well in case with more obstacles (e.g., ten obstacles).

Fig. 8(g)–(i) shows the comparisons of LSTM-DSAC, AL-DSAC, and ALN-DSAC against the state of the arts in convergence. In the five-obstacle case, these three algorithms outperform all state of the arts. In the ten-obstacle case, AL-DSAC and ALN-DSAC outperform all state of the arts. LSTM-DSAC converges faster than the state of the arts, but its converged result is poor than that of SARL.

C. Model Evaluations

We first present the converged reward of all algorithms in training. Then, all trained models are evaluated according to evaluation criteria, which consist of qualitative evaluation, quantitative evaluation, computational evaluation, and robustness evaluation.

1) Converged Reward: Converged rewards are listed in Table II. AL-DSAC and ALN-DSAC outperform other algorithms in the five-obstacle case. ALN-DSAC outperforms other algorithms in the ten-obstacle case. Note that LSTM-A2C is an on-policy RL algorithm, which is trained with 60k episodic experiences.

TABLE II
CONVERGED REWARD OF ALL ALGORITHMS IN CASES WITH FIVE AND TEN OBSTACLES

Algorithm	Converged reward
ORCA [5][38][37]	---
CADRL [36][20]	0.61* (Training with one obstacle)
LSTMRL [14][20]	0.49/0.33
SARL [16][20]	0.55/0.50
LSTM-A2C [11][14]	0.30/0.25* (60k episodes in 10-obstacle case)
Our LSTM-DSAC	0.59/0.42
Our AL-DSAC	**0.60**/0.50
Our ALN-DSAC	**0.60/0.54**

2) Qualitive Evaluation (Trajectory Quality): The motion planning process and the policy evolvement process are described before evaluation to clarify: 1) how motion planning task is accomplished and 2) how the model evolves with the increase of training data. Motion planning processes with five and ten obstacles are shown in Fig. 9. The robot is controlled by ALN-DSAC, while obstacles are controlled by ORCA. Policy evolvement processes with five and ten obstacles are shown in Fig. 10, in the supplementary material. Models of ALN-DSAC trained with a different number of experiences (1k–30k episodes) are executed. Trajectories are generated according to trained models. Policy evolvement is observed by analyzing the time to reach the goal. The policy evolves stably despite small fluctuations (e.g., 18k and 24k in cases with five and ten obstacles, respectively).

We analyze trajectories and conclude that trajectories consist of three types : bypass, wait-cross, and cross. The bypass strategy is the most efficient and safe strategy, while the wait-cross and cross strategies lack efficiency and safety. The bypass strategy means the robot fast bypasses all obstacles that move toward the center and their goals (Fig. 10). The wait-cross strategy means that the robot keeps waiting or slow until obstacles move away from the center. Then, the robot moves fast and right across the center to reach its goal. The cross strategy means that the robot moves toward the center and its goal with medium speed and short distance to obstacles. Learned motion planning strategies are shown in Table III. LSTM-DSAC, AL-DSAC, ALN-DSAC, and SARL learned the bypass strategy (the most efficient and safe strategy). LSTM-DSAC, LSTM-A2C, LSTMRL, and CADRL learned the wait-cross strategy. ORCA is the only algorithm, which uses the cross strategy.

3) Quantitative Evaluation: Quantitative evaluation of algorithms is measured by six criteria: success rate, time to goal, collision rate, timeout rate, mean distance to obstacles, and mean discounted reward. The success rate, collision rate, and timeout rate denote the rates of the case in which the robot reaches the goal, collides with obstacles, and does not reach its goal within time limit (25 s), respectively. The 500 tests are conducted in circle-crossing simulator to evaluate each algorithm in cases with five or ten obstacles. The detailed comparisons are shown in Table IV. LSTM-DSAC, AL-DSAC, and ALN-DSAC outperform near all state of the arts in cases with five and ten obstacles. LSTM-DSAC performs good in the five-obstacle case because of two reasons: 1) high efficiency of DSAC in convergence and 2) the competence

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

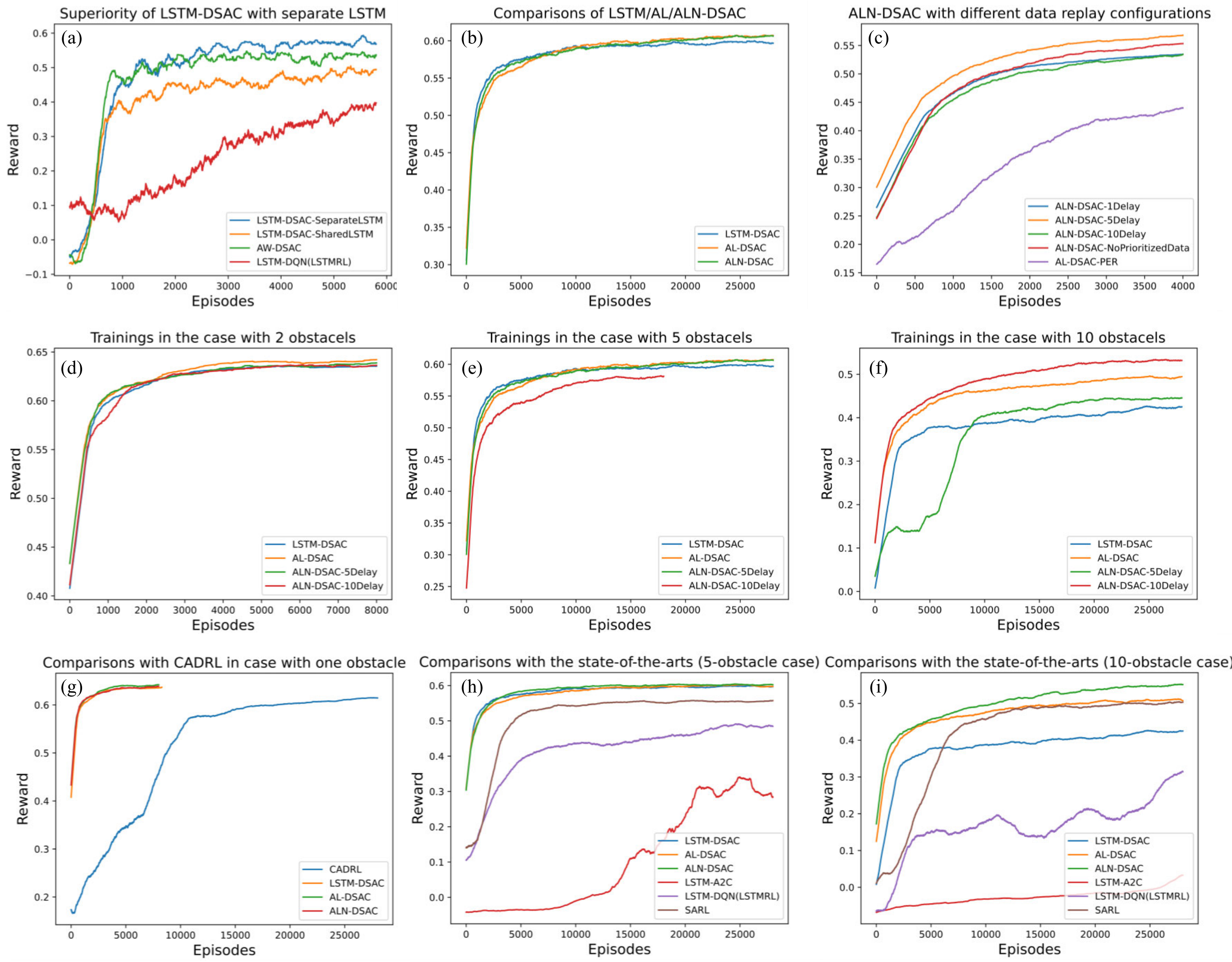

Fig. 8. Training results. In (c), ALN-DSAC-1/5/10Delay denotes the novel data replay with one/five/ten delayed experiences. ALN-DSAC-NoPrioritizedData denotes that the prioritized sampling method-1/2 is replaced by random sampling in the novel data replay. AL-DSAC-PER denotes that AL-DSAC uses classical PER. In (g), CADRL does not support multiobstacle training. Its networks are trained in one-obstacle case, and trained models can be applied to multiobstacle motion planning. Hence, the training of CADRL is presented in independent figure.

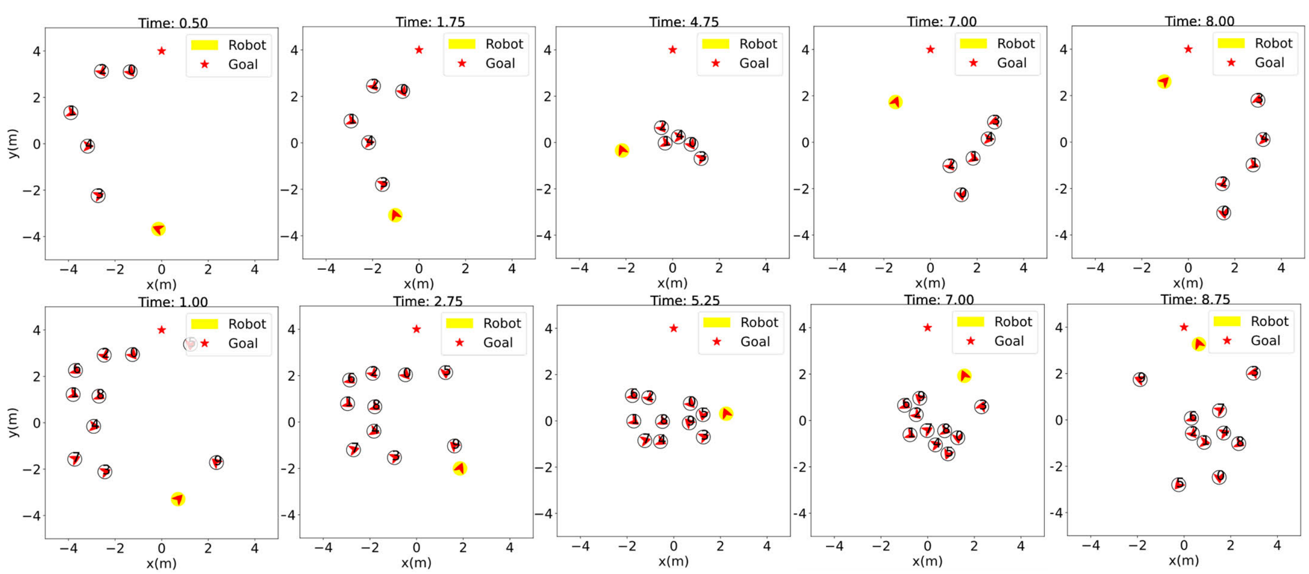

Fig. 9. Motion planning process (ALN-DSAC) in cases with five and ten obstacles.

of distance-based LSTM to represent the obstacle importance. However, distance-based LSTM partly represents the obstacle importance, and therefore, its weakness appears in the ten-obstacle case. AL-DSAC improves the distance-based

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE III
LEARNED MOTION PLANNING STRATEGIES

Algorithm	Learnt motion planning strategy
ORCA	Cross (medium speed)
CADRL	Wait-cross (slow and fast speed)
LSTMRL	Wait-cross (slow and fast speed)
SARL	Bypass (fast speed)
LSTM-A2C	Wait-cross (slow and fast speed)
LSTM-DSAC	**Bypass** (fast speed); Wait-cross (slow and fast speed)
AL-DSAC	**Bypass** (fast speed)
ALN-DSAC	**Bypass** (fast speed)

TABLE IV
FIVE HUNDRED TESTS OF ALL ALGORITHMS IN CASES WITH FIVE AND TEN OBSTACLES IN CIRCLE-CROSSING ENVIRONMENT

Algorithm	Success rate (10/5 obs.)	Time to goal (10/5 obs.) (s)	Collision rate (10/5 obs.)	Timeout rate (10/5 obs.)	Mean distance (10/5 obs.) (m)	Mean rewards (10/5 obs.)
ORCA	0.21/0.43	12.49 /10.86	0.79/0.564	0.00/0.006	0.08/0.08	---
CADRL	0.71/0.89	13.76/11.30	0.29/0.106	0.00/0.004	0.15/0.16	0.2610/0.4659
LSTMRL	0.90/0.988	15.56/11.70	0.02/0.002	0.08/0.01	0.11/0.14	0.3751/0.5304
SARL	0.99/0.992	11.99/10.96	0.006/0.008	0.004/0.00	0.17/0.18	0.4917/0.5649
LSTM-A2C	0.79/0.88	18.46/17.04	0.05/0.05	0.16/0.07	0.13/0.12	0.3138/0.3616
LSTM-DSAC	0.97/**0.998**	15.67/9.90	0.022/**0.002**	0.008/**0.00**	**0.15**/0.16	0.4315/0.6036
AL-DSAC	0.984/0.996	12.74/**9.79**	0.016/0.004	**0.00/0.00**	0.14/**0.15**	0.5073/0.6028
ALN-DSAC	0.99/**0.998**	**11.20/9.79**	0.01/**0.002**	**0.00/0.00**	0.16/**0.15**	**0.5571/0.6046**

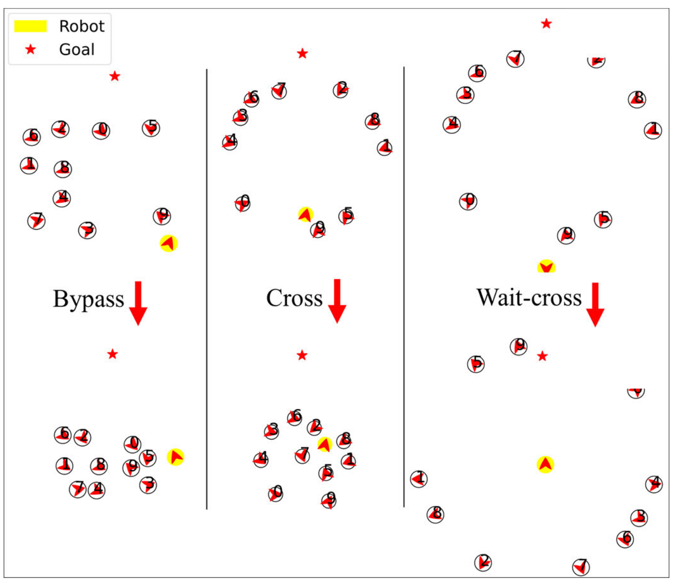

Fig. 10. Learned motion planning strategy.

encoding by attention-based encoding, which is robust and shows good convergence either in five or ten obstacle case. ALN-DSAC moves further in the improvement of convergence by improving the efficacy of data replay. This is achieved by learning from full-episodic experience and offline experience simultaneously.

4) Computational Evaluation: Time costs are recorded in Table V. The on-policy algorithm LSTM-A2C costs the least time in training compared to the rest algorithms. LSTM-DSAC costs the least time in training among off-policy algorithms.

5) Robustness Evaluation: Robustness here is defined by the value change in a new environment (square-crossing simulator). The value denotes six criteria used in the quantitative evaluation. The value comparisons are presented in Table VI where ALN-DSAC outperforms the rest algorithms in both ten-obstacle case and five-obstacle case. The value changes are presented in Table VII. LSTM-DSAC, AL-DSAC, and ALN-DSAC perform stable in square-crossing environment, while the state of the arts have large fluctuations in values. Robust evaluation shows good performances of our ALN-DSAC in two simulators with large differences. This indicates the potential of our algorithm on other general motion planning tasks.

Main hardware is Intel Core i7-9750H processor with 16-GB memory. The trainings and evaluations are based on the CrowdNav simulation system [5], [16]. The robot operation system (ROS) and physical implementations are shown on websites: https://www.youtube.com/watch?v=9znVReBmfwI and https://www.youtube.com/watch?v=bH5FbA14AqE.

V. CONCLUSION

Existing motion planning algorithms face challenges in data quality (representation learning), efficacy of data replay, and optimality of RL algorithm. Given these challenges, we first proposed LSTM-DSAC, which uses DSAC to learn features pooled by distance-based LSTM. Then, distance-based pooling is improved by priority-based pooling. This is AL-DSAC, which uses an attention network to compute the importance of each obstacle. LSTM then pools the pairwise robot–obstacle features in a priority-based order. Finally, we proposed ALN-DSAC where online episodic experience is utilized in training. Hence, the algorithm learns from online and offline experiences simultaneously. This further improves the convergence. We did extensive evaluations of ALN-DSAC against the state of the arts. Experiments show that ALN-DSAC outperforms the state of the arts in most evaluations. Future research will focus on the improvement of representation learning such as attention mechanism. It faces the challenge of overfitting if

TABLE V
TIME COST OF ALGORITHM TRAINING IN CASES WITH FIVE AND TEN OBSTACLES

Algorithm	Time cost (hour) for 5/10 obs.
ORCA	---
CADRL	22.2 (train with one obstacle)
LSTMRL	48.25/89.5
SARL	44.15/63.5
LSTM-A2C	**1.25/8.00** (60k episodes in 10-obstacle case)
LSTM-DSAC	**6.75/15.25**
AL-DSAC	7.95/16.00
ALN-DSAC	10.70/16.90

TABLE VI
FIVE HUNDRED TESTS IN SQUARE-CROSSING ENVIRONMENT IN CASES WITH FIVE AND TEN OBSTACLES

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.)	Mean rewards (10/5 obs.)
ORCA	0.44/0.74	10.64/9.12	0.55/0.256	0.01/0.004	0.08/0.08	---
CADRL	0.76/0.88	12.77/11.19	0.07/0.01	0.17/0.11	0.16/0.17	0.3439/0.4863
LSTMRL	0.74/0.91	13.62/10.54	0.06/0.03	0.20/0.06	0.11/0.12	0.3055/0.4903
SARL	0.92/0.93	12.38/11.06	0.01/0.01	0.07/0.06	0.16/0.17	0.4839/0.5479
LSTM-A2C	0.30/0.45	16.92/15.61	0.45/0.41	0.25/0.14	0.12/0.10	0.0510/0.1245
LSTM-DSAC	0.968/0.994	15.66/9.89	0.024/0.006	0.008/**0.00**	**0.15**/0.16	0.4306/0.6028
AL-DSAC	0.984/0.994	12.75/9.79	0.016/0.006	**0.00/0.00**	0.14/**0.15**	0.5069/0.6006
ALN-DSAC	**0.99/0.998**	11.19/9.80	**0.01/0.002**	**0.00/0.00**	0.16/**0.15**	**0.5556/0.6038**

TABLE VII
VALUE CHANGES FROM CIRCLE-CROSSING ENVIRONMENT TO SQUARE-CROSSING ENVIRONMENT

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.)	Mean rewards (10/5 obs.)
ORCA	0.23/0.31	-1.85/-1.56	-0.24/-0.308	0.01/-0.002	0/0	---
CADRL	0.05/-0.01	-0.99/-2.11	0.18/-0.096	0.17/0.106	0.01/0.01	0.0829/0.0177
LSTMRL	-0.16/-0.078	-1.94/-1.16	0.04/0.008	0.12/0.05	0/-0.02	-0.0696/-0.0401
SARL	-0.07/-0.062	0.39/0.1	0.004/0.002	0.066/0.06	-0.01/-0.01	-0.0078/-0.017
LSTM-A2C	-0.49/-0.43	-1.54/-1.43	0.4/0.36	0.09/0.07	-0.01/-0.02	-0.2628/-0.2371
LSTM-DSAC	-0.002/-0.004	**-0.01**/-0.01	0.002/0.004	**0/0**	**0/0**	-0.0009/**-0.0008**
AL-DSAC	**0**/-0.002	**0.01/0**	**0**/0.002	**0/0**	**0/0**	**-0.0004**/-0.0022
ALN-DSAC	**0/0**	**-0.01**/0.01	**0/0**	**0/0**	**0/0**	-0.0015/**-0.0008**

the network is too deep and complex. This is expected to be solved by integrating the skip connection and other robust pooling methods such as LSTM pooling.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[2] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 723–730.

[3] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, Oct. 1990.

[4] J. Funke et al., "Up to the limits: Autonomous Audi TTS," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2012, pp. 541–547.

[5] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2008, pp. 100–107.

[6] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 2061–2067.

[7] R. T. Farouki and Z. Šír, "Rational pythagorean-hodograph space curves," *Comput. Aided Geometric Des.*, vol. 28, no. 2, pp. 75–88, Feb. 2011.

[8] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[9] Z. Bai, B. Cai, W. ShangGuan, and L. Chai, "Deep learning based motion planning for autonomous vehicle using spatiotemporal LSTM network," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2018, pp. 1610–1614.

[10] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 1995–2003.

[11] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, vol. 4, 2016, pp. 2850–2869, 2016.

[12] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 6252–6259.

[13] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

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

[15] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 961–971.

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

[17] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 4601–4607.

[18] W. L. Hamilton, *Graph Representation Learning* (Synthesis Lectures on Artificial Intelligence and Machine Learning), 1st ed. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-031-01588-961-5.

[19] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, "Relational graph learning for crowd navigation," 2019, *arXiv:1909.13165*.

[20] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR) Conf. Track*, 2016, pp. 1–21.

[22] R. Chai, H. Niu, J. Carrasco, F. Arvin, H. Yin, and B. Lennox, "Design and experimental validation of deep reinforcement learning-based fast trajectory planning and control for mobile robot in unknown environment," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Oct. 10, 2022, doi: 10.1109/TNNLS.2022.3209154.

[23] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell. (AAAI)*, 2016, pp. 2094–2100.

[25] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 3, 2017, pp. 2171–2186.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn. (ICML)*, vol. 1, 2014, pp. 605–619.

[27] A. Harutyunyan, T. Stepleton, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1054–1062.

[28] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, vol. 3, 2015, pp. 1889–1897.

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

[30] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 4, 2018, pp. 2587–2601.

[31] T. Haarnoja et al., "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 5, 2018, pp. 2976–2989.

[33] P. Christodoulou, "Soft actor-critic for discrete action settings," 2019, *arXiv:1910.07207*.

[34] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao, "Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32 no. 12, pp. 5435–5444, Dec. 2021.

[35] P. Christodoulou, "Soft actor-critic for discrete action settings," 2019, *arXiv:1910.07207*.

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

[37] S. J. Guyy et al., "ClearPath: Highly parallel collision avoidance for multi-agent simulation," in *Proc. Comput. Animat. Conf.*, 2009, pp. 177–187.

[38] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Int. Symp. Robot. Res.*, 2009, pp. 3–19.

[39] E. Bas, "An introduction to Markov chains," in *Basics of Probability and Stochastic Processes*. Wiesbaden, Germany: Vieweg+Teubner Verlag Wiesbaden, 2019, pp. 179–198.

[40] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Machine Learning Proceedings*. San Mateo, CA, USA: Morgan Kaufmann, 1995, pp. 30–37.

[41] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[42] Z. Lin et al., "A structured self-attentive sentence embedding," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR) Conf. Track*, 2017, pp. 1–15.

[43] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[44] C. Banerjee, Z. Chen, and N. Noman, "Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 19, 2022, doi: 10.1109/TNNLS.2022.3174051.

Chengmin Zhou (Student Member, IEEE) is currently a Ph.D. Researcher with the University of Eastern Finland, Joensuu, Finland. He has many publications, including two articles in top journals. His research interests include robotics, deep learning, reinforcement learning, representation learning, path planning, and motion planning.

Bingding Huang received the master's degree from the Max Planck Institute, Munich, Germany, in 2004, and the Ph.D. degree from the Technical University of Dresden, Dresden, Germany, in 2007.

He is currently a Distinguished Professor at Shenzhen Technology University, Shenzhen, China. His research interests include machine learning, bioinformatic algorithms, and data analysis.

Pasi Fränti (Senior Member, IEEE) has been a Professor and a Team Leader of the Machine Learning Group, University of Eastern Finland, Joensuu, Finland, since 2000.

He is one of the most prolific authors in Finland. His current research interests include machine learning, data mining, and location-based services.