Robust Motion Planning for Multi-Robot Systems Against Position Deception Attacks

Wenbing Tang, Yuan Zhou, *Member, IEEE*, Yang Liu, *Senior Member, IEEE*, Zuohua Ding, *Member, IEEE*, and Jing Liu, *Member, IEEE*

***Abstract*— Deep reinforcement learning (DRL) is widely applied in motion planning for multi-robot systems as DRL leverages the offline training process to improve the real-time computation efficiency. In DRL-based methods, the DRL models compute an action for a robot based on the states of its surrounding obstacles, including other robots in the system. They always assume that the number of obstacles is fixed and the obtained obstacles' states are reliable. However, in the real world, a multi-robot system may suffer from various attacks, such as remote control attacks and network attacks, that cause wrong positions of the surrounding obstacles received by a robot. In this paper, we propose a robust motion planning method `DAE-Crit-LSTM`, integrating a denoising autoencoder (DAE) with DRL models, to mitigate such position deception attacks in environments with a different number of obstacles. `DAE-Crit-LSTM` shows the following two advantages. First, `DAE-Crit-LSTM` can be applied in benign and attacked scenarios and thus does not require any detector. It learns an encoder and a decoder to approximate the accurate positions of the obstacles, no matter under attack or not. Second, `DAE-Crit-LSTM` applies an LSTM (Long Short-Term Memory)-based DRL model to deal with a variable number of obstacles in the environment. It is worth noting that `DAE-Crit-LSTM` is method-agnostic and can be easily implemented in state-of-the-art motion planning methods. Comprehensive experiments show that `DAE-Crit-LSTM` can mitigate position deception attacks and guarantee safe motion. We also demonstrate the effectiveness and generalization of `DAE-Crit-LSTM`.**

***Index Terms*— Deep reinforcement learning, denoising autoencoder, motion planning, multi-robot systems, position deception attacks.**

Manuscript received 29 November 2022; revised 12 October 2023; accepted 18 December 2023. Date of publication 25 December 2023; date of current version 3 January 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFC3302600; in part by the National Natural Science Foundation of China under Grant 61972150 and Grant 62132014; in part by the Fundamental Research Funds for Central Universities of China; in part by the Zhejiang Provincial Key Research and Development Program of China under Grant 2022C01045; in part by the Academic Research Fund Tier 2 by the Ministry of Education in Singapore under Grant MOE-T2EP20120-0004; in part by the National Research Foundation, Singapore; in part by Defence Science Organisation (DSO) National Laboratories under the AI Singapore Program (AISG) under Award AISG2-GC-2023-008; and in part by the National Research Foundation (NRF) Investigatorship under Grant NRF-NRFI06-2020-0001. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. George Loukas. *(Corresponding authors: Jing Liu; Yuan Zhou.)*

Wenbing Tang and Jing Liu are with the Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China (e-mail: wenbingtang@hotmail.com; jliu@sei.ecnu.edu.cn).

Yuan Zhou is with the Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China, and also with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: y.zhou@ntu.edu.sg).

Yang Liu is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: yangliu@ntu.edu.sg).

Zuohua Ding is with the School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou, Zhejiang 310018, China (e-mail: zouhuading@hotmail.com).

Digital Object Identifier 10.1109/TIFS.2023.3346647

I. Introduction

A MULTI-ROBOT system is a system containing multiple robots that can cooperate and communicate with each other to accomplish tasks by moving around in a given environment. With the collective behavior and distributed decision of multiple robots, multi-robot systems can increase functionality, improve efficiency, extend flexibility, and enhance reliability, compared with their single-robot counterparts [1], [2]. Multi-robot systems have been applied in a wide range of applications, such as disaster rescue, traffic monitoring, cargo delivery, industrial assembly, space and underwater explorations, and military operations [1], [3], [4].

To accomplish complicated tasks cooperatively, one of the most important requirements in multi-robot systems is collision-free motion planning. It aims to move robots from their initial positions to the given targets without causing collisions. Many methods have been proposed for collision-free motion planning, such as velocity obstacles [5], [6], mathematical programming [7], [8], and deep reinforcement learning (DRL) [9], [10], [11]. By combing reinforcement learning with deep neural networks, DRL gains a better trade-off between motion efficiency and computation efficiency and has become a promising technology for robot motion planning [12]. DRL methods generally take the robot's state (e.g., position, velocity, and target) and the surrounding obstacles' observable states (e.g., position and velocity) as input to predict a collision-free action [10].

Current DRL-based motion planning algorithms always assume that a robot can obtain accurate information about its surrounding obstacles. However, in the real world, robots are vulnerable to various attacks, resulting in receiving wrong information about the obstacles [13], [14], [15], [16], [17], [18], [19]. Position deception attacks (PDAs) are one of the widely existing attacks in multi-robot systems, where an attacker can tamper with the positions of environmental obstacles (including other robots and the surrounding obstacles) and report the malicious data to the planning module of a robot [13], [20], [21], [22], [23], [24], [25]. Consequently, the

robot will make wrong decisions and cause severe accidents, e.g., collisions. For example, a robot may be intruded into by an attacker and send wrong position messages to others [13], [20]. The onboard sensors of a robot may be manipulated by an attacker to perceive a designated position (e.g., a non-existent obstacle) [21], [25]. An attacker may also perform attacks against sensor signals to modify the transmitted position data [23], [24].

Although some technologies have been made to deal with position attacks in robotics, they usually focus on attack detection. For example, control invariants [26], cumulative sum statistics [27], and deep learning methods [28] represent the state-of-the-art sensor attack detection techniques. However, on one hand, when an attack is detected, they have limited responses (raising alarms or activating failure modes) to the attack; on the other hand, they may cause high false positive rates, resulting in unnecessary responses. In addition, most of the current attack mitigation methods, such as software sensors-based recovery (SSR) [29] and PID-Piper [30], are proposed for a single robot. The performance degrades significantly when they are applied to mitigate attacks against multiple robots simultaneously (empirical evidence will be given in our experiments).

In this paper, focusing on mitigating the effects of PDAs in multi-robot systems, we propose a detector-agnostic motion planning method DAE-Crit-LSTM. On the one hand, DAE-Crit-LSTM proposes a Denoising AutoEncoder (DAE) [31] to estimate the real positions of the obstacles, as a well-trained DAE has the ability to restore uncorrupted values (i.e., real positions of obstacles) by learning some intrinsic properties from the corrupted ones (i.e., attacked positions) [32]. It means that DAE-Crit-LSTM can maintain benign positions while correcting attacked ones without any detector. Therefore, DAE-Crit-LSTM is detector-agnostic. On the other hand, DAE-Crit-LSTM utilizes a long short-term memory (LSTM)-based DRL method [9] to deal with a variable number of obstacles in the environment. However, DAE-Crit-LSTM is method-agnostic and can be applied with any existing DRL motion planning method.

Specifically, DAE-Crit-LSTM is implemented with two modules: a DAE-based module to process the originally received positions of obstacles and a DRL module to compute collision-free actions. The DAE module contains an encoder and a decoder. First, the encoder takes the received position with the recorded previous position and velocity of each obstacle as input to generate the latent representation of each obstacle's current position. Then, the decoder retrieves the approximate real position based on the encoded latent representation. The DRL module contains an LTSM model to learn a feature vector of the surrounding environment based on the approximated obstacles' states and a value network to compute the value of each action candidate based on the learned feature and the state of the robot. Finally, the action with the maximal value is selected to control the motion of the robot.

To train and test DAE-Crit-LSTM, we design six types of PDAs: biased attack (adding random values), percentage attack (multiplying a weight), zero replacement attack (replacing by a zero vector), delay attack (receiving the previous positions), rotation attack (rotating the position vectors), and physical removal attack (removing existent obstacle positions). We train DAE-Crit-LSTM under the first three types of attacks and test the model using all six attacks, including three attacks that are not experienced during the training phase. To comprehensively evaluate the effectiveness and efficiency of DAE-Crit-LSTM, we firstly perform DAE-Crit-LSTM on various scenarios with a variable number of obstacles. The results show that DAE-Crit-LSTM can mitigate different PDAs effectively, improving the success rate from 77.5% to 98.1%. Then, we compare our DAE-based mitigation strategy with other attack recovery methods, and the results show our strategy consistently outperforms the best baseline (i.e., SSR), increasing the success rate by 8.9% (from 88.2% to 97.1%). Finally, we integrate the proposed DAE-based mitigation strategy into other DRL methods to demonstrate the generalization of DAE-Crit-LSTM.

The main contributions of this paper are threefold:

- We propose the first DAE-based strategy to mitigate PDAs in multi-robot systems without any attack detector.
- We propose a robust motion planning method, DAE-Crit-LSTM, for multi-robot systems moving around in adversarial environments with a variable number of robots.
- We conduct extensive experiments to demonstrate the effectiveness and efficiency of DAE-Crit-LSTM.

The rest of this paper is organized as follows. Section II is a brief review of related work. The problem statement is given in Section III. Section IV gives an overview of DAE-Crit-LSTM. Sections V and VI present the detailed procedures for the DAE-based attack mitigation strategy and the robust motion planning method, respectively. Experiments are conducted in Section VII to demonstrate the effectiveness and generalization of our approach. Conclusion and future work are finally provided in Section VIII.

II. Related Work

This paper is related to the topic of motion planning, which is a key and popular topic in robotics. Many methods have been proposed in this area. They can be mainly divided into conventional methods and DRL-based methods. The conventional methods, such as velocity obstacles [5], [6], rely on the precise models of robots and/or the environment. However, since they lack self-learning ability, it is inefficient to apply them to crowded dynamic environments. In addition, most conventional motion planning methods cannot guarantee computation efficiency and smooth motion simultaneously [33].

Recently, since there is a better trade-off between smooth motion and computation efficiency [10], [11], DRL has been applied in motion planning. According to their inputs, the current DRL methods can be divided into two categories: sensor-based DRL methods [34], [35], which take the raw data from sensors as inputs, and agent-based DRL methods [10], [11], which take the preprocessed agent-level information (e.g., types and states) as inputs. In practice, with such agent-level information, the DRL models can make more precise

decisions [36]. However, one challenge for agent-based DRL models is the variable number of obstacles. To handle a variable number of obstacles, the authors in [37] proposed a DRL method with an LSTM model, where the LSTM model is used to transform the obstacles' states into a fixed-size hidden state based on their distances to the robot. Inspired by [37], Xu et al. [9] proposed a novel LSTM model, Crit-LSRM, to encode a variable number of obstacles, where the states of obstacles are ordered according to their collision risks with the robot. However, these methods assume that the obtained obstacles' states are always reliable, which is inappropriate for today's adversarial environments containing various PDAs. Hence, in this paper, we propose a robust DRL motion planning method for multiple robots moving around in adversarial environments.

In the case of PDAs, some attack recovery techniques have been proposed. However, the current methods always require detecting the attacks in advance [38], which is still a significant challenge. For example, using system identification techniques, SSR [29] generates an approximate model of the corresponding physical sensor; the model can predict and replace the corresponding sensor data when the accumulated error exceeds a given threshold. In [30], an LSTM model is applied to correct attacked sensors' data and construct an attack-resilient controller running concomitantly with the primary PID controller; the backup controller will replace the primary one when attacks against sensors are detected. In [20], a replay-based recovery strategy is proposed to navigate a robot out of the attack zone by replaying the historic states collected from the attack-free phase. They are proposed for single-robot systems. Different from these works, our method is proposed for multi-robot systems. In addition, our method does not require any attack detector as a prerequisite.

III. Background and Problem Statement

In this section, we first present the optimal motion planning problem in multi-robot systems. Second, we present the threat model of PDAs. Finally, we give a motivating example and the problem statement of the robust motion planning for such systems with PDAs.

A. Motion Planning in Multi-Robot Systems

In this paper, we consider the collision-free motion planning problem for a multi-robot system moving in an adversarial environment. Assume that there are $N (N \geq 2)$ holonomic robots in the system. Given $\mathbb{I}_N = \{1, 2, \ldots, N\}$, $\forall i \in \mathbb{I}_N$, r^i denotes a robot in the system. The motion task is to move all robots from their initial positions $\overline{\mathbf{P}} = \{\overline{\mathbf{p}}^1, \overline{\mathbf{p}}^2, \ldots, \overline{\mathbf{p}}^N\}$ to the target positions $\underline{\mathbf{P}} = \{\underline{\mathbf{p}}^1, \underline{\mathbf{p}}^2, \ldots, \underline{\mathbf{p}}^N\}$ within a given duration, where $\forall i, j \in \mathbb{I}_N$, $i \neq j$, $\overline{\mathbf{p}}^i \neq \overline{\mathbf{p}}^j$, and $\underline{\mathbf{p}}^i \neq \underline{\mathbf{p}}^j$. To guarantee safety, each robot needs to avoid collisions with other robots during its motion. At any time t, two robots r^i and r^j are in a collision if $||\mathbf{p}_t^i - \mathbf{p}_t^j||_2 < \rho^i + \rho^j$, where ρ^i and ρ^j are the safe radii of r^i and r^j, respectively. Furthermore, each robot has a maximum speed v_{max}^i and a preferred speed v_p^i.

Suppose $[0, \tau^i]$ is the motion interval for robot r^i to move, which can be divided into a set of discrete time instants with an equal time step Δt. For simplicity, we assume the only obstacles for a robot are other robots in the system. However, other obstacles can be handled in the same way as robots. Hence, at any time instant t, suppose the detected robots are $\mathcal{O}^i = \{r^{i_1}, r^{i_2}, \ldots, r^{i_n}\}$, whose observable states are $\mathbf{S}_t^i = \{\mathbf{so}_t^{i_1}, \cdots, \mathbf{so}_t^{i_n}\}$, where $\mathbf{so}_t^j = \{\mathbf{p}_t^j, \mathbf{v}_t^j, \rho^j\}$ is the observable state of r^j, and $\mathbf{p}_t^j = (x_t^j, y_t^j) \in \mathbb{R}^2$, $\mathbf{v}_t^j = (vx_t^j, vy_t^j) \in \mathbb{R}^2$, and ρ^j are the position, velocity, and safe radius of r^j, respectively. Then, the discretized motion of r^i can be formalized as:

$$\begin{aligned} \mathbf{p}_\tau^i &= \mathbf{p}_t^i + (\tau - t\Delta t)\mathbf{v}_t^i, \ \ \tau \in [t\Delta t, (t+1)\Delta t) \\ \|\mathbf{p}_\tau^i - \mathbf{p}_\tau^j\| &\geq \rho^i + \rho^j, \forall\ \mathbf{so}_\tau^j = (\mathbf{p}_\tau^j, \mathbf{v}_\tau^j, \rho^j) \in \mathbf{S}_\tau^i \\ \|\mathbf{v}_\tau^i\|_2 &\in [0, v_{max}^i], \\ \mathbf{p}_0^i = \overline{\mathbf{p}}^i, &\quad \mathbf{p}_{\tau^i}^i = \underline{\mathbf{p}}^i. \end{aligned}$$

It is worth noting that in a multi-robot system, to achieve collaboration, each robot r^i can retrieve other robots' observable states by fusing the perception information from its onboard sensors (e.g., LiDAR and Radar) and other robots (via robot communication) [39].

Finally, the optimal motion planning considered in this paper is to determine a velocity control policy π^i such that the robot can arrive at its target position as soon as possible without any collision. Formally, the optimal motion planning problem can be described as:

$$\arg\min_{\pi} \ T \tag{1}$$

$$s.t.\ \mathbf{p}_0^i = \overline{\mathbf{p}}^i, \ \mathbf{p}_T^i = \underline{\mathbf{p}}^i, \quad \forall i; \tag{2}$$

$$\|\mathbf{p}_t^i - \mathbf{p}_t^j\| \geq \rho^i + \rho^j, \quad \forall i \neq j, \forall t \in \{0, 1, \ldots, T\}; \tag{3}$$

$$\mathbf{p}_{t+1}^i = \mathbf{p}_t^i + \pi^i(\mathbf{s}_t^i, \mathbf{S}_t^i) \cdot \Delta t, \quad \forall i, \forall t. \tag{4}$$

where $\pi^i(\mathbf{s}_t^i, \mathbf{S}_t^i)$ is the velocity to be determined at t, and $\mathbf{s}_t^i = (\mathbf{p}_t^i, \mathbf{v}_{t-1}^i, \underline{\mathbf{p}}^i, v_p^i, \rho^i) \in \mathbb{R}^8$ is the state of r^i at t. Here the motion time T in (1) is the optimization objective, (2) is the target constraints, (3) is the safety (collision avoidance) constraints, and (4) is the kinematics of robots.

B. Position Deception Attacks (PDAs)

1) Threat Model: In this paper, we consider position deception attacks (PDAs) in multi-robot systems. Particularly, we assume the positions of some robots in a multi-robot system are malicious and can be modified by the adversary, which will be sent to the motion planning module and then compromise the entire system. There are several reasons that make this assumption realistic. (1) Some adversarial robots under such as Byzantine attacks can send malicious messages to others to affect their decisions indirectly. A Byzantine attack is successfully demonstrated against multi-robot systems in [13], where the attacked robot can arbitrarily tamper with the messages sent from it at any time. (2) The equipped sensors under attack cannot generate accurate signals for localizing the positions of other robots accurately. Previous works have confirmed the feasibility of removing existent obstacles [21] or adding non-existent obstacles [25], in the LiDAR point cloud data. (3) The sensor signals are vulnerable to communication

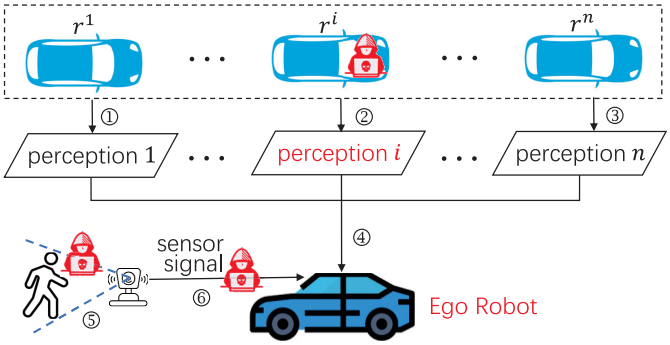


Fig. 1. The adversary performs position deception attacks in a multi-robot system via different attack ways.

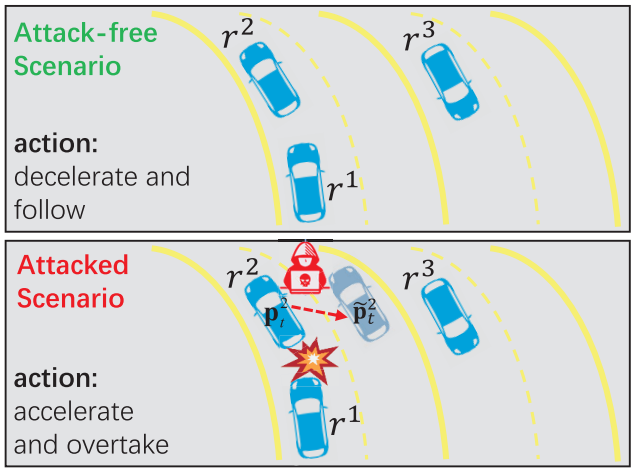


Fig. 2. An example to illustrate the effect of position deception attacks on the motion planning of multi-robot systems.

attacks, and the attacker can modify the transmitted positions before the data is used as input to the planning module. For instance, Zhou et al. [40], [41] investigated communication attacks/failures in multi-robot systems, and found that the data transmission channel can be easily jammed and disrupted by the adversary. Recently, Jang et al. [23] implemented an attack to distort the sensor communication channel using electromagnetic interference.

We assume that the PDAs can be time-varying with time elapsing. It means that a robot r^i can be under attack at some time instant while attack-free at another time instant. Hence, as shown in Fig. 1, we assume that during the motion of robots, the attacker has the following capabilities. (1) Manipulating a robot to send a fake position to others. For example, the adversary falsifies the real position of r^i and reports a fake perception state to the robot through the path ②→④. (2) Spoofing the related sensors (e.g., LiDAR) such that they cannot localize the surrounding robots accurately. For example, by removing the point cloud of the selected object in LiDAR sensor data, e.g., laser-based spoofing attack against the LiDAR's FOV (Field of View) [21], the adversary prevents the robot from detecting surrounding safety-critical obstacles. The wrong data will propagate along the attack path ⑤→⑥ and affect the subsequent planning results. (3) Modifying the position signals transmitted in the communication channel between the sensors and planning module. For example, even if the LiDAR sends a correct signal of the obstacle's position to the communication channel, the adversary can still modify the digital signal during the path ⑥. Consequently, a malicious position will be received by the robot, which is further converted into wrong actions.

It is important to note that modern robots typically rely on a diverse array of sensors, including IMUs, GPS, LiDAR, and cameras, to comprehensively perceive both their own state and the surrounding obstacles [29]. These sensors serve distinct roles: a gyroscope measures angular velocities, an accelerometer tracks linear accelerations, GPS provides position data, and LiDAR or cameras identify nearby objects. They introduce various attacks, and different attacks focus on different attack targets. For example, position attacks (e.g., GPS spoofing [17], LiDAR attack [21], [25], and localization attack [42]) are generally designed against localization sensors [23], and often conducted in position-sensitive tasks (e.g., motion planning and formation control [42], [43]), while velocity attacks are generally targeted at speed measurement sensors (e.g., IMU) [43]. In this paper, we focus on position attacks, while assuming other non-position information is reliable. In the future, we will consider velocity attacks and explore the scenario when both position and velocity are attacked simultaneously.

We also assume the attacker cannot access the control software and hence cannot bypass the DAE in `DAE-Crit-LSTM`. In addition, all robots are assumed to be able to locate their own positions accurately. Several technologies can guarantee this assumption, such as multiple sensor fusion [27], [44].

2) Attack Example: Under PDAs, the attacked robots send faked positions to the motion planning module. Consequently, the positions received by a robot may be poisoned by an attacker. As shown in Fig. 1, all robots form a cooperative system. Each obstacle sends its own perception information to the robot. The robot also perceives the positions of other robots using the equipped sensors. At the current time t, suppose the real position of r^i is $\mathbf{p}_t^i = (x_t^i, y_t^i)$. However, due to PDAs, the received position by the robot is a faked position $\tilde{\mathbf{p}}_t^i = (\tilde{x}_t^i, \tilde{y}_t^i)$. With such a faked position, the motion planning process will generate a wrong action and put the robot in danger.

C. Motivating Example and Problem Statement

In the case of PDAs, there are position errors in the received obstacles' observable states. We use the system shown in Fig. 2 as a motivating example to demonstrate the detrimental effects of such errors on motion planning. In this example, there are three robots (i.e., r^1, r^2, and r^3), where r^2 is running in front of the ego robot r^1. In the attack-free scenario, r^1 should decelerate and follow the front robot to avoid potential collisions. However, in the attacked scenario, an attacker tampers with the real position $\mathbf{p}_t^2$ to a fake position $\tilde{\mathbf{p}}_t^2$ that indicates r^2 is on the right lane. Then, based on the fake position, r^1 will mistakenly accelerate, and the safety distance will be reduced, which will cause a collision. We can observe that although only one robot is attacked, the decisions of some robots in the system will be significantly affected, which can cause serious accidents.

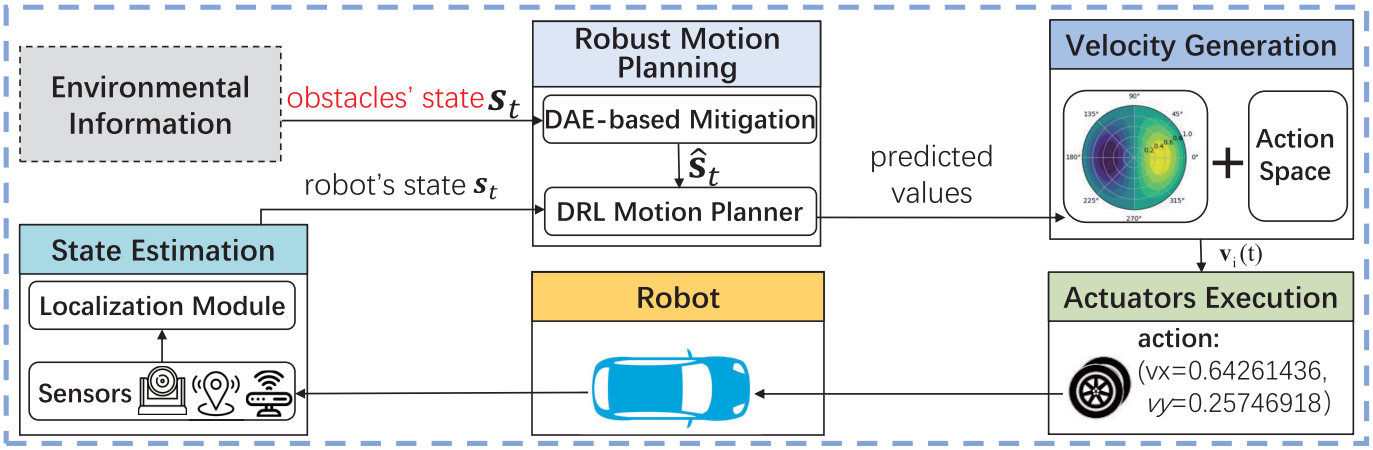


Fig. 3. The feedback control process of a robot with our proposed robust motion planning method.

Based on the above descriptions, to generate a collision-free trajectory for a robot in an adversarial environment existing PDAs, the robot needs to mitigate the detrimental effect of attacks and determine its velocity. Hence, the problem studied in this paper can be described as follows:

Problem 1: *Given a multi-robot system with N robots moving around in an adversarial environment with PDAs, design a robust motion planning method such all robots can move toward their target positions safely without any attack detector.*

IV. Approach Overview

To deal with PDAs and generate collision-free motion actions, a DAE-aided DRL motion planning method, `DAE-Crit-LSTM`, is proposed to generate collision-free motion for each robot. This section gives the high-level architecture of the proposed `DAE-Crit-LSTM`, while the details are given in the following sections.

Fig. 3 shows the control process of a robot under `DAE-Crit-LSTM`. Typically, the state estimation module and the robust motion planning module are executed in a feedback loop. In the loop, the state estimation module uses multi-sensor measurements to determine the state of the robot. In practice, sensor fusion techniques are often used, e.g., Kalman filter and particle filter, to reduce uncertainty and produce more accurate state estimation [44]. Then, the robot's estimated state and the obstacles' observable states (which may be under PDAs) are taken as the input of the motion planning module. Next, the planning module computes an optimal action for a robot according to its current state and the received obstacles' observable states. Finally, the action will be sent to the robot to execute. By repeating this loop, the robot moves toward its target, while dealing with the encountered obstacles and PDAs.

As shown in Fig. 3, the main idea of `DAE-Crit-LSTM` implemented in the motion planning module is to correct the attacked positions of obstacles via a well-trained DAE model, and then generate collision-free actions using a well-trained DRL model. Hence, it mainly contains two sub-modules, i.e., DAE-based mitigation and DRL motion planner.

A. DAE-Based Mitigation

For each robot, the obstacles' observable states, mainly positions, obtained from the environmental information may be poisoned by the attacker. Hence, robots need to prevent the effect of the attacks from propagating in the control loop. In this paper, a DAE-based mitigation strategy is proposed to mitigate PDAs. It aims to learn an encoder and a decoder to correct the potential positions of obstacles according to the attacked ones and maintain the real-time motion performance of robots. Hence, in `DAE-Crit-LSTM`, the obstacles' positions are first pre-processed by the DAE model, and new observable states are generated to approximate the real observable states of the obstacles.

B. DRL Motion Planner

The DRL motion planner takes the corrected observable states from DAE and the robot's state as input to compute an action. It contains an LSTM model and a value network. First, the corrected observable states are sorted according to their risks to the robot. Then, the LSTM model takes the ordered observable states as input and generates a unified hidden state. Finally, the value network computes the value of each action candidate based on the hidden state and the robot's current state, and the action with the maximal value is selected as the final action for the robot to execute.

V. Attack Mitigation Strategy

In this section, we detail the DAE-based attack mitigation strategy. We first give a brief introduction to DAEs and then describe the design of our DAE-based mitigation strategy.

A. Denoising Autoencoders (DAEs)

DAE is a powerful deep learning model to learn robust representations of the input data using unsupervised learning [31], [32]. It is a stochastic version of the standard autoencoder model. The main idea of DAE is that the generated representations are robust against the introduction of noise in the input data. A DAE model is typically composed of an encoder and a decoder. Given an input $z \in \mathbb{R}^d$, the encoder maps a high-dimension noisy/corrupted input $\tilde{z}$ to a low-dimension latent

representation $h \in \mathbb{R}^l$ $(l < d)$. Then, the decoder maps h back to a reconstructed output $\hat{z} \in \mathbb{R}^d$ such that z and $\hat{z}$ are as similar as possible.

In practice, the encoder and the decoder are two neural networks. Training is carried out in an unsupervised manner. DAE first corrupts the original input z into $\tilde{z}$ by adding some additional noises (e.g., Gaussian noises) or masking some elements randomly, i.e., $\tilde{z} = z \oplus \sigma$, where $\oplus$ denotes the element-wise corruption operation. Then, the corrupted input $\tilde{z}$ is mapped to a reconstructed output $\hat{z}$ through the encoder and the decoder. Finally, the DAE learns to minimize the reconstruction error $L(z, \hat{z})$, between the original noiseless input z and the reconstructed output $\hat{z}$, where $L(\bullet)$ is a loss function, such as mean square error. Formally, the training process can be described as:

$$\begin{aligned} \min_{\varphi,\theta} \quad & L(z, \hat{z}) = mse(z, \hat{z}) \\ s.t. \quad & \tilde{z} = z \oplus \sigma, \\ & h = M_{enc}(\tilde{z}; \varphi), \\ & \hat{z} = M_{dec}(h; \theta). \end{aligned}$$

where σ is a vector of noise sampled from a noise distribution, $M_{enc}(\cdot)$ is the encoder neural network and φ is the network weights, $M_{dec}(\cdot)$ is the decoder neural network and θ is the corresponding network weights.

When a DAE model is trained, it has two capabilities: (1) encoding the inputs to preserve the essential information and (2) eliminating the effects of the noises added to the inputs. That is, besides good feature representation learning, the DAE also performs noise elimination, which is the main characteristic of the application of DAEs to mitigate attacks. DAEs have been introduced in many applications, such as image denoising [45], speech enhancement [46], missing data imputation [47], and robust feature learning [48], [49]. In these applications, the encoded robust representations are generally applied to a set of downstream tasks. However, in this paper, we apply DAEs to generate the potential positions for the obstacles under PDAs, which is based on the reconstructed output.

B. DAE-Based Mitigation Strategy

Inspired by the application of DAEs in denoising, in the sequel, we propose a DAE-based attack mitigation strategy. Its architecture is shown in Fig. 4. Our main idea is to train a DAE on an attack-free dataset, and then generate an approximate value of the attacked position using the trained DAE. As the training data is collected in an offline manner and it is not mandatory to guarantee the same environments for training data collection and real-world deployment, we can use a simulator or construct a trusted environment to collect attack-free data. In addition, existing attack detectors can be applied to detect whether there are attacks in the collected dataset. Once an attack-free dataset is collected, during the training phase, by adding various PDAs to attack-free inputs, the DAE model learns how to eliminate the effects of attacks and generate the reconstructed output as close to the original one as possible. Thus, given an attacked position as the input, the well-trained encoder and decoder models can restructure the original attack-free position effectively. The detailed training process is given as follows.

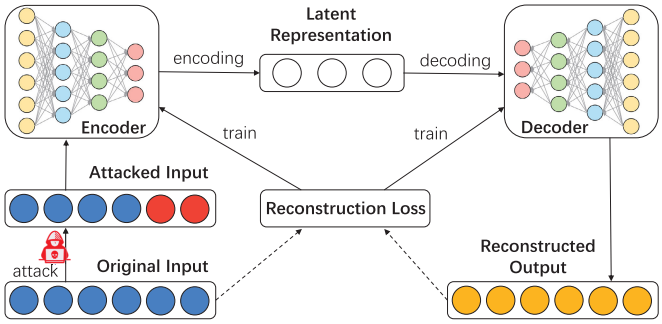


Fig. 4. Proposed DAE architecture for attack mitigation.

(1) Attacked Input Generation. Given an obstacle r^j, the training data of a DAE is defined as:

$$z_t^j = (\mathbf{p}_{t-1}^j, \mathbf{v}_{t-1}^j, \mathbf{p}_t^j) \in \mathbb{R}^6$$

where $\mathbf{p}_{t-1}^j = (x_{t-1}^j, y_{t-1}^j)$ and $\mathbf{v}_{t-1}^j = (vx_{t-1}^j, vy_{t-1}^j)$ are the recorded position and velocity at $t-1$, and $\mathbf{p}_t^j = (x_t^j, y_t^j)$ is the current position of r^j. Note that the current position relies on the previous position and speed. To learn the potential latent representations accurately, we take $\mathbf{p}_{t-1}^j$ and $\mathbf{v}_{t-1}^j$ as a part of the input to restructure the current position.

The training process will first generate a faked input $\tilde{z}_t^j = (\mathbf{p}_{t-1}^j, \mathbf{v}_{t-1}^j, \tilde{\mathbf{p}}_t^j)$ from each training data z_t^j by introducing various attacks to position $\mathbf{p}_t^j$. The attacked position $\tilde{\mathbf{p}}_t^j$ can be written as $\tilde{\mathbf{p}}_t^j = \mathbf{p}_t^j \oplus \sigma$, where σ is generated from one of PDAs, such as the biased attacks. Then the faked inputs are sent to the encoder $M_{enc}(\cdot)$.

(2) Encoding. The encoder M_{enc} is a multi-layer perceptron (MLP), which takes the faked data $\tilde{z}_t^j$ as input and generates a hidden representation h_t^j for the input. Formally, the encoding process can be described as:

$$h_t^j = M_{enc}(\tilde{z}_t^j, \varphi)$$

where $\tilde{z}_t^j = (\mathbf{p}_{t-1}^j, \mathbf{v}_{t-1}^j, \tilde{\mathbf{p}}_t^j)$, and φ is the network weight to be learned.

(3) Decoding. The decoder M_{dec} is another MLP. It takes the representation h_t^j as the input, and outputs the reconstructed value $\hat{z}_t^j$, i.e.,

$$\hat{z}_t^j = M_{dec}(h_t^j, \theta)$$

where θ is the weight of the decoder that needs to be learned.

(4) Reconstruction Loss. Given the training dataset $D^j = \{z^j\}$, the original learning process aims to minimize the average error between z^j and $\hat{z}^j$ over the dataset, i.e.,

$$L_{rec} = \frac{1}{|D^j|} \sum_{z^j \in D^j} ||z^j - \hat{z}^j||_2 \tag{5}$$

In this paper, as we aim to generate an approximate of each position $\mathbf{p}_t^j$, we also introduce a position reconstruction error:

$$L_{pos} = \frac{1}{|D^j|} \sum_{D^j} ||\mathbf{p}^j - \hat{\mathbf{p}}^j||_2 \tag{6}$$

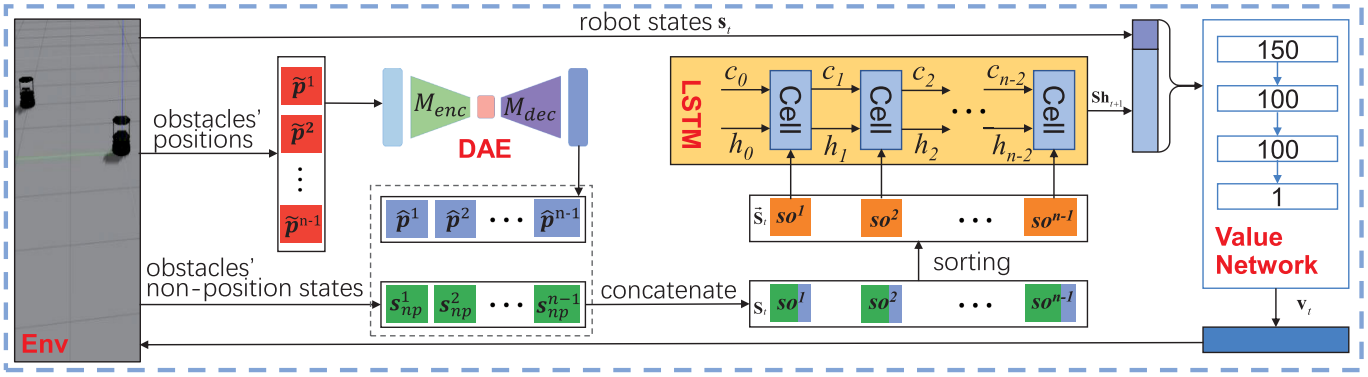


Fig. 5. The architecture of our robust motion planning method DAE-Crit-LSTM.

where $\mathbf{p}^j$ is the original position in z^j, and $\hat{\mathbf{p}}^j$ is the reconstructed position in $\hat{z}^j$. L_{pos} encourages the encoder and the decoder to reconstruct positions that are as close to the original ones as possible.

Therefore, by combining (5) and (6), we can obtain the final loss function for the training process:

$$L_{DAE} = L_{rec} + \lambda L_{pos}, \tag{7}$$

where λ is a hyper-parameter that controls the relative importance of these two losses. Finally, the encoder and decoder can be obtained by minimizing (7), i.e.,

$$\varphi^*, \theta^* = \arg_{\varphi,\theta} \min \ L_{DAE}. \tag{8}$$

When the encoder and the decoder are well-trained, at any time instant t of the inference phase, they take the received state z_t^j for each robot r^j, whether is under attack or not, as input and generate the approximate state $\hat{z}_t^j$:

$$\hat{z}_t^j = DAE(z_t^j) = M_{dec}(M_{enc}(z_t^j, \varphi^*), \theta^*) \tag{9}$$

VI. Robust Motion Planning

Following the DAE-based mitigation strategy, we first propose our robust motion planning method DAE-Crit-LSTM. Then, we demonstrate the distributed execution of DAE-Crit-LSTM in a multi-robot system.

A. DRL Framework for Motion Planning

As stated in [37], the optimal motion planning problem described in (1)–(4) can be resolved efficiently under the following DRL framework:

$$\mathbf{v}_t^* = \arg\max_{\mathbf{v}\in\mathcal{A}} V(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v}), \tag{10}$$

$$V(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v}) = R(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v}) + \gamma^{\Delta t \cdot v_p} V^*(\mathbf{s}_{t+1;\mathbf{v}}, \mathbf{S}_{t+1}) \tag{11}$$

$$V^*(\mathbf{s}_t, \mathbf{S}_t) = \sum_{t'=t}^{T-1} \gamma^{(t'-t)\cdot\Delta t\cdot v_p} R(\mathbf{s}_{t'}, \mathbf{S}_{t'}, \mathbf{v}_{t'}^*) \tag{12}$$

where $\mathcal{A}$ is the set of candidate actions, $R(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v})$ is the one-step reward by performing action $\mathbf{v}$ at $(\mathbf{s}_t, \mathbf{S}_t)$ (the corresponding next state is $\mathbf{s}_{t+1;\mathbf{v}}$), $\gamma \in [0, 1]$ is a discount factor, $V^*(\cdot)$ is the optimal value function. In practice, the value function can be approximated by a deep neural network (DNN). Following [9], [37], the reward function is defined as follows:

$$R(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v}_t) = \begin{cases} 1 & \text{if } \mathbf{p}_{t+1} = \underline{\mathbf{p}}, \\ -0.25 & \text{if } d_{min} \le 0, \\ -0.1 + 0.5 d_{min} & \text{if } 0 < d_{min} \le 0.2, \\ 0 & \text{otherwise.} \end{cases}$$

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

B. Robust DRL Method DAE-Crit-LSTM

Fig. 5 shows the architecture of DAE-Crit-LSTM. At any time instant, the received obstacles' positions are first pre-processed by the well-trained DAE model; then, the resulted states are sent to an LTSM model to generate a unified hidden state; finally, the robot's state and the hidden state are sent to a value network to compute the corresponding value.

In the sequel, we illustrate the process of training DAE-Crit-LSTM. It contains two stages. The first one is training a DAE model on the training dataset $D = \{(\mathbf{p}_{t-1}, \mathbf{v}_{t-1}, \mathbf{p}_t)\}$. The second one is training a DRL model (i.e., the LSTM model and the value network) according to $\mathbf{s}_t$ and $\hat{S}_t = \{(\hat{\mathbf{p}}_t^j, \mathbf{v}_t^j, \rho^j)\}$, where $\hat{\mathbf{p}}_t^j$ is generated by the DAE model trained at the first stage. Algorithm 1 describes the detailed training process.

Lines 1–8 describe the training process for the DAE model. First, the training dataset is generated from ORCA method [6] (Line 1). Second, the encoder model M_{enc} and the decoder model M_{dec} are initialized randomly (Line 2). Third, the algorithm repeats η_{dae} episodes to train the DAE model. For each episode, the DAE model generates the attacked input by adding attacks on the original one (Lines 4–5) and updates the weights based on the reconstruction loss (Lines 6–7).

Lines 9–45 are the training process for the DRL model. First, the value network is initialized with imitation learning using a set of trajectories generated from the ORCA (Lines 9–10). Second, the target network V' and the replay memory E are initialized (Lines 11–12). Third, the algorithm repeats η_{drl} episodes to train the DRL model. For each episode, we replay the robot's motion μ times according to the target value network and then update the memory E.

Algorithm 1 Training Process for `DAE-Crit-LSTM`

Input: DAE's training episodes η_{dae}, DRL's training episodes η_{drl}, DRL's replay iterations μ, the action space $\mathcal{A}$, the exploration probability ϵ, the maximal time steps T_m, and the update frequency for the target value network ω.
Output: Trained `DAE-Crit-LSTM`.

`// Training DAE`
1: Collect a training dataset D via ORCA;
2: Initialize an encoder model M_{enc} and a decoder model M_{dec};
3: **for** $episode1 = 1 : \eta_{dae}$ **do**
4: Sample z from D;
5: Generate an attack input $\tilde{z} = z + \sigma$;
6: $\hat{z} = M_{dec}(M_{enc}(\tilde{z}, \theta), \varphi)$;
7: Update θ and φ by optimizing (8);
8: **end for**
`// Training DRL`
9: Collect a set of trajectories L via ORCA;
10: Initialize a value network V via imitation learning on L;
11: Initialize the target network $V' \leftarrow V$;
12: Initialize a replay memory E with L;
13: **for** $episode2 = 1 : \eta_{drl}$ **do**
14: **for** $ite = 1 : \mu$ **do**
15: Sample $\mathbf{s}_0$, $\mathbf{S}_0$, and $\underline{\mathbf{p}}$;
16: $t = 0$, $done = False$, $traj = \{(\mathbf{s}_0, \mathbf{S}_0)\}$;
17: **while** *not done* **do**
18: Compute $\hat{\mathbf{S}}_t$ based on M_{enc} and M_{dec};
19: Generate a random value α in $[0, 1]$;
20: **if** $\alpha < \epsilon$ **then**
21: Select an action $\mathbf{v}_t^*$ from $\mathcal{A}$ randomly;
22: **else**
23: **for** each $\mathbf{v}_t \in \mathcal{A}$ **do**
24: Predict the obstacles' next states $\hat{\mathbf{S}}_{t+1}$ according to $\hat{\mathbf{S}}_t$;
25: $\mathbf{Sh}_{t+1} \leftarrow LSTM(criticality(\hat{\mathbf{S}}_{t+1}))$;
26: Compute the value $V(\mathbf{s}_t, \mathbf{S}_t, \mathbf{v})$ according to (11) and the target value network V';
27: **end for**
28: Select the action $\mathbf{v}_t^*$ with the maximal value;
29: **end if**
30: The robot moves to $\mathbf{s}_{t+1}$ controlled by $\mathbf{v}_t^*$;
31: $t \leftarrow t + 1$, $traj = traj \cup \{(\mathbf{s}_t, \mathbf{S}_t)\}$;
32: Detect the new obstacles' states $\mathbf{S}_{t+1}$;
33: **if** *collided* or *reached* or $t > T_m$ **then**
34: $done = True$;
35: **end if**
36: **end while**
37: **if** *collided* or *reached* **then**
38: Update the memory E with $traj$;
39: **end if**
40: **end for**
41: Update V and $LSTM$ by gradient descent with E;
42: **if** $(episode2 \bmod \omega) == 0$ **then**
43: Update target network $V' \leftarrow V$;
44: **end if**
45: **end for**

In each replay (Lines 15–36), we first reset the initial state $[\mathbf{s}_0, \mathbf{S}_0]$ and the target position $\underline{\mathbf{p}}$ randomly, and then control the robot move to $\underline{\mathbf{p}}$ according to (10) and (11) (Lines 17–36). In detail, at each time instant t, the original states of the obstacles $\mathbf{S}_t$ are fed to the trained M_{enc} and M_{dec}, generating the reconstructed states $\hat{\mathbf{S}}_t$ (Line 18). Then the ϵ-greedy policy is used to select $\mathbf{v}_t$ from the action space $\mathcal{A}$. Given a random value from $[0, 1]$ (Line 19), if it is less than the given probability ϵ, an action $\mathbf{v}_t^*$ is selected randomly from the action space $\mathcal{A}$ (Lines 20 and 21); otherwise, the value with respect to each action is computed based on the corresponding prediction of the obstacles' next states and the target value network (Lines 24–26), where $criticality(\hat{\mathbf{S}}_{t+1})$ sorts $\hat{\mathbf{S}}_{t+1}$ based on the risks of the obstacles to the robot [9]. The action with maximal value is selected (Line 28). Note that the prediction of the obstacles' next states can be done by linear motion predictor or other planning methods such as ORCA. All elements in $\hat{\mathbf{S}}_{t+1}$ are stored according to their criticality and transformed into a hidden state $\mathbf{Sh}_{t+1}$ via the Crit-LSTM proposed in [9] (Line 25). Therefore, the robot will move with $\mathbf{v}_t$ in the next duration $[t\Delta t, (t+1)\Delta t)$. When the robot collides with an obstacle, arrives at its target position, or reaches the maximal motion time, the current replay is terminated (Lines 33–35).

Only when the replay is terminated with the *collided* or *reached* status, the generated trajectory is used to update the replay memory E (Lines 37–39). The parameters of value network V and $LSTM$ are updated simultaneously by gradient descent at the end of each episode (Line 41). The target network V' is updated every ω episodes (Lines 42–44).

C. Distributed Execution Process of `DAE-Crit-LSTM`

Once the training process is completed, each robot in a multi-robot system can use the trained DAE and DRL models for attack mitigation and motion generation. Each robot receives the states of the detected obstacles and performs its local robust motion planning (i.e., attack mitigation, motion generation, and motion execution) in a distributed way. Algorithm 2 gives the distributed robust motion planning algorithm for a multi-robot system. Lines 2–13 describe the motion control for each robot. At each time instant t, a robot r^i first detects its own state $\mathbf{s}_t^i$, and then retrieves the obstacles' states $\mathbf{S}_t^i$ (Line 4). For each obstacle in $\mathbf{S}_t^i$, its position is corrected via the well-trained DAE model (Lines 5–7). When all positions are corrected, the new corrected obstacles' states $\hat{\mathbf{S}}_t^i = \{(\mathbf{p}_t^j, \mathbf{v}_t^j, \rho_t^j)\}_{j=1}^{N-1}$ are generated (Line 8). Next, the robot updates its velocity based on the well-trained DRL model generated from Algorithm 1 (Line 9). Note that the process for velocity update is the same with Lines 23–28 in Algorithm 1. Finally, the robot moves forward with the new velocity the next time duration, i.e., $\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}_t \cdot \Delta t$ (Line 10). As time elapses, the robot updates the current time instant and checks whether it arrives at its target position. The motion is terminated if the robot arrives at its target position or reaches the maximal motion time (Line 3).

VII. Experimental Evaluation

In this section, we conduct a set of simulations to validate the attack mitigation and motion planning performance of `DAE-Crit-LSTM` in different scenarios. Specifically, we first evaluate the performance of `DAE-Crit-LSTM` with a variable number of obstacles (Section VII-B). We then demonstrate the mitigation capacity of `DAE-Crit-LSTM` with six PDAs (Section VII-C). In addition, we compare our DAE-based mitigation strategy with other four existing attack recovery methods to highlight their respective capabilities

Algorithm 2 Execution Process for `DAE-Crit-LSTM`

Input: The initial positions $\{\overline{\mathbf{p}}^1, \overline{\mathbf{p}}^2, \ldots, \overline{\mathbf{p}}^N\}$ and target positions $\{\underline{\mathbf{p}}^1, \underline{\mathbf{p}}^2, \ldots, \underline{\mathbf{p}}^N\}$ of all robots, the maximal time steps T_m, the trained DAE model, the trained Crit-LSTM model, the time step Δt.
Output: Collision-free trajectories.

for each robot r_i **do**
  $t = 0$;
  while $\mathbf{p}_t^i \neq \underline{\mathbf{p}}^i$ and $t < T_m$ **do**
    Detect states $\mathbf{s}_t^i$ and $\mathbf{S}_t^i$;
    for $\forall(\mathbf{p}_t^j, \mathbf{v}_t^j, \rho^j) \in \mathbf{S}_t^i$ **do**
      Correct the attacked position $\hat{\mathbf{p}}_t^j \leftarrow DAE(\mathbf{p}_t^j)$;
    end for
    Generate the corrected state $\hat{\mathbf{S}}_t^i$;
    Select $\mathbf{v}_t$ from $\mathcal{A}$ via the DRL model trained in Algorithm 1;
    Robot r^i moves to $\mathbf{s}_{t+1}$ controlled by $\mathbf{v}_t$;
    $t \leftarrow t + 1$
  end while
end for

(Section VII-D). Finally, we evaluate the generalization of `DAE-Crit-LSTM` with other three DRL motion planning methods (Section VII-E).

A. Simulation Setup

We conduct our experiments in the simulation environment developed by [10]. All models are implemented in PyTorch. For training the DAE model, we collect 138,500 samples via ORCA. The network architectures of the encoder and decoder are (6, 5, 4, 3) and (3, 4, 5, 6), respectively. We train the encoder and decoder for 500 episodes (i.e., $\eta_{dae} = 500$), with a batch size of 128. They are trained in an end-to-end way with a learning rate of 0.01. To encourage position correction, λ in (8) is set to 8 empirically. In the training process, to simulate PDAs, we design three types of attacks to generate the faked position $\tilde{\mathbf{p}}_t$ in each $\tilde{z}_t$: (1) $\tilde{\mathbf{p}}_t = [x_t + \delta_1, y_t + \delta_2]$, where δ_1 and δ_2 are generated from the Gaussian distribution $N(0, 1)$, and $[x_t, y_t]$ represents the original position, (2) $\tilde{\mathbf{p}}_t = [2x_t, 2y_t]$, and (3) $\tilde{\mathbf{p}}_t = [0, 0]$. In each training episode, the model randomly selects one of the three ways to generate the attacked inputs.

For training of the Crit-LSTM model, the preferred speed $v_p = 1$, the safe radius $\rho = 0.3$, and the action space $\mathcal{A} = \{(v_i \cos\theta_j, v_i \sin\theta_j) | v_i = (e^{i/5} - 1)/(e - 1), \theta_j = j/8, i = 1, \cdots, 5, j = 0, \cdots, 15\}$. In each replay, each obstacle is randomly located and needs to move to the opposite location with respect to the origin of the coordinates. In addition, $\eta_{drl} = 30,000$, $\mu = 1$, $\omega = 50$, $T_m = 100$, $\gamma = 0.9$. The exploration rate of ϵ-greedy policy decreases linearly from 0.5 to 0.1 in the first $4k$ episodes and stays 0.1 for the remaining episodes. The dimension of the LSTM's hidden state is 50, the values of c_0 and h_0 are zero vectors, and the architecture of the value network is (150, 100, 100, 1). The LSTM model and value network are trained simultaneously with a variable number of obstacles, varying from 1 to 10. During the training phase, the learning rate is 0.001.

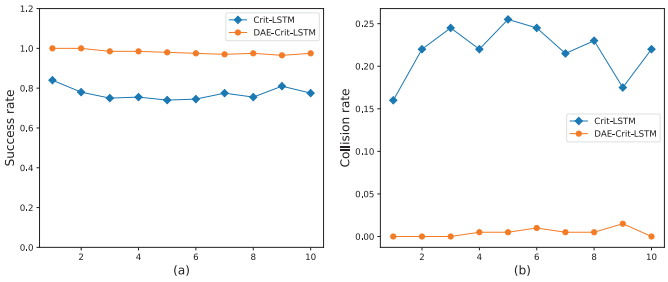


Fig. 6. Success and collision rates of DAE-Crit-LSTM and Crit-LSTM with the number of obstacles varying from 1 to 10.

B. Overall Performance of `DAE-Crit-LSTM`

To evaluate the effectiveness of the `DAE-Crit-LSTM`, we compare it with the corresponding pure Crit-LSTM method. Note that Crit-LSTM is trained with the same parameters in the DRL training process of `DAE-Crit-LSTM`. We first evaluate the success and collision rates of either model under PDAs. We conduct 10 independent test sets for the two methods, where the number of obstacles varies from 1 to 10. Each test set contains 200 test cases. Specifically, for each test case, all obstacles are under attacks, i.e., $\tilde{\mathbf{p}}_t = [x_t + 2\delta_1, y_t + 2\delta_2]$, where δ_1 and δ_2 are generated from the Gaussian distribution $N(0, 1)$, from the second time step to the termination of the motion. Note that this attack is different from the attacks applied in the training phase. Fig. 6 shows the success and collision rates of either method on 10 test sets. The results show `DAE-Crit-LSTM` achieves a higher success rate (98.1% vs. 77.5% on average) and a lower collision rate (0.4% vs. 21.8% on average). It means `DAE-Crit-LSTM` can stably promote Crit-LSTM to mitigate PDAs and guarantee high success rates.

Furthermore, in all test sets, we evaluate either model under two scenarios: with PDAs and without PDAs. The metrics to evaluate the experiments are *success* rate, *collision* rate, *timeout* rate, *average navigation time of the successful runs (NavTime)*, and *average decision time of the successful runs (DecTime)*. Table I shows their means and standard deviations (SDs) over 10 test sets. From Table I, we have the following conclusions:

- **The DAE model does not affect the performance of the DRL model under benign scenarios**. The table shows that the two models show almost the same performance on the five metrics in the attack-free scenario. This means that the DAE-based mitigation strategy can reserve the benign states of the obstacles.
- **`DAE-Crit-LSTM` can mitigate PDAs significantly**. In the case of PDAs, `DAE-Crit-LSTM` can obtain a 98.1% success rate, while the success rate of Crit-LSTM is only 77.5%. Crit-LSTM causes a higher average collision rate (21.8% vs. 0.4%), which means the robot falls into collisions at many runs under PDAs.
- **`DAE-Crit-LSTM` can generate smooth motion**. Under attacks, `DAE-Crit-LSTM` has a lower

TABLE I
PERFORMANCE COMPARISON BETWEEN DAE-CRIT-LSTM AND CRIT-LSTM IN THE ATTACK-FREE AND THE ATTACKED SCENARIOS

Attack	Method	Success (%)		Collision (%)		Timeout (%)		NavTime (s)		DecTime (s)	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
N	Crit-LSTM	**0.981**	0.012	0.005	0.006	0.014	0.008	11.412	1.536	0.076	0.016
	DAE-Crit-LSTM	0.980	0.013	**0.004**	0.003	0.016	0.010	11.249	1.497	0.079	0.016
Y	Crit-LSTM	0.775	0.031	0.218	0.030	0.007	0.005	11.290	1.513	0.076	0.016
	DAE-Crit-LSTM	**0.981**	0.012	**0.004**	0.005	0.015	0.009	11.252	1.512	0.078	0.016

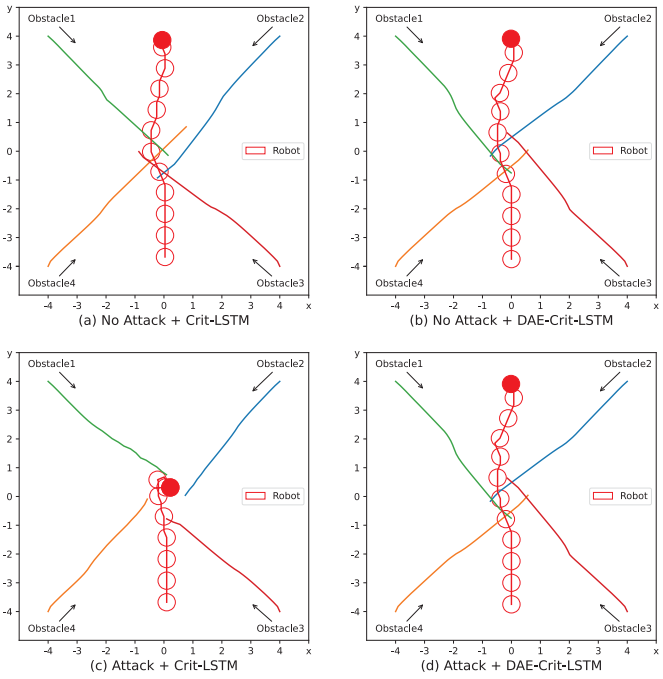


Fig. 7. The trajectories of the robot and four obstacles. The target position of the robot is $[0, 4]$. Circles are the locations of the robot. Filled circles indicate the terminal positions of the robot's motion. The solid line without circles represents the trajectories of the obstacles. Their target positions are $[4, -4]$, $[-4, -4]$, $[-4, 4]$, and $[4, 4]$, respectively. All obstacles are controlled by ORCA.

average navigation time (11.290s for Crit-LSTM and 11.252s for DAE-Crit-LSTM). It means that DAE-Crit-LSTM can navigate a robot moving to its target position with a shorter and smoother motion trajectory.

- **DAE-Crit-LSTM does not affect the real-time performance**. From the table, we can find that compared with Crit-LSTM, DAE-Crit-LSTM has a similar average decision time (DecTime) in either scenario. The computational overhead introduced by the DAE model increases by less than 3.9% (i.e., $(0.079-0.076)/0.076 = 0.039$ and $(0.078 - 0.076)/0.076 = 0.013$).

Hence, we can conclude that DAE-Crit-LSTM can mitigate PDAs efficiently without causing significant performance degradation. Moreover, as DAE-Crit-LSTM is suitable for benign and attacked scenarios, no attack detectors are required.

In this sequel, we further show some trajectories generated by the two methods with and without PDAs, as shown in Fig. 7. From Fig. 7(a), we can observe that in the benign scenario, Crit-LSTM can control the robot to move towards its target smoothly. As shown in Fig. 7(b), DAE-Crit-LSTM can also generate a smooth trajectory for the robot, which shows that the DAE-based mitigation strategy can reserve benign states. Fig. 7(c) shows that under the PDA, the robot collides with *obstacles*1 (green line) in the 24-*th* time interval and fails to reach its target, as Crit-LSTM generates a wrong action according to the faked obstacles' positions. From Fig. 7(d), we can see that using DAE to correct attacked positions can navigate the robot to its target while maintaining smooth motion. The comparison between Figs. 7(c) and 7(d) illustrates that DAE-Crit-LSTM can correct the attacked states and navigate the robot towards its target safely and smoothly. The results in Figs. 7(b) and 7(d) illustrate that DAE-Crit-LSTM can deal with both benign and attacked scenarios using a unified DAE-based mitigation strategy without any attack detector.

C. Performance of DAE-Crit-LSTM Under Different Attacks

In this section, we evaluate the mitigation capability of DAE-Crit-LSTM against different attacks. Inspired by [50], [51], and [21], we test the trained DAE-Crit-LSTM using six types of PDAs. (1) *Biased Attack*, where positions are added with a Gaussian distribution $N(0, 1)$. In this attack, an adversary can tamper with the corresponding sensor data to generate the faked positions near the real data. Since this attack can generate close positions, it is an efficient way to deceive existing detectors. When the attacker chooses this attack, an obstacle robot is manipulated to approach or move away from the ego robot, thus affecting the actions of the ego robot. (2) *Percentage Attack*, where the positions are multiplied by 1.25. This attack can be viewed as the attacker disrupting the sensor signals by inducing resonance in the mechanical structure of the internal sensors. During the implementation of this attack, the attacker usually cannot directly access the sensor readings. Instead, the attacker uses a crafted acoustic wave or electromagnetic interference to resonate the sensor. (3) *Zero Replacement Attack*, where all positions are replaced by a zero vector. In this way, the adversary sends a virtual position, $[0, 0]$, to the robot, while concealing the actual data. This approach is chosen when the attacker aims to create a virtual static obstacle in the environment. (4) *Delay Attack*, where the previous position $\mathbf{p}_{t-1}$ is sent to the robot at current time instant t, i.e., $\tilde{\mathbf{p}}_t = \mathbf{p}_{t-1} = [x_{t-1}, y_{t-1}]$. This attack is designed to simulate an adversary deliberately introducing delays into the transmitted signals. The attacker can conduct this attack during the transmission process of sensor signals. (5) *Rotation Attack*, where each position vector is rotated to $\tilde{\mathbf{p}}_t = [y_t, x_t]$. In this attack, an adversary rotates the position data, resulting in the incorrect position and relative orientation

TABLE II

SUCCESS RATES AND SIGNIFICANCE TESTING OF DAE-CRIT-LSTM AND CRIT-LSTM UNDER SIX PDAS WITH DIFFERENT ATTACK STRENGTHS

Attack Type	Method	Mean	SD	t-Test	Strength
Biased	Crit-LSTM	0.789	0.035	1.530E-15	1.457
	DAE-Crit-LSTM	**0.967**	**0.025**		
Percentage	Crit-LSTM	0.922	0.030	1.760E-04	2.832
	DAE-Crit-LSTM	**0.966**	**0.025**		
Zero Replacement	Crit-LSTM	0.787	0.043	9.003E-14	2.263
	DAE-Crit-LSTM	**0.968**	**0.029**		
Delay	Crit-LSTM	0.873	0.073	1.050E-04	0.162
	DAE-Crit-LSTM	**0.965**	**0.029**		
Rotation	Crit-LSTM	0.189	0.243	5.218E-09	3.629
	DAE-Crit-LSTM	**0.965**	**0.027**		
Physical Removal	Crit-LSTM	0.702	0.098	4.730E-04	0.520
	DAE-Crit-LSTM	**0.841**	**0.094**		

information being sent to the robot. When Byzantine or communication attacks are feasible, attackers will choose this type of attack. (6) *Physical Removal Attack*, where the position of one robot is replaced by the position of another robot, i.e., $\tilde{\mathbf{p}}_t^j = \mathbf{p}_t^k = [x_t^k, y_t^k]$, $j \neq k$. The intention of such an attack is to remove specific regions in equipped sensors, e.g., LiDAR, to hide the presence of genuine obstacles in the robot's route. This attack can be applied in the scenario when the attacker can manipulate the laser pulses. Note that the first three types of attacks are the same as those used in the training phase, while the last three are previously unseen attacks.

We conduct 15 independent test sets for each attack. Each test set contains 200 test cases with a variable number of obstacles, from 1 to 15. For comparison, we also perform all test cases using Crit-LSTM. The average success rate and the SD of either method are given in Table II. The comparison results show that `DAE-Crit-LSTM` can achieve a higher success rate and a lower SD under each attack. To further evaluate the statistical significance of the results, we conduct t-test for equal means [52]. The result is statistically significant, which means there are significant differences in the success rate between `DAE-Crit-LSTM` and Crit-LSTM. Hence, we can conclude that **`DAE-Crit-LSTM` can mitigate different PDAs efficiently and guarantee a higher success rate**.

In addition, we also compute the attack strength to quantify the differences between PDAs. It is defined as the mean distance between the actual position $\mathbf{p}_t$ and the attacked one $\tilde{\mathbf{p}}_t$. The results are given in the last column of Table II. The results show the attack strength employed during the training phase is within [1, 3]. We can find that testing `DAE-Crit-LSTM` with strengths out of this range can still achieve good attack mitigation performance. The result indicates that `DAE-Crit-LSTM` can work well in previously unknown PDAs. It implies that **`DAE-Crit-LSTM` has the ability to deal with PDAs with various attack strengths.**

D. Comparison With Other Defense Strategies

We further compare our DAE-based mitigation strategy with other strategies in the literature. We select four recovery strategies: Kalman filter [53], SSR [29], PID-Piper [30], and Delorean [20]. Specifically, the Kalman filter combines the sensor signals and the predicted positions to generate more accurate state estimations. SSR aims to learn surrogate models (i.e., discrete-time state-space models) to replace the attacked physical sensors and generate the predicted data from the surrogate models for the motion controller. PID-Piper applies an LSTM model to predict the attacked sensor' data for the motion controller. Delorean corrects the attacked signals by replaying a sequence of historic states collected from an attack-free phase. In our experiments, we implement SSR for the attacked position sensor using Matlab's System Identification Toolbox. We also apply the generated state-space model of SSR as the dynamics in the prediction phase of the Kalman filter. The size of the hidden state in the LSTM model of PID-Piper is 5, and the threshold of SSR equals 1. The size of the sliding window for Delorean is 10.

TABLE III

PERFORMANCE COMPARISON AMONG DIFFERENT RECOVERY STRATEGIES UNDER BIASED ATTACK

Strategy	Success	Collision	Timeout	Correction Error
Kalam Filter	0.879	0.108	0.013	0.417
SSR	0.882	0.022	0.096	0.413
PID-Piper	0.807	0.180	0.012	0.141
Delorean	0.849	0.136	0.015	0.533
DAE	**0.971**	**0.015**	0.015	**0.057**

For fair comparisons, we integrate these strategies with the Crit-LSTM model. Similarly, we conduct 10 independent test sets for each strategy. Each test set contains 200 test cases, where the number of obstacles varies from 1 to 10. Since Delorean assumes that the attack signals only exist in an attack zone, we add the biased attack during the time interval [10, 20] in each test case. In particular, in order to simulate the attack detection module in Delorean, we only attack one obstacle that is visible to Delorean. We first compute the average success/collision/timeout rates to evaluate the attack mitigation capability of five strategies. We also compute the correction error (i.e., the mean error between the actual position $\mathbf{p}_t$ and the corrected position $\hat{\mathbf{p}}_t$) to evaluate the position correction capability. The results are given in Table III. The results show that the DAE-based mitigation strategy outperforms the other four methods, achieving the highest accuracy rate of 97.1% and the lowest correction error of 0.057. The reason is that by precisely correcting the attacked positions via the well-trained DAE model, the motion planning model can generate better motion commands. Therefore, **the proposed mitigation strategy can perform more accurate position corrections with a well-trained DAE model**.

E. Generalization of `DAE-Crit-LSTM`

Finally, we evaluate the generalization of the DAE-based mitigation strategy, i.e., its compatibility with other DRL motion planning methods. We select three well-established DRL methods: CADRL [11], SARL [10], and LSTM-RL [37]. In detail, CADRL processes each obstacle information one by one via a trained value network and then selects the action that has the highest value. SARL applies a self-attention mechanism to deal with the variable length of obstacles' states. LSTM-RL utilizes an LSTM to transform a variable

TABLE IV
PERFORMANCE OF THE DAE-BASED MITIGATION STRATEGY WITH DIFFERENT DRL MOTION PLANNING METHODS

Attack	Method	Success	Collision	Timeout
N	CADRL	0.975	0.021	0.004
	DAE-CADRL	0.960	0.035	0.005
Y	CADRL	0.748	0.252	0.000
	DAE-CADRL	**0.962**	**0.031**	0.007
N	SARL	0.976	0.020	0.003
	DAE-SARL	0.910	0.089	0.002
Y	SARL	0.700	0.298	0.003
	DAE-SARL	**0.906**	**0.093**	0.002
N	LSTM-RL	0.565	0.002	0.433
	DAE-LSTM-RL	0.551	0.017	0.432
Y	LSTM-RL	0.477	0.142	0.380
	DAE-LSTM-RL	**0.546**	**0.020**	0.434

number of obstacles into a fixed-sized hidden state based on their distances from the robot. The three methods are also trained for 30,000 episodes. Hence, we have six models: CADRL, DAE-CADRL, SARL, DAE-SARL, LSTM-RL, and DAE-LSTM-RL. Experimental settings are the same as those given in Section VII-B. Table IV shows the average success/collision/timeout rates over 10 test sets.

First, like `DAE-Crit-LSTM` , the three DAE-based models show similar performance with their corresponding DAE-free models under attack-free scenarios. Second, under the attacked scenarios, the three DAE-free models show a significant decrease in the success rate and an increase in the collision rate. In contrast, the DAE-based models maintain a high success rate and low collision and timeout rates. Third, compared with CADRL and SARL, LSTM-RL shows a lower success rate and a higher collision (timeout) rate. The reason is that LSTM-RL only considers the obstacles' distances from the robot, which cannot affect the real influence of the obstacles on the robot's decision-making. In addition, compared with the three methods, Crit-LSTM achieves a higher performance (i.e., the success rates for the attack-free and the attacked scenarios are 98% and 98.1%, respectively). The reason is that Crit-LSTM deals with obstacles' states according to their criticality, which can capture more useful high-level semantic information.

In conclusion, **the proposed DAE-based mitigation strategy against PDAs is universal and can be applied with different DRL motion planning methods**.

VIII. CONCLUSION

In this paper, we propose the first DAE-based mitigation strategy for position deception attacks in multi-robot systems. By training a DAE model, we apply the trained encoder and decoder to approximate the actual positions of the attacked obstacles while reserving the benign ones. So no attack detectors are required for our method. Then, we propose a robust motion planning framework to improve the robustness of motion planning methods in adversarial environments. We also instantiate a method `DAE-Crit-LSTM`, to address a variable number of obstacles in an environment with potential position deception attacks. The comprehensive simulation experiments demonstrate the effectiveness and generalization of the proposed approach.

In the future, there are several directions we can continue. First, we can apply the DAE-based mitigation strategy to the conventional motion planning methods and compare their performance and scope of application. Second, this paper only considers position deception attacks. Thus, we may also explore using the proposed strategy to deal with other attacks or failures in multi-robot systems, such as velocity deception attacks. Third, we also plan to implement and test our approach on real-world platforms to study the effects on different parameters.

REFERENCES

[1] Z. Zhou, J. Liu, and J. Yu, "A survey of underwater multi-robot systems," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 1, pp. 1–18, Jan. 2022.

[2] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A distributed method to avoid higher-order deadlocks in multi-robot systems," *Automatica*, vol. 112, Feb. 2020, Art. no. 108706.

[3] J. Wu, C. Song, J. Ma, J. Wu, and G. Han, "Reinforcement learning and particle swarm optimization supporting real-time rescue assignments for multiple autonomous underwater vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6807–6820, Jul. 2022.

[4] R. S. de Moraes and E. P. de Freitas, "Multi-UAV based crowd monitoring system," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 56, no. 2, pp. 1332–1345, Apr. 2020.

[5] M. S. Catherine and E. Lucet, "A modified hybrid reciprocal velocity obstacles approach for multi-robot motion planning without communication," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 5708–5714.

[6] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*. Cham, Switzerland: Springer, 2011, pp. 3–19.

[7] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 49, no. 12, pp. 2636–2650, Dec. 2019.

[8] Y. Ouyang, B. Li, Y. Zhang, T. Acarman, Y. Guo, and T. Zhang, "Fast and optimal trajectory planning for multiple vehicles in a nonconvex and cluttered environment: Benchmarks, methodology, and experiments," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 10746–10752.

[9] L. Xu, F. Wu, Y. Zhou, H. Hu, Z. Ding, and Y. Liu, "Criticality-guided deep reinforcement learning for motion planning," in *Proc. China Autom. Congr. (CAC)*, Oct. 2021, pp. 3378–3383.

[10] 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.

[11] 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.

[12] E. Marchesini and A. Farinelli, "Enhancing deep reinforcement learning approaches for multi-robot navigation via single-robot evolutionary policy search," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 5525–5531.

[13] G. Deng, Y. Zhou, Y. Xu, T. Zhang, and Y. Liu, "An investigation of Byzantine threats in multi-robot systems," in *Proc. 24th Int. Symp. Res. Attacks, Intrusions Defenses*, Oct. 2021, pp. p17–32.

[14] M. Delcourt and J.-Y. L. Boudec, "TDOA source-localization technique robust to time-synchronization attacks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4249–4264, 2021.

[15] Z. Sun, S. Balakrishnan, L. Su, A. Bhuyan, P. Wang, and C. Qiao, "Who is in control? Practical physical layer attack and defense for mmWave-based sensing in autonomous vehicles," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3199–3214, 2021.

[16] Z. Hong, X. Li, Z. Wen, L. Zhou, H. Chen, and J. Su, "ESP spoofing: Covert acoustic attack on MEMS gyroscopes in vehicles," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 3734–3747, 2022.

[17] B. Pardhasaradhi and L. R. Cenkeramaddi, "GPS spoofing detection and mitigation for drones using distributed radar tracking and fusion," *IEEE Sensors J.*, vol. 22, no. 11, pp. 11122–11134, Jun. 2022.

[18] S. Dasgupta, M. Rahman, M. Islam, and M. Chowdhury, "A sensor fusion-based GNSS spoofing attack detection framework for autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 23559–23572, Dec. 2022.

[19] J. Liu and J.-M. Park, "'Seeing is not always believing': Detecting perception error attacks against autonomous vehicles," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2209–2223, May 2021.

[20] P. Dash, G. Li, M. Karimibiuki, and K. Pattabiraman, "Replay based for recovering autonomous robotic vehicles from sensor deception attacks," 2022, *arXiv:2209.04554*.

[21] Y. Cao, S. H. Bhupathiraju, P. Naghavi, T. Sugawara, Z. M. Mao, and S. Rampazzi, "You can't see me: Physical removal attacks on LiDAR-based autonomous vehicles driving frameworks," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 2993–3010.

[22] W. Fu, J. Qin, Y. Shi, W. X. Zheng, and Y. Kang, "Resilient consensus of discrete-time complex cyber-physical networks under deception attacks," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4868–4877, Jul. 2020.

[23] J. Jang, M. Cho, J. Kim, D. Kim, and Y. Kim, "Paralyzing drones via EMI signal injection on sensory communication channels," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.

[24] W. Wang, Z. Han, K. Liu, and J. Lü, "Distributed adaptive resilient formation control of uncertain nonholonomic mobile robots under deception attacks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 9, pp. 3822–3835, Sep. 2021.

[25] Y. Cao et al., "Adversarial sensor attack on LiDAR-based perception in autonomous driving," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2267–2281.

[26] H. Choi et al., "Detecting attacks against robotic vehicles: A control invariant approach," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 801–816.

[27] Q. Liu, Y. Mo, X. Mo, C. Lv, E. Mihankhah, and D. Wang, "Secure pose estimation for autonomous vehicles under cyber attacks," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 1583–1588.

[28] F. van Wyk, Y. Wang, A. Khojandi, and N. Masoud, "Real-time sensor anomaly detection and identification in automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 1264–1276, Mar. 2020.

[29] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2020, pp. 349–364.

[30] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "PID-piper: Recovering robotic vehicles from physical attacks," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 26–38.

[31] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 1–38, 2010.

[32] X. Liu et al., "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, Jan. 2023.

[33] J. A. Douthwaite, S. Zhao, and L. S. Mihaylova, "Velocity obstacle approaches for multi-agent collision avoidance," *Unmanned Syst.*, vol. 7, no. 1, pp. 55–64, Jan. 2019.

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

[35] K. Wu, H. Wang, M. A. Esfahani, and S. Yuan, "Learn to navigate autonomously through deep reinforcement learning," *IEEE Trans. Ind. Electron.*, vol. 69, no. 5, pp. 5342–5352, May 2022.

[36] S. H. Semnani, H. Liu, M. Everett, A. de Ruiter, and J. P. How, "Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 3221–3226, Apr. 2020.

[37] 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.

[38] Z. Wang and R. S. Blum, "Algorithms and analysis for optimizing the tracking performance of cyber attacked sensor-equipped connected vehicle networks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 5061–5076, 2021.

[39] Y. Han, H. Zhang, H. Li, Y. Jin, C. Lang, and Y. Li, "Collaborative perception in autonomous driving: Methods, datasets and challenges," 2023, *arXiv:2301.06262*.

[40] L. Zhou, V. Tzoumas, G. J. Pappas, and P. Tokekar, "Distributed attack-robust submodular maximization for multirobot planning," *IEEE Trans. Robot.*, vol. 38, no. 5, pp. 3097–3112, Oct. 2022.

[41] L. Zhou and V. Kumar, "Robust multi-robot active target tracking against sensing and communication attacks," in *Proc. Amer. Control Conf. (ACC)*, Jun. 2022, pp. 4443–4450.

[42] W. Tang et al., "GAN-based robust motion planning for mobile robots against localization attacks," *IEEE Robot. Autom. Lett.*, vol. 8, no. 3, pp. 1603–1610, Mar. 2023.

[43] Y. Yang, Y. Xiao, and T. Li, "Attacks on formation control for multiagent systems," *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 12805–12817, Dec. 2022.

[44] Y. Han et al., "Deep reinforcement learning for robot collision avoidance with self-state-attention and sensor fusion," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 6886–6893, Jul. 2022.

[45] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *Proc. Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 241–246.

[46] H.-P. Liu, Y. Tsao, and C.-S. Fuh, "Bone-conducted speech enhancement using deep denoising autoencoder," *Speech Commun.*, vol. 104, pp. 106–112, Nov. 2018.

[47] Q. Ma, W.-C. Lee, T.-Y. Fu, Y. Gu, and G. Yu, "MIDIA: Exploring denoising autoencoders for missing data imputation," *Data Mining Knowl. Discovery*, vol. 34, no. 6, pp. 1859–1897, Nov. 2020.

[48] P. Singh and A. Sharma, "Attention-based convolutional denoising autoencoder for two-lead ECG denoising and arrhythmia classification," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–10, 2022.

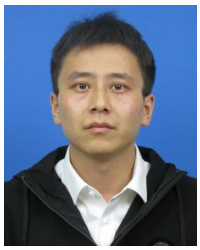
[49] J. Liu et al., "Toward robust fault identification of complex industrial processes using stacked sparse-denoising autoencoder with softmax classifier," *IEEE Trans. Cybern.*, vol. 53, no. 1, pp. 428–442, Jan. 2023.

[50] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting UAVs with reinforcement learning-assisted flight control under cyber-physical attacks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 7358–7364.

[51] S. Rivera, S. Lagraa, A. K. Iannillo, and R. State, "Auto-encoding robot state against sensor spoofing attacks," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2019, pp. 252–257.

[52] N. A. Heckert and J. J. Filliben, *NIST/SEMATECH e-Handbook of Statistical Methods; Chapter 1: Exploratory Data Analysis*. Gaithersburg, MD, USA, 2003.

[53] C. Suliman, C. Cruceru, and F. Moldoveanu, "Mobile robot position estimation using the Kalman filter," *Acta Marisiensis. Seria Technologica*, vol. 6, p. 75, Jan. 2009.



Wenbing Tang received the B.E. degree in software engineering from the Nanjing University of Posts and Telecommunications, Nanjing, China, in June 2017, and the M.E. degree in software engineering from Zhejiang Sci-Tech University, Hangzhou, China, in April 2020. He is currently pursuing the Ph.D. degree with the Software Engineering Institute, East China Normal University, Shanghai, China. His research interests include causal inference, motion planning, and reinforcement learning.

Yuan Zhou (Member, IEEE) received the M.S. degree in computational mathematics from Zhejiang Sci-Tech University, Hangzhou, China, in March 2015, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in June 2019. He is currently a Research Fellow with the School of Computer Science and Engineering, Nanyang Technological University. He has published about 40 research papers in various journals and conferences, including IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, IEEE TRANSACTIONS ON FUZZY SYSTEMS, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON RELIABILITY, the IEEE Symposium on Security and Privacy (S&P), the International Conference on Robotics and Automation (ICRA), the International Symposium on Software Testing and Analysis (ISSTA), and the International Conference on Software Engineering (ICSE). His research interests include the safety and security of autonomous unmanned systems, including multi-robot systems and autonomous vehicles.

Yang Liu (Senior Member, IEEE) received the B.Comp. degree (Hons.) from the National University of Singapore (NUS) in 2005 and the dual Ph.D. degree from NUS and MIT in 2010. He was a Post-Doctoral Researcher with NUS and MIT. In 2012, he joined Nanyang Technological University (NTU). He is currently a Full Professor and the Director of the Cybersecurity Laboratory, NTU. He specializes in software verification, security, and software engineering. His research has bridged the gap between the theory and practical usage of formal methods and program analysis to evaluate the design and implementation of software for high assurance and security. By now, he has more than 270 publications in top-tier conferences and journals. He received a number of prestigious awards, including the MSRA Fellowship, the TRF Fellowship, the Nanyang Assistant Professor, the Tan Chin Tuan Fellowship, the Nanyang Research Award, and eight best paper awards in top conferences, such as ASE, FSE, and ICSE.

Zuohua Ding (Member, IEEE) received the M.S. degree in computer science and the Ph.D. degree in mathematics from the University of South Florida, Tampa, FL, USA, in 1996 and 1998, respectively. He is currently a Professor and the Director of the Laboratory of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China. He has authored or coauthored more than 120 articles. His current research interests include system modeling, software reliability prediction, intelligent software systems, and service robots.

Jing Liu (Member, IEEE) is currently a Professor of computer science with East China Normal University, China. In recent years, she has been involved in the area of model-driven architecture. Her research interests include the design of real-time embedded systems and cyber-physical systems.