



Anomalous acting agents: the deployment problem

Ingeborg Wenger¹ · Henrik Ebel² · Peter Eberhard¹

Received: 31 October 2023 / Accepted: 7 May 2024 / Published online: 6 June 2024
© The Author(s) 2024

Abstract

Detecting intentionally antagonistic behavior in robot swarms brings challenges that exceed identifying merely erroneous behavior. We investigate a data-based approach to recognize anomalous and, in particular, antagonistic behavior in robots executing a deployment task. The task requires a robot swarm of variable size and starting positions to be optimally distributed within an arbitrary convex surveillance area. Combining a long short-term memory neural network and a normalizing flow, our approach learns to approximate the probability of a robot action. Thus, actions with low probability density values can be categorized as anomalous. The applicability of the proposed approach is validated on simulated runs containing benevolent, antagonistic, and erroneous robots. Both antagonistic and erroneous robots are detected with more than 90 percent accuracy.

Keywords Machine learning · Robot swarm · Anomaly detection · Deployment problem

1 Introduction

Robots have brought sustained progress to industry, transportation, logistics, and to the service and security sectors. They have become more accurate, more versatile, and more cost-effective over the years, with much of the progress coming from improvements in the designs and algorithms of individual robots. Future progress may be achieved by letting a swarm of communicating robots cooperate. On the one hand, employing a robot swarm may be more flexible and robust since it can dynamically adapt to the amount of work to be done, and it may be able to reorganize if individual robots break down [1]. In addition, the modular combination of robots allows for the parallel execution of sub-tasks by simpler robots whose individual capabilities may be limited to certain tasks [2]. On the other hand, some

✉ P. Eberhard
peter.eberhard@itm.uni-stuttgart.de

I. Wenger
ingeborg.wenger@itm.uni-stuttgart.de

H. Ebel
henrik.ebel@lut.fi

¹ Institute of Engineering and Computational Mechanics, University of Stuttgart, Pfaffenwaldring 9, 70569 Stuttgart, Germany

² Department of Mechanical Engineering, LUT University, Yliopistonkatu 34, 53850 Lappeenranta, Finland

tasks may only be achievable when multiple robots cooperate. Examples of the latter include coverage tasks where a group of robots shall cover a large environment that cannot be covered by a single robot alone, e.g., to monitor an area with sensors to direct relief efforts in a disaster scenario, to track the shape of contaminant clouds, or to provide signal coverage for radio communication [3, 4]. Sometimes, a more dense coverage is desirable in certain areas. Mathematically, all of these practical problems can be subsumed in the same abstract framework of a general deployment or coverage problem, which is the setting of this paper. Coverage tasks have been studied for a variety of circumstances, see, e.g., [5–7].

As is often the case with networked robotics, the proposed approaches for coverage tasks generally assume that all involved robots are cooperative, whereas uncooperative behavior is studied more seldom. In general, potential attacks on robot swarms have been identified in [3, 8], with [8] defining an attack on a robot swarm as altering the swarm's behavior with the goal of reducing the efficiency of the swarm or preventing it from accomplishing its task. Applying this definition to the coverage problem, antagonistic behavior may, for instance, consist of a robot physically moving in an altered manner to gain coverage of a particularly sensitive area, thereby preventing that area from being covered by one of the benevolent robots. This goes beyond purely erroneous behavior that is not directed at a specific antagonistic goal. Whereas methodological inspiration can be drawn from approaches dealing with faulty, or generally anomalous, behavior [9], antagonistic behavior does raise additional research questions. For instance, overt antagonistic behavior may be more obstructive if left unaddressed, whereas more stealthy antagonistic behavior may be harder to detect but less effective at disrupting the swarm's task. In this context, the present work aims to develop and analyze an approach that can identify anomalous behavior in deployment tasks. In particular, we focus on the physical manifestation of erroneous or antagonistic behavior that affects the swarm by influencing mechanical-dynamical quantities such as robot motion. Other antagonistic influences, like the manipulation of communication, are not the subject of this study.

Indeed, identifying the presence and identity of deliberately antagonistic agents in a robot swarm has received little attention [10], and the authors are not aware of any work that studies antagonistic behavior in the deployment-problem setting. Several papers investigate a robot swarm's ability to reach consensus in a collective decision problem despite the presence of misbehaving agents [11–14]. In [15] and [16], the detection of anomalies is investigated in network traffic and physical properties such as power consumption, speed, or residual energy. The research in [10] assumes prior knowledge of signatures of abnormal swarm behavior, which can then be used to check against the observed swarm behavior. Further, by examining the effect of a randomized secret control signal on the robot's behavior, [17] proposes to verify that a robot is not controlled by an intruder mimicking normal swarm behavior. However, this approach presupposes the existence of a central control unit that is usually not available in decentralized robot swarms. Thus, beyond the setting, the novelty of this work lies in proposing a data-driven approach for detecting antagonistic behavior, requiring only data on normal, benevolent behavior, no central control unit, and no a priori expert knowledge.

Methodologically, this work combines a neural network architecture called normalizing flow with a feature embedding based on recurrent neural networks. The proposed scheme is validated on data that was generated using the multibody model of an omnidirectional mobile robot.

The paper is organized as follows: Sect. 2 describes the deployment task and the different types of anomalous robots considered in this work. Section 3 provides information regarding the optimal solution to the deployment task as well as the neural network architecture used

in this work. Section 4 describes the detection approach, including additional information regarding its implementation. In Sects. 5 and 6, the results are visualized and discussed, and an outlook to future work is provided.

2 Problem setting

Surveillance or monitoring tasks require the robot swarm to deploy within a fixed or dynamic area. In this work, the coverage area is represented as a single convex polygon, and it is the task of the robot swarm to distribute uniformly within this area. While it generally cannot be assumed that the optimal actions of the robot agents in a swarm are known, we facilitate the examination of the proposed anomaly detection method by computing the benevolent behavior of the robot swarm deterministically and using the computed actions as a reference. The deterministic behavior of a robot swarm that cooperatively solves the deployment problem is characterized as follows. Each robot is informed about the vertices of the area's boundary and knows its current position relative to the vertices. Additionally, in order to avoid collisions and ensure proper coverage, each robot communicates its position at regular time steps t_j , $j \in \mathbb{N}_0$, to the other agents in the robot swarm. The coverage is optimized iteratively using Lloyd's Algorithm [18], see Algorithm 1. At each time step t_j , a robot leverages the position information to compute its Voronoi cell, where the Voronoi cell of the i th robot represents all points within the coverage area that are closer to the robot than to any other robot and are therefore under the surveillance of robot i . Figure 1 visualizes the Voronoi decomposition for the respective time step as black, dashed lines. In order to improve the coverage of its cell area, the robot moves towards the area center of its Voronoi cell until the next time step. It then reports its new position. If the vertices of the coverage area are fixed, the robots will converge to their optimal position after a finite number of time steps. In this work, we employ a swarm of omnidirectional mobile robots, as depicted in Fig. 2. Each robot has four Mecanum wheels [19] arranged in a rotation-symmetric fashion. This setup allows the robots to accelerate in any direction independent of their current configuration and, neglecting inertia, to always move in straight lines towards the cell center. Since Voronoi cells are convex, the robots do not leave their cell when moving straight towards the center of their Voronoi cell.

Deviating from the normal behavior, where all robots cooperate according to Lloyd's algorithm, in this work, three types of anomalous behavior are investigated. First, in the event of a hardware failure, a robot may become immobile but may still communicate its current

Algorithm 1 Application of Lloyd's Algorithm [18] to the Deployment Problem

```

 $j \leftarrow 0$ ,  $n_r \leftarrow$  number of robots,  $n_v \leftarrow$  number of polygon vertices
while  $t_j < t_{\text{end}}$  do
   $V(t_j) \in \mathbb{R}^{n_v \times 2} \leftarrow$  vertices of coverage area at time step  $t_j$ 
   $X(t_j) \in \mathbb{R}^{n_r \times 2} \leftarrow$  current robot positions at time step  $t_j$ 
  for  $i$  in  $\{1, \dots, n_r\}$  do
    compute the Voronoi cell  $W_i(V, X)$  of robot  $i$ 
    compute the (weighted) area center  $c_i$  of Voronoi cell  $W_i$ 
    move robot  $i$  towards  $c_i$ 
  end for
   $t_{j+1} \leftarrow t_j + \Delta t$ 
end while

```

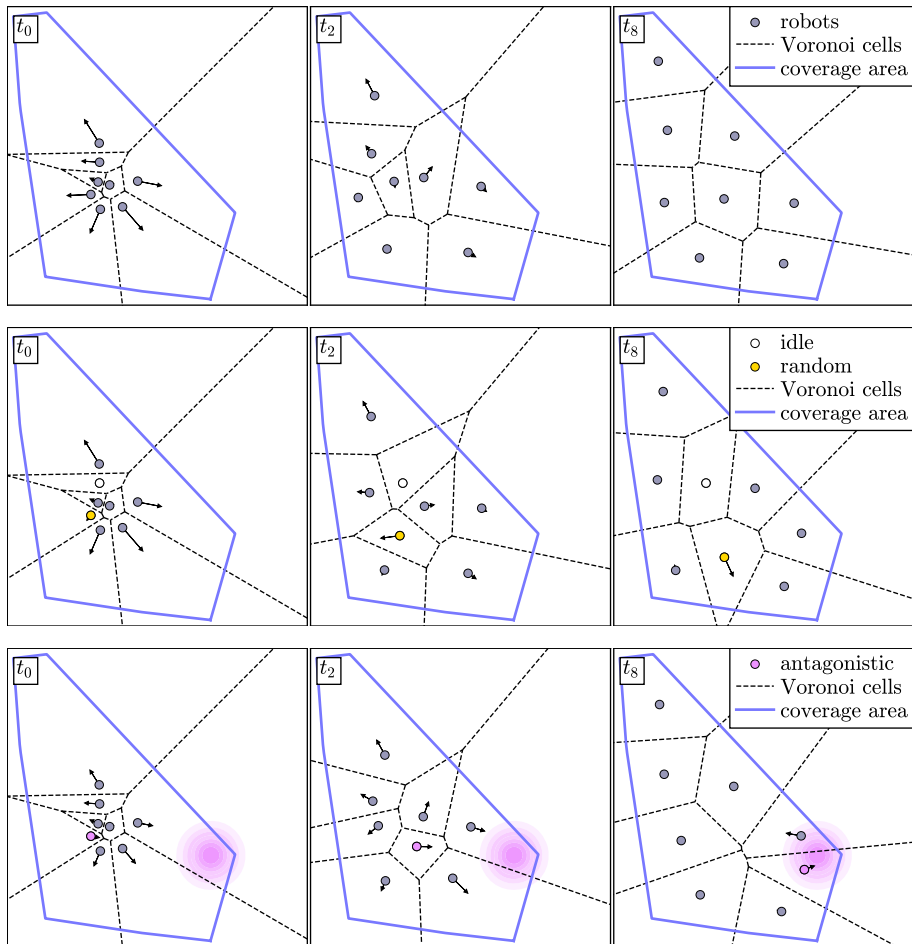


Fig. 1 Development of the robot deployment for time steps t_0 , t_2 and t_8 using a swarm that includes only normal robots (*top*), one random (yellow) and one idle (white) robot (*middle*) and an antagonistic (pink) robot that moves towards the pink target area (*bottom*). Color figure online

position. The idle robot will thus induce a sub-optimal coverage and monitoring of the area. Second, software or hardware errors resulting in the improper propulsion of the robot wheels or inaccurate tracking of the robot position can cause a robot to move erroneously. The effect of such actor or sensor problems is exemplified in the second scenario, with a robot performing random actions. Such random motions will prevent the swarm from converging to its optimal positions and increase energy consumption. The effect of erroneous robots on task completion is also visualized in the second row of Fig. 1. Thirdly, with the goal of keeping the robot swarm from monitoring a certain area of interest within the coverage area, a robot may actively behave antagonistically, e.g., due to being hacked. The behavior of such an antagonistic robot is visualized in the last row of Fig. 1, with the pink density representing the antagonistic robot's area of interest. By intentionally moving to its area of interest, the antagonistic robot will establish control over that area since the robot swarm's ambition to optimize the coverage of the entire area will cause the benevolent robots to dis-

Fig. 2 Omnidirectional mobile robot used as an agent in the robot swarm [1]



tribute within the remaining area. Whereas straight movement toward the area of interest is likely to be easier to detect, the antagonistic robot can, e.g., hide its target position by only slightly shifting its trajectory as later described in Sect. 3.1. Subsequently, the theoretical foundations necessary to understand the proposed approach are presented.

3 Theoretical foundations

The optimal solution to the deployment problem, the computation of antagonistic actions as well as the concept and architecture of a normalizing flow are described in detail in the following subsections.

3.1 Deployment problem

The optimal coverage of a polygonal area W is found by minimizing the cost function

$$H(\mathbf{x}_1, \dots, \mathbf{x}_n, W) = \sum_{i=1}^n \int_{W_i} \|\mathbf{q} - \mathbf{x}_i\|^2 \phi_i(\mathbf{q}) dW_i \quad (1)$$

with $\mathbf{x}_i \in \mathbb{R}^2$ being the position of robot i and W_i representing the Voronoi cell that encloses \mathbf{x}_i in the Voronoi decomposition of W generated for $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ [20]. For each Voronoi cell, $\mathbf{q} \in \mathbb{R}^2$ represents the positions that lie within W_i . The weighting function $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}$ determines the importance of different locations within the coverage area and is set to a uniform distribution by default. This leads to the benevolent robots assigning equal importance to every point in W . In contrast, the density function ϕ_{ant} of an antagonistic robot is set to a normal distribution $\phi_{\text{ant}} = \mathcal{N}(\mu_{\text{ant}}, \sigma_{\text{ant}})$, as depicted in the bottom row of Fig. 1. Using the Delaunay triangulation [21, 22] of its Voronoi cell W_{ant} , the antagonistic robot weights the points in W_{ant} according to their density value in ϕ_{ant} . The weighted center of W_{ant} is then selected as the target position for the current time step. The antagonistic robot behavior can be set to aggressive or stealthy behavior by means of decreasing or increasing the covariance σ_{ant} .

3.2 Normalizing flows

Normalizing flows are a type of neural network that learns a model of an unknown, potentially complicated target probability distribution p_x^* over continuous random variables [23, 24]. In order to evaluate p_x^* and draw samples from it, the normalizing flow learns an invertible and differentiable transformation function $T = T_K \circ \dots \circ T_1$. Each of the stacked transformations T_k is computed by one layer of the neural network. The transformation function maps from a base distribution $p_u(u)$ to $p_x(x; \theta)$, with $p_x(x; \theta)$ being the model of the target distribution approximated by a normalizing flow with parameters θ , that consist of the trainable parameters as well as the hyperparameters of the neural network. The transformation between both distributions is visualized in Fig. 3. The base distribution is usually chosen as a multivariate normal or uniform distribution or any other probability density that is easy to evaluate. Using the change of variables formula, see [25], the probability density value $p_x(x)$ of a data point x can thus be computed as

$$p_x(x) = p_u(T^{-1}(x)) |\det(J_{T^{-1}}(x))|, \quad (2)$$

where $J_{T^{-1}}$ represents the Jacobian matrix of the inverse transformation function T^{-1} with respect to x . Similarly, it is possible to sample from p_x by drawing a sample $u_s \sim p_u$ and then transforming the sample in order to get

$$x_s = T(u_s). \quad (3)$$

In order to fit the model distribution p_x to the target distribution p_x^* during training, the Kullback-Leibler divergence $D_{KL}[p_x^*(x) || p_x(x; \theta)]$ [26] between both distributions is minimized. As elaborated in [25], assuming that the training data is sampled from the target distribution p_x^* , this is equivalent to maximizing the probability of the training data, i.e.,

$$\begin{aligned} \mathcal{L}(\theta) &= D_{KL}[p_x^*(x) || p_x(x; \theta)] \\ &= -\mathbb{E}_{p_x^*(x)}[\log p_x(x; \theta)] + \text{const} \\ &\approx -\frac{1}{N} \sum_{n=1}^N \log p_u(T^{-1}(x_n; \theta)) + \log |\det(J_{T^{-1}}(x_n; \theta))| + \text{const}. \end{aligned} \quad (4)$$

Normalizing flows have been successfully used, e.g., for anomaly detection in industrial time series data [27] and for simulation-based inference [28].

This work employs a type of autoregressive flow with a spline-based transformer. Autoregressive flows compute the transformation T_k for each layer k using a strictly monotonic function τ called transformer function and a so-called conditioner function c_i , such that

$$z'_i = \tau(z_i, c_i(z_{<i})) \quad (5)$$

as shown in Fig. 3, see [25]. Since each c_i only depends on $z_{<i} = [z_0, \dots, z_{i-1}]$, the Jacobian matrix in Equation (2) is triangular and therefore easy to invert. The transformer function is built of quadratic splines [29] for K segments with endpoints z_{i0}, \dots, z_{iK} . For each layer, these segments are determined by a conditioner implemented as a masked autoencoder for distribution estimation (MADE) [30].

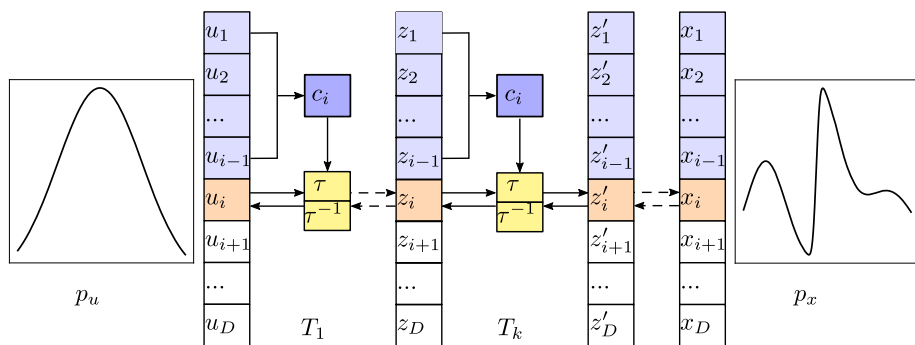


Fig. 3 The normalizing flow learns an invertible and differentiable transformation $T = T_K \circ \dots \circ T_1$ between a uniform base distribution p_u and the probability density function p_x . As a result, it is possible to sample actions from p_x , as well as evaluate the probability density value of a given action [25]

4 Detection approach

An important consideration in developing the detection approach is that anomalies, including antagonistic behavior, are characterized as having a low probability of occurrence. Consequently, there is rarely enough data available to narrow down and classify different anomalous patterns in order to detect them [9]. Instead, it is a common approach to anomaly detection to use data to learn a representation of normal behavior and to identify anomalies as the deviation thereof. Thus, under the assumption of unaltered normal behavior, the approach proposed in this work may, in principle, detect arbitrary physically anomalous behavior.

Further benefits of a data-driven approach to anomaly detection include its adaptability to different settings as long as corresponding data is available. For instance, the approach can be trained on the behavior of a swarm performing a different task or following a different algorithm for solving the coverage problem. In addition, the robustness of the model to environmental influences can be adapted by adding noise to the training data in order to increase the variance of the normal motion behavior.

While different types of neural networks have been considered, see Sect. 5, this work leverages the ability of an autoregressive flow with a spline-based transformer, also called neural spline flow (NSF), to model a conditional probability distribution $p(a|s; \theta)$ of a normal, i.e., benevolent, robot action. Both the action a and the state s are computed using the spatial information that is communicated between the robots, as shown in Fig. 4. The robot action a at time t_j is represented as the motion vector between a robot's position at t_j and its position at $t_{j+1} = t_j + \Delta t$. The state is composed of two parts. In order to get information about the closeness of other robots, the direction vector from the position of each robot to the position of each of its c nearest neighbors is computed. Since the robots must also take the boundary of the coverage area into account, the direction vector between each robot position and the positions of the coverage area's vertices is computed as well. Both the robot swarm size and the number of polygon vertices vary between runs, which allows one to deal with different polygon shapes and a variable number of robots. Consequently, the number of state features is variable as well and has to be embedded in a vector of fixed size before it is passed to the normalizing flow. With recurrent neural networks being suitable for an arbitrary number of features, we select the Long Short Term Memory (LSTM) [31] network architecture to embed the state information. Due to their dissimilar effect on the robot actions, the distances to the robot positions and the distances to the positions of the coverage

area's vertices are embedded separately, using one LSTM each so that two LSTMs are used in total. As visualized in Fig. 5, the embedding contains a concatenation of the last hidden states of both LSTMs. The entire network can then be trained end-to-end.

The capability of normalizing flows to generate samples is leveraged for categorizing an action as normal or anomalous. Since a trained model maximizes the log probability of training data samples under p_x , see Equation (4), it can be assumed that normal actions drawn from

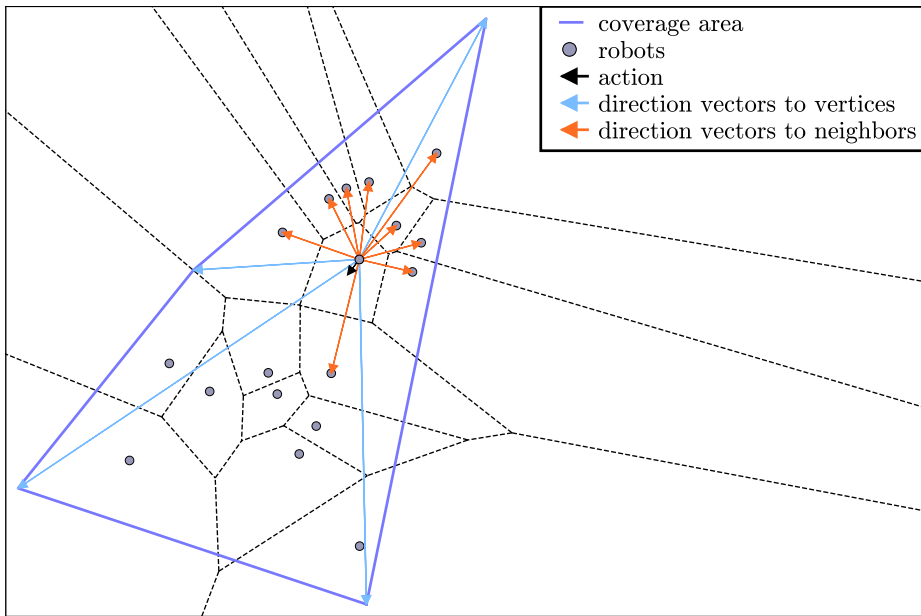


Fig. 4 Illustration of an emblematic situation of a deployment task. The corresponding data, i.e., a robot's action as well as the direction vectors to its neighbors and the area vertices, is collected at each time step. Color figure online

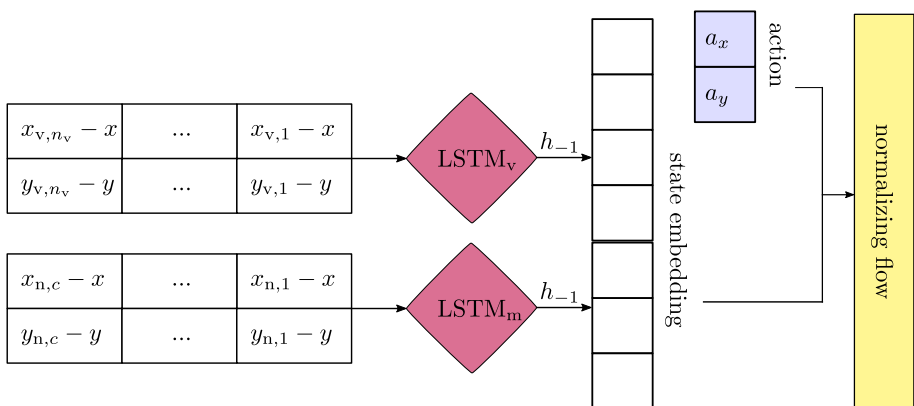


Fig. 5 Embedding of the distance between a robot position (x, y) and the positions of its nearest c neighbors and all n_v vertices of the coverage area

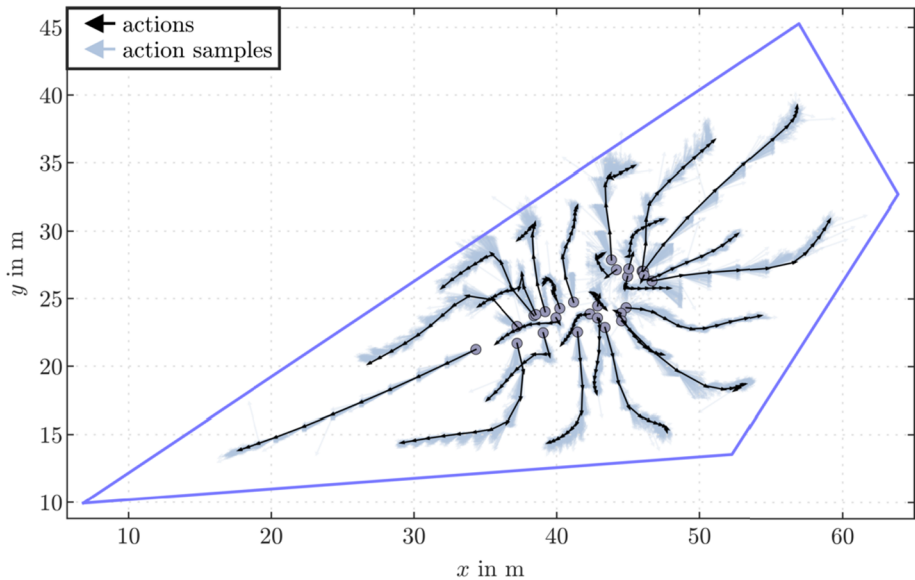


Fig. 6 Observed robot actions in a simulation run and action samples from the trained model. Color figure online

p_x lie in regions with a high probability value. According to this observation, the predicted probability density value of an observed action \mathbf{a}_{obs} should be significantly higher than the probability density value of the uniform base distribution,

$$p_x(\mathbf{a} = \mathbf{a}_{\text{obs}} | \mathbf{s}_{\text{obs}}) = p_u(\mathbf{T}^{-1}(\mathbf{a}_{\text{obs}}) | \mathbf{s}_{\text{obs}}) |\det(\mathbf{J}_{\mathbf{T}^{-1}}(\mathbf{x}))| \gg p_u(\mathbf{T}^{-1}(\mathbf{a}_{\text{obs}}) | \mathbf{s}_{\text{obs}}).$$

Consequently, an action is classified as anomal, if

$$p_x(\mathbf{a} = \mathbf{a}_{\text{obs}} | \mathbf{s}_{\text{obs}}) < s p_u(\mathbf{T}^{-1}(\mathbf{a}_{\text{obs}}) | \mathbf{s}_{\text{obs}}) \quad (6)$$

with a scaling factor s .

An approximation of the probability density of an action \mathbf{a}_{obs} at a state \mathbf{s}_{obs} can be computed by drawing samples $\mathbf{a}_{\text{sample}} \sim p_x(\mathbf{a} | \mathbf{s}_{\text{obs}}, \boldsymbol{\theta})$ from the trained model. Taking missing information from neighbors or noise into account, the observed action should lie approximately uniformly within the action samples. Figure 6 visualizes the observed actions as well as action samples.

The following subsections provide detailed information regarding the robots, the collection of simulation data, the hyperparameters, and the training of the neural network.

4.1 Robots

In simulation scenarios, the robots are modeled using a multibody model of an omnidirectional mobile robot with four Mecanum wheels as depicted in Fig. 2. The model is taken from [1, Sect. 4.3.2] and considers the robot chassis and the four wheels as individual bodies but, for simplicity purposes, neglects the robot's wheels' suspensions and considers only planar motion. The wheels are assumed to roll without slipping. Moreover, each wheel's contact point with the ground is assumed to have a position that is constant relative to the

robot chassis, and the wheels are modeled as one monolithic body each. The derivation of the robots' kinematics is rather intricate, which is due to the geometry of the Mecanum wheels and due to the fact that the model is subject to two nonholonomic kinematic constraints, which, however, do not affect the macroscopic maneuverability of the robot. Due to these intricacies, for compactness, the reader is kindly referred to [1, Sect. 4.3.2] for full detail on the model and its derivation. This work picks the same model parameters as in the reference since these are chosen to mimic the robot from Fig. 2. The simulation model's system-theoretic state vector comprises states sufficient to describe the rotatory motion of the robot chassis and wheels. To this paper's detection method, only robot positions are relevant and, hence, made available. Similar to the hardware robot, the simulated robot takes the desired translational velocities of the robot's center as inputs. From these, desired angular velocities for the wheels are calculated that kinematically fit the inputs. Local motor controllers are then tasked to govern the motors to quickly reach the desired angular wheel velocities. In simulations, as later on in hardware, these motor controllers run at a frequency of 100 Hz, whereas the desired chassis velocity is recalculated at a frequency of 5 Hz to always point into the direction of the area center of the robot's Voronoi cell. The robots reach a maximum velocity of 0.4 m/s in each direction, resulting in a maximum range of an action of $d_v = \Delta t \cdot \sqrt{0.4^2 + 0.4^2}$ m. The Voronoi tessellation and, in consequence, the cell centroids the robots aim for, are updated every $\Delta t = 5$ seconds.

4.2 Simulation data

The coverage area is represented as a convex polygon with an area of 300 m² to 1200 m². The number of entities in a robot swarm $n_r \in [10, 30] \cap \mathbb{N}$ and polygon vertices $n_v \in [3, 7] \cap \mathbb{N}$ are sampled uniformly. At the start of each coverage task, the robot swarm is randomly positioned within a coverage area that has been randomly generated while considering the constraints on the area and number of vertices. An example of the start configuration can be seen in the first column of Fig. 1. The coverage vertices and robot positions are scaled to lie between 0 and 1 in order to improve the range that the training data lies in. Each robot communicates its position at regular time steps t_j , with the time span Δt between sampling instances set to 5 seconds. Considering the possibility of communication issues due to poor signal coverage, a large coverage area, or a large number of robots in the swarm, robots only receive the position information of swarm entities that are spatially close. For validation and test data, it is assumed that for each robot the position of its $c = \min(25, n_r - 1)$ closest neighbors is available, with n_r being the maximum number of robots in the swarm. However, to enforce flexibility, during training the number of neighbors is sampled in the range $c = [\min(15, n_r - 1), 25]$. At each time step t_j a robot starts at position (x, y) and, using the position $(x_{t_{j+1}}, y_{t_{j+1}})$ at the next time step, we observe its action $(a_x, a_y) := (x_{t_{j+1}} - x, y_{t_{j+1}} - y)$. Additionally, state information is collected as the distance vectors from (x, y) to the robot's closest c neighbors, as well as the distance vectors to all n_v coverage area vertices. The neural network is trained on a dataset built by simulating 200 different coverage tasks of benevolent robot behavior for a duration of 10 time steps. Collecting the state and action of each robot at each time step, the training dataset contains 40,880 samples. The training performance is validated on a second dataset of normal robot behavior, simulating 50 coverage tasks that result in 10,220 data samples. For the test data that contains both normal and anomalous behavior, 100 coverage tasks are simulated, yielding 20,330 samples. In each of the 50 test simulations, one to three robots behave anomalously, with $P(\text{random}) = P(\text{idle}) = 0.25$ and $P(\text{antagonistic}) = 0.5$. The mean μ_{ant} of the antagonistic agent's area of interest is chosen randomly within the coverage area, while the covariance is restricted to $\sigma_{\text{ant}} \in \{0.5^2, 1^2, 1.5^2, \dots, 3.5^2\}$.

Table 1 Hyperparameters of the LSTM + NSF model

LSTM		NSF		Training	
layers	3	K		learning rate	0.00664
hidden size		layers		seed	31
neighbors	20	MADE		dropout	0.096
area	10	blocks per layer	1		
		hidden features	15		

4.3 Model hyperparameters and training

As described in Sect. 3.2, we use an autoregressive normalizing flow with a spline-based transformer and add an LSTM for the embedding of the state information. Since each action is conditioned on a state, the conditioner leverages both $\mathbf{z}_{<i}$ and the state embedding. In accordance with the dimensionality and range d_v of the robot motion per time step, the base distribution is defined as a two-dimensional uniform distribution, described by the probability density

$$p_u(\mathbf{u}) = \begin{cases} \frac{1}{(d_v - (-d_v))^2} & \text{for } -d_v \leq u_1 \leq d_v \text{ and } -d_v \leq u_2 \leq d_v, \\ 0 & \text{else.} \end{cases} \quad (7)$$

The final model contains 19,212 trainable parameters and has been trained for 178 training epochs needing a duration of 30 minutes on an NVIDIA GeForce RTX 2070 graphics card. Since almost half of the actions are smaller than 0.5 m due to the robots converging, a weighted random sampler is employed to ensure that the model is trained uniformly on actions of different magnitudes. The model and training hyperparameters of the model can be found in Table 1. They are selected using the random hyperparameter optimization provided by `weights and biases` (WandB) [32]. The code is based on the Python [33] package `nflows` [34] and the model is trained using `PyTorch` [35].

5 Results

The performance of the final model is evaluated on the validation set based on multiple criteria. The log probability of the actions in the weighted training set increases from -2.96 after the first training epoch to 1.12 after 178 epochs, with both training and validation loss showing converging behavior. The unweighted final validation log probability is 0.77 . Additionally, if the normalizing flow fits the target density well, transforming the training data to values in the base distribution p_u should result in noise that also resembles the base distribution [36]. As shown in Fig. 7, this validation criterium is fulfilled reasonably well.

After the training and validation of the model is completed, the trained model can be used to investigate the test dataset containing both anomalous and benevolent robots. Observed actions are classified as anomal according to Equation (6), inserting the base uniform probability density value $p_u(\mathbf{u}_v) = 0.03125$ and setting the scaling factor to $s = 2$, such that \mathbf{a}_{obs} is anomal if $p_x(\mathbf{a} = \mathbf{a}_{\text{obs}}) < 0.625$. From the samples in the validation dataset, only 4.3% of the actions fall below this threshold. Thus, the probability of a robot performing more than 2-anomal steps is about 0.8% and a robot is categorized as anomal when taking more than 2-anomal steps. Table 2 summarizes the model performance on the test data

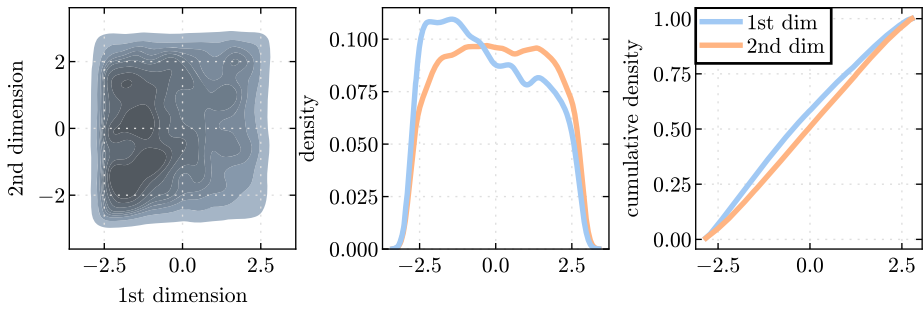


Fig. 7 Kernel density estimation and cumulative density for both dimensions of the training data transformed to noise values

Table 2 Accuracy, percentage of actions categorized as anomalous, and detection accuracy based on a threshold of more than two anomalous actions for all robot types. The results for the antagonistic robot are first presented for all σ_{ant} , then evaluated for the different σ_{ant}

Robot type	σ_{ant}	% Anomal steps	Accuracy
normal		7.5	0.908
idle		75.4	0.971
random		90.9	1.0
antagonistic		79.3	0.906
	0.25	68.5	0.788
	1.0	76.1	0.944
	2.25	90.44	0.957
	4.0	84.1	0.955
	6.25	74.8	0.889
	9.0	81.9	0.938
	12.25	85.2	0.952

for the normal, erroneous, and antagonistic robots. The detection method proposed in this work yielded an accuracy of correctly identifying anomalous robot behavior exceeding 90% for all three robot types, i.e., idle, random, and antagonistic. While the random robot is detected in 100% of the cases, the idle robot probably benefits from actions taken close to convergence that have a very small magnitude and is therefore close to no motion. Most importantly, the antagonistic robot is detected in 90% of the cases. Since antagonistic robots can be coincidentally close to their antagonistic target position, it is expected that some of them will behave almost normally. The accuracy is therefore satisfying for all robot types.

Figure 8 visualizes how larger covariances of the antagonistic area of interest lead to robots moving less aggressively towards the area. For the covariance $\sigma_{\text{ant}} = 0.25$, most antagonistic robots move directly to the center of the area μ_{ant} . With increasing covariance, the end positions of the robots are further away from the center of the target area, indicating a more cautious movement towards the area of interest. However, due to their rapidly decreasing density, it can also be seen that areas with a small covariance have little influence on robots with large starting distances. The lower right plot depicts, for each covariance, the percentage of antagonistic steps categorized as anomalous. There is no apparent correlation between detection accuracy and covariance.

In addition, Figs. 9, 10, and 11 depict the detection of antagonistic and erroneous robots in different coverage settings. The figures trace the robot's motions over 10 time steps, with the black arrows being the actions, and the blue-gray arrows representing samples from

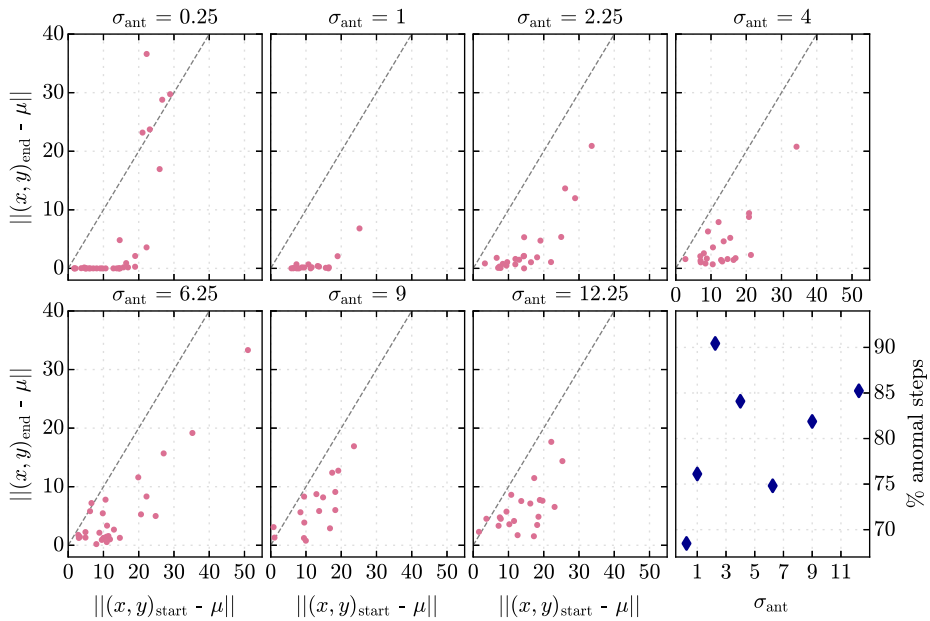


Fig. 8 The robot's incentive to move towards the antagonistic target depends on the covariance σ_{ant} of the target area. Plots 1 to 7 depict the Euclidean distance in m between the antagonistic target and the robot's position at time t_0 on the x-axis and the position at time t_{10} on the y-axis. Larger covariances lead to robots moving faster towards the target area. The lower right plot shows the detection accuracy per covariance

the action density. A robot is marked with a square if it has exceeded the threshold of two anomalous actions and is consequently categorized as anomalous. The robot swarm in Fig. 9 contains one random and two antagonistic agents. The behavior of the left antagonistic robot is set to maximum stealth by setting the covariance of its target area to $\sigma_{\text{ant}} = 12.25$. In contrast, the right robot moves aggressively towards its area of interest with $\sigma_{\text{ant}} = 0.25$. While the neural network has no information about these covariances, it correctly categorizes all robots after only a few steps. In Fig. 10, one of the three antagonistic robots has a target close to its starting position and to its optimal coverage position. Consequently, the robot's behavior is nearly normal, and the robot is not detected. Whereas unfortunate, this is to be expected of an anomalous robot that, by pure chance, behaves almost identically as a normal one given the same situation, and is therefore more a question of observability independent of the detection method employed. Figure 11 visualizes a different kind of difficulty. From the qualitative results, it appears that large distances between vertices lead to an inadequate representation of the connecting edge. Thus, the predicted actions of robots close to long edges are skewed towards the edge of the coverage area, thereby causing some of the benevolent robots to be categorized as anomalous. In order to mitigate this problem, further investigation is required.

As an alternative to the approach described so far, as a lightweight approach to learning an action probability, we investigated Gaussian process regression (GPR) [37] in a setting of five robots within a fixed coverage area. The GPR was able to fit into the training data and yielded worse predictions for anomalous actions than for normal actions. However, even though the setting was simplified, the GPR overall was not able to generalize well to unknown states.

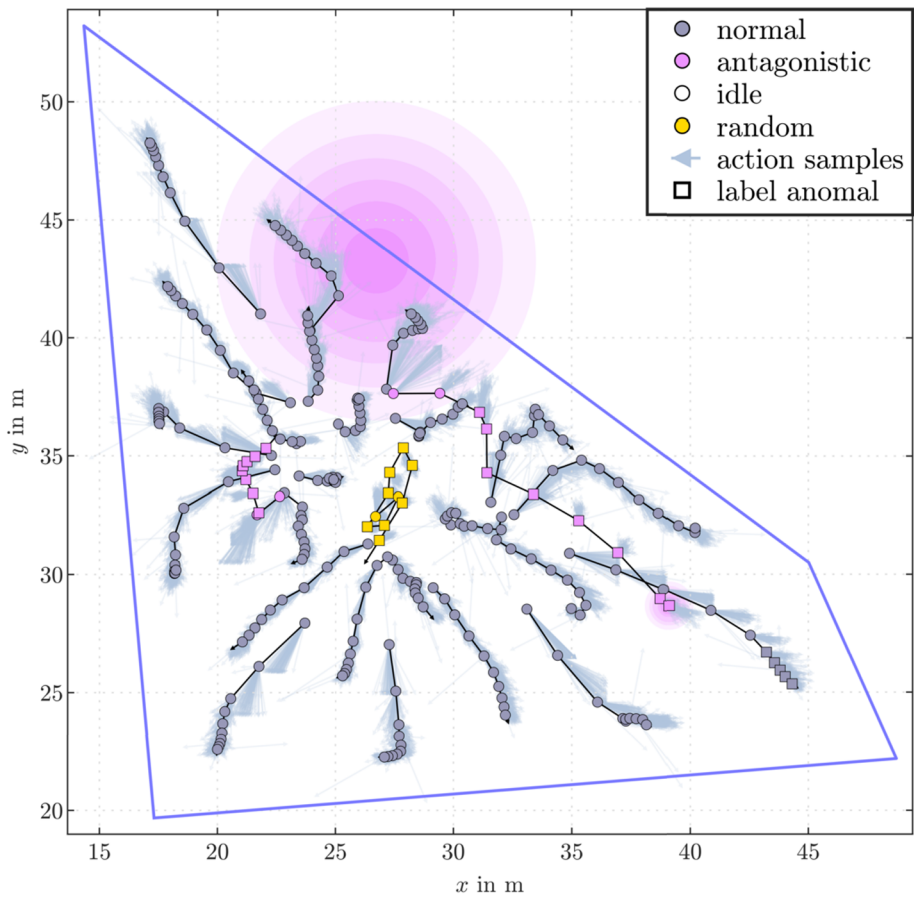


Fig. 9 Robot swarm containing 2 antagonistic robots with $\sigma_{\text{ant}} = 0.25$ for the more aggressive robot and $\sigma_{\text{ant}} = 12.25$ for the rather stealthy robot. Color figure online

Besides that, the spatial dispersion of the robot entities and the vertices of the coverage area can be represented as a graph, therefore we also investigated different graph neural network methods for the state embedding [38–41]. However, whereas various approaches learned the local structure of neighbors and the area around a robot entity, the resulting state embedding was rotation invariant. This rotation invariance cannot be reconciled with the robot actions describing a motion in a certain direction within the area's coordinate system.

Since the robot actions are computed deterministically in the current setting, it might be possible to use a purely threshold-based detection approach. This means that robot entities could compute the optimal action of a neighboring robot and check if the Euclidean distance between optimal action and executed action exceeds a threshold. However, a clear advantage of the probabilistic approach presented in this work is that it results in a very intuitive anomaly recognition based on the probability of an action and simultaneously allows quantifying the uncertainty about an action being anomalous.

Additionally, in many real-world use cases the optimal action used for a threshold-based approach is not known or only partially known. This includes robots that have been trained with reinforcement learning [42], robot swarms that have been deployed within an area that

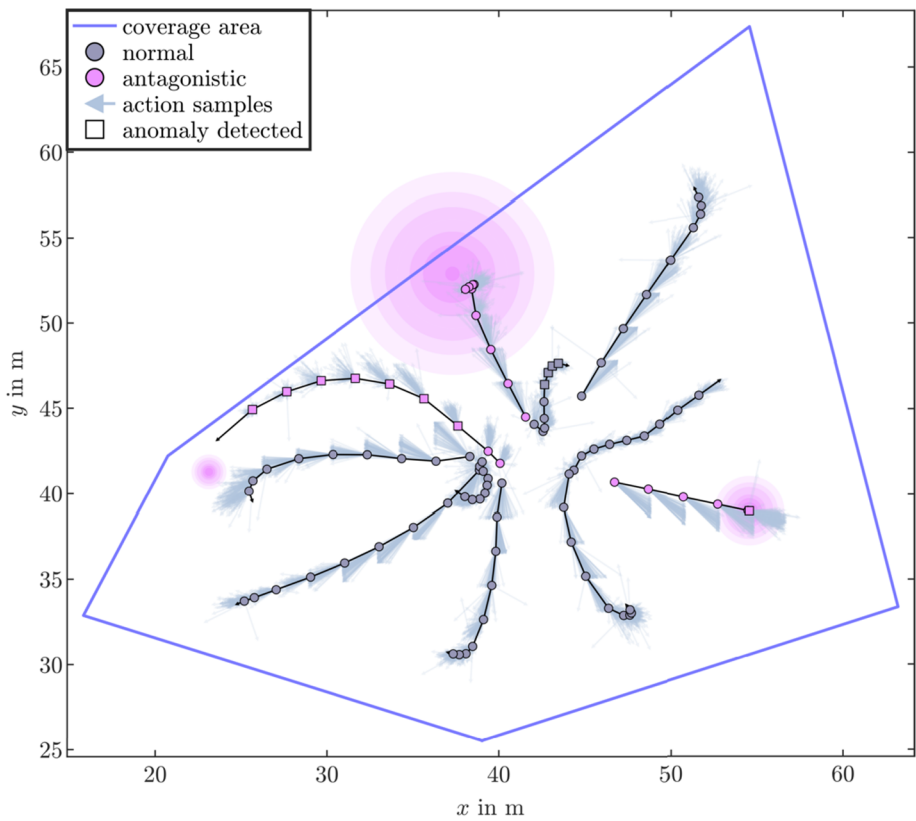


Fig. 10 In this run, one of the antagonistic robot's start position allows it to perform near-normal motions towards its target. As a result, it is not detected. Color figure online

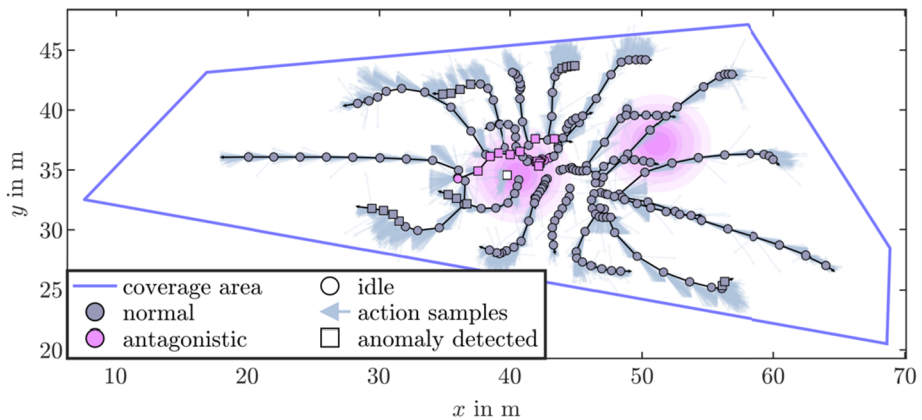


Fig. 11 Large distances between vertices of a coverage area may lead to actions being shifted towards the edges. As a result, multiple normal robots are mistakenly being categorized as anomalous in this run. Color figure online

contains obstacles, such that a robot might need to dynamically adapt its behavior to avoid collisions [43], or agents performing complex tasks that are very expensive to compute, such as road traffic [15].

6 Conclusion

This paper presents a method for the reliable detection of antagonistic robots in a deployment setting. The approach uses a neural network comprising an LSTM and a normalizing flow to learn the probability density of robot actions and categorize actions with a low probability value as anomalous. The neural network was trained solely on the benevolent behavior of a robot swarm being deployed in diverse coverage areas, consequently not being limited to the detection of a specific type of anomalous behavior. The presented detection approach was able to correctly identify more than 90 percent of the antagonistic and erroneous robots used for the testing phase. All things considered, the proposed methodology is suitable for alleviating a variety of security challenges for robot swarms that are mentioned in [3], especially for intrusion detection, ensuring swarm mobility and adhering to energy constraints.

We plan to extend our work by validating the applicability of the proposed detection approach in settings with environmental noise, obstacles, and more advanced antagonistic agents. Leveraging the intuitiveness of our probabilistic approach, we plan to introduce a dynamic adaptation of the anomaly threshold that reacts to disturbances caused by environmental influences on multiple entities in the robot swarm. A further interesting point of research is the development of counteractions against antagonistic agents.

Author contributions I.W. has done most of the research and wrote the paper draft (50%). H.E. contributed a lot to the research, wrote a part of the manuscript and corrected the paper (40%). P.E. has contributed to the research and corrected the paper (10%).

Funding Open Access funding enabled and organized by Projekt DEAL. This work is partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 501890093 within the DFG Priority Programme 'SPP 2353: Daring More Intelligence – Design Assistants in Mechanics and Dynamics'. Moreover, it is supported also under DFG project number 433183605 (Research on Multibody Dynamics and Control for Collaborative Elastic Object Transportation by a Heterogeneous Swarm with Aerial and Land-Based Mobile Robots) and through Germany's Excellence Strategy (Project PN4-4 Theoretical Guarantees for Predictive Control in Adaptive Multi-Agent Scenarios) under Grant EXC 2075-390740016 (SimTech).

Declarations

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Ebel, H.: Distributed Control and Organization of Communicating Mobile Robots: Design, Simulation, and Experimentation. Dissertation, Schriften aus dem Institut für Technische und Numerische Mechanik der Universität Stuttgart, vol. 69. Shaker Verlag, Düren (2021). <https://doi.org/10.2370/9783844081725>
2. Parker, L.E.: ALLIANCE: an architecture for fault tolerant multirobot cooperation. *IEEE Trans. Robot. Autom.* **14**(2), 220–240 (1998). <https://doi.org/10.1109/70.681242>
3. Higgins, F., Tomlinson, A., Martin, K.M.: Survey on security challenges for swarm robotics. In: 2009 Fifth International Conference on Autonomic and Autonomous Systems, pp. 307–312 (2009). <https://doi.org/10.1109/ICAS.2009.62>
4. Subchan, S., White, B.A., Tsourdos, A., Shanmugavel, M., Zbikowski, R.: Pythagorean hodograph (PH) path planning for tracking airborne contaminant using sensor swarm. In: 2008 IEEE Instrumentation and Measurement Technology Conference, pp. 501–506. IEEE, Victoria (2008). <https://doi.org/10.1109/IMTC.2008.4547087>
5. Spears, D., Kerr, W., Spears, W.: Physics-based robot swarms for coverage problems. *Int. J. Intell. Control Syst.* **11**(3), 11–23 (2006)
6. Elamvazhuthi, K., Berman, S.: Optimal control of stochastic coverage strategies for robotic swarms. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 1822–1829 (2015). <https://doi.org/10.1109/ICRA.2015.7139435>
7. Albani, D., Manoni, T., Nardi, D., Trianni, V.: Dynamic UAV swarm deployment for non-uniform coverage. In: Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems. AAMAS '18, pp. 523–531. International Foundation for Autonomous Agents and Multiagent Systems, Stockholm (2018). <https://dl.acm.org/doi/10.5555/3237383.3237462>
8. Sargeant, I., Tomlinson, A.: Review of potential attacks on robotic swarms. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) Proceedings of SAI Intelligent Systems Conference (IntelliSys), vol. 16, pp. 628–646. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-56991-8_46
9. Qin, L., He, X., Zhou, D.H.: A survey of fault diagnosis for swarm systems. *Syst. Sci. Control Eng.* **2**(1), 13–23 (2014). <https://doi.org/10.1080/21642583.2013.873745>
10. Sargeant, I., Tomlinson, A.: Intrusion detection in robotic swarms. In: Bi, Y., Bhatia, R., Kapoor, S. (eds.) Intelligent Systems and Applications, vol. 1038, pp. 968–980. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-29513-4_71
11. Franceschelli, M., Egerstedt, M., Giua, A.: Motion probes for fault detection and recovery in networked control systems. In: 2008 American Control Conference, pp. 4358–4363 (2008). <https://doi.org/10.1109/ACC.2008.4587180>
12. Saulnier, K., Saldaña, D., Prorok, A., Pappas, G.J., Kumar, V.: Resilient flocking for mobile robot teams. *IEEE Robot. Autom. Lett.* **2**(2), 1039–1046 (2017). <https://doi.org/10.1109/LRA.2017.2655142>
13. Strobel, V., Castelló Ferrer, E., Dorigo, M.: Managing Byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '18, pp. 541–549. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2018). <https://dl.acm.org/doi/10.5555/3237383.3237464>
14. Cavorsi, M., Akgün, O.E., Yemini, M., Goldsmith, A.J., Gil, S.: Exploiting trust for resilient hypothesis testing with malicious robots. In: 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 7663–7669 (2023). <https://doi.org/10.1109/ICRA48891.2023.10160385>
15. Vuong, T.P., Loukas, G., Gan, D., Bezemskij, A.: Decision tree-based detection of denial of service and command injection attacks on robotic vehicles. In: 2015 IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6 (2015). <https://doi.org/10.1109/WIFS.2015.7368559>
16. Basan, E., Basan, A., Nekrasov, A.: Method for detecting abnormal activity in a group of mobile robots. *Sensors* **19**(18), 4007 (2019). <https://doi.org/10.3390/s19184007>
17. Candido, S., Hutchinson, S.: Detecting intrusion faults in remotely controlled systems. In: 2009 American Control Conference, pp. 4968–4973 (2009). <https://doi.org/10.1109/ACC.2009.5160086>
18. Lloyd, S.: Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **28**(2), 129–137 (1982). <https://doi.org/10.1109/TIT.1982.1056489>
19. Ilon, B.E.: Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base. 3,876,255 (1975)
20. Voronoi, G.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire sur quelques propriétés des formes quadratiques positives parfaites. *J. Reine Angew. Math.* **1908**(133), 97–102 (1908). <https://doi.org/10.1515/crll.1908.133.97>
21. Delaunay, B.: Sur la sphère vide. *Bull. Acad. Sci. URSS, Class. Sci. Math. Nat.* **6**, 793–800 (1934)

22. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: Fundamental algorithms for scientific computing in Python. *Nat. Methods* **17**, 261–272 (2020). <https://doi.org/10.1038/s41592-019-0686-2>
23. Tabak, E.G., Turner, C.V.: A family of nonparametric density estimation algorithms. *Commun. Pure Appl. Math.* **66**(2), 145–164 (2013). <https://doi.org/10.1002/cpa.21423>
24. Rezende, D.J., Mohamed, S.: Variational inference with normalizing flows. In: Bach, F., Blei, D. (eds.) *International Conference on Machine Learning*. Proceedings of Machine Learning Research, vol. 37, pp. 1530–1538 (2015). <https://dl.acm.org/doi/10.5555/3045118.3045281>
25. Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S., Lakshminarayanan, B.: Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.* **22**(1), 2617–2680 (2021). <https://dl.acm.org/doi/abs/10.5555/3546258.3546315>
26. Kullback, S., Leibler, R.A.: On information and sufficiency. *Ann. Math. Stat.* **22**(1), 79–86 (1951). <https://doi.org/10.1214/aoms/1177729694>
27. Schmidt, M., Simic, M.: Normalizing flows for novelty detection in industrial time series data (2019). <https://arxiv.org/abs/1906.06904>
28. Boelts, J., Lueckmann, J.-M., Gao, R., Macke, J.H.: Flexible and efficient simulation-based inference for models of decision-making. *eLife* **11**, 77220 (2022). <https://doi.org/10.7554/eLife.77220>
29. Müller, T., McWilliams, B., Rousselle, F., Gross, M., Novák, J.: Neural importance sampling. *ACM Trans. Graph.* **38**(5), 1–19 (2019). <https://doi.org/10.1145/3341156>
30. Germain, M., Gregor, K., Murray, I., Larochelle, H.: MADE: masked autoencoder for distribution estimation. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. ICML'15, vol. 37, pp. 881–889. JMLR.org, Lille (2015). <https://dl.acm.org/doi/10.5555/3045118.3045213>
31. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997). <https://doi.org/10.1162/neco.1997.9.8.1735>
32. Biewald, L.: Experiment tracking with weights and biases (2020). Software available from [wandb.com](https://www.wandb.com/). <https://www.wandb.com/>
33. Van Rossum, G., Drake, F.L.: *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA (2009). <https://dl.acm.org/doi/book/10.5555/1593511>
34. Durkan, C., Bekasov, A., Murray, I., Papamakarios, G.: nflows: normalizing flows in PyTorch. Zenodo (2020). <https://doi.org/10.5281/zenodo.4296287>
35. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 33, pp. 8024–8035. Curran Associates, Red Hook (2019). <https://dl.acm.org/doi/10.5555/3454287.3455008>
36. Papamakarios, G., Pavlakou, T., Murray, I.: Masked autoregressive flow for density estimation. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17, pp. 2335–2344. Curran Associates, Red Hook (2017). <https://dl.acm.org/doi/10.5555/3294771.3294994>
37. MacKay, D.J.: *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge (2003). <https://dl.acm.org/doi/10.5555/3237383.3237462>
38. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17, pp. 1025–1035. Curran Associates, Red Hook (2017). <https://dl.acm.org/doi/abs/10.5555/3294771.3294869>
39. Donnat, C., Zitnik, M., Hallac, D., Leskovec, J.: Learning structural node embeddings via diffusion wavelets. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18, pp. 1320–1329. Association for Computing Machinery, New York (2018). <https://doi.org/10.1145/3219819.3220025>
40. You, J., Ying, R., Leskovec, J.: Position-aware graph neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) *Proceedings of the 36th International Conference on Machine Learning*. Proceedings of Machine Learning Research, vol. 97, pp. 7134–7143. PMLR (2019). <https://doi.org/10.1016/j.neucom.2023.126581>
41. Salha, G., Hennequin, R., Vazirgiannis, M.: Keep it simple: graph autoencoders without graph convolutional networks (2019). <https://doi.org/10.48550/arXiv.1910.00942>

42. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* **32**(11), 1238–1274 (2013). <https://doi.org/10.1177/0278364913495721>
43. Ebel, H., Luo, W., Yu, F., Tang, Q., Eberhard, P.: Design and experimental validation of a distributed cooperative transportation scheme. *IEEE Trans. Autom. Sci. Eng.* **18**(3), 1157–1169 (2021). <https://doi.org/10.1109/TASE.2020.2997411>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.