A Distributed Approach for Fault Detection in Swarms of Robots

Alessandro Carminati
Politecnico di Milano
Milano, Italy
alessandro.carminati@polimi.it

Davide Azzalini
Politecnico di Milano
Milano, Italy
davide.azzalini@polimi.it

Simone Vantini
Politecnico di Milano
Milano, Italy
simone.vantini@polimi.it

Francesco Amigoni
Politecnico di Milano
Milano, Italy
francesco.amigoni@polimi.it

Abstract

Swarm Robotic Systems (SRSs) are multi-robot systems usually composed of relatively simple robots. Local decisions and communication between robots allow for the emergence of complex behaviors of the entire SRS. The distributed nature of the SRSs enables their use in many real-world applications. However, despite the common belief that these systems are inherently robust and fault tolerant, it has been shown that even a few faulty robots could considerably hinder the performance of the entire SRS. In this paper, we propose a distributed fault detection approach that exploits machine learning classifiers to allow each robot of a SRS to detect faults in other robots and/or in itself. The proposed fault detection approach is data-driven, requiring a reduced amount of explicit domain knowledge, and is based on data that can be easily collected by common swarm robotics platforms. We test the proposed fault detection approach in simulation and analyze the results using non-parametric statistical tests. Our extensive experimental campaign shows that our approach has good performance and is robust regardless of the ratio of faulty robots in the SRS.

Keywords

Swarm Robotics; Fault Detection

ACM Reference Format:
Alessandro Carminati, Davide Azzalini, Simone Vantini, and Francesco Amigoni. 2024. A Distributed Approach for Fault Detection in Swarms of Robots. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

A Swarm Robotic System (SRS) is a distributed multi-robot system composed of relatively simple robots. Local decisions and communication between robots allow for the emergence of complex behaviors of the entire SRS [8]. The distributed nature of the SRSs enables their use in many real-world applications, e.g., logistics, precision agriculture, and environmental monitoring [25].

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Due to the usually large number of robots composing a swarm, these systems are often considered to be inherently robust and fault-tolerant. However, Winfield *et al.* [29] show that even a few faulty robots could considerably hinder the work of the entire SRS. Thus, various distributed fault detection approaches have been proposed in the last few years [4]. These approaches exploit the distributed nature of the SRSs, allowing each robot to detect faults in the other robots and/or in itself [5, 12, 15, 17, 23, 24, 26, 27], usually requiring relatively little efforts for computation and communication due to the simple nature of robots.

Designing such fault detection approaches often requires to elicitate domain knowledge about the environment, the robots, and the task; this knowledge informs decisions such as how to process the measurements on the observed robots, how to choose the parameters of the classification methods, and how to create models of non-faulty robot behavior [13]. The complexity of the real-world applications of the SRSs could make it complicated to obtain and formalize such domain knowledge. On the other hand, in recent years, data storing solutions have become faster and cheaper and, at the same time, the field of machine learning has grown considerably, with many real-world applications [22].

In this paper, we propose a fault detection approach exploiting machine learning methods to classify a robot as faulty or non-faulty. We propose to model the observed robots with numerical features, obtained with minimal processing of raw measurements so to leave freedom to the machine learning methods, which automatically learn the best way to elaborate further on the features to obtain better performance in classification. We test the proposed fault detection approach in simulation with several swarm behaviors. Then, we analyze the results and compare the performance against the fault detection approach described in [24] using non-parametric statistical tests. The analysis highlights that the proposed fault detection approach often outperforms that of [24] and that its performance remains robust even when the number of faulty robots becomes large.

The main original contributions of this paper are two: (1) the use of machine learning classifiers for labeling the robots of the swarm as faulty or non-faulty and (2) the use of numerical features close to raw measurements.

This paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed fault detection approach. Section 4 describes the setup for the experiments. Section 5 reports the results of the experiments and their analysis. Section 6 discusses

the implications of the obtained results and directions for future studies. Section 7 presents our conclusions.

The Supplementary Material and the code for the experiments are at: `github.com/AleCarminati/ml-faultdetection-epuck`.

2 RELATED WORK

A SRS is "a system that consists of multiple intelligent interconnected nodes and possesses swarm capability" [20]. While robots considered individually are "relatively incapable or inefficient on their own with respect to the task at hand" [21], their ability to communicate and to interact with other nearby robots allows the SRS to carry out the required task through the cooperation of the robots. Due to their distributed nature, SRSs do not require centralized or hierarchical control [29].

In principle, an SRS is inherently robust [8, 21, 29]: its fully distributed nature implies the absence of a single point of failure, and non-faulty robots can compensate for faulty ones. However, Winfield *et al.* [29] show that some faults, which cause partial failures in robots, can considerably hinder the work of the entire SRS. Therefore, it emerges the necessity for a distributed fault detection approach, which allows each robot to detect faults in other robots [3, 4] and/or in itself.

Fault detection approaches for robot swarms are mainly characterized by the limited requirements in terms of computation and communication (according to the simple nature of robots) and can be categorized according to their scope and the classification algorithm they use. The scope can be *exogenous* or *endogenous* [24]. In the first case, a robot runs the fault detection algorithm to detect faults in other robots; in the second case, a robot runs the fault detection algorithm to detect faults in itself. Some fault detection algorithms can detect faults both in an endogenous and an exogenous way. Classification algorithms analyze collected data to classify a robot as faulty or non-faulty. Qin *et al.* [20] divide them into two sorts: *quantitative* and *qualitative* classification algorithms. Qualitative classification algorithms exploit straightforward schemes that do not need complex models of the behavior of the robots. Therefore, they are resource-saving and time-saving but not precise [20]. For instance, the classification algorithm in [1] exploits fuzzy rules for classification of faulty agents in a satellite formation flight. Quantitative classification algorithms can be further classified into two classes [20]: *model-based* and *data-driven*, which are discussed below.

Model-based classification algorithms use a model of the behavior of the robots based on domain knowledge about the robots and the task. Christensen *et al.* [5] propose one of the first fault detection approaches for an SRS. The behavior of the SRS, inspired by some species of fireflies, consists in synchronizing the flashing of the LEDs of the robots. To detect faults, each robot scans its surroundings for out-of-sync robots. Millard *et al.* [17] propose an exogenous fault-detection approach with a model-based classification algorithm that simulates the controller of another robot to predict its behavior, which, if different from the observed one, may signal a fault.

Data-driven classification algorithms collect data to learn autonomously to distinguish between faulty and non-faulty behaviors. In general, they need less prior domain knowledge w.r.t. model-based classification algorithms. Several fault detection approaches based on data-driven classification methods have been proposed. In [15], the classification algorithm is based on statistical error detection and is used for endogenous fault detection: each robot compares the values obtained on some task-specific performance indicators with the ones of its neighbors to detect if it is faulty. In [12], each robot r compares data it collected about itself with data that another robot r' collected about r. If they are coherent, both robots are classified as non-faulty; otherwise, they consult a third robot r'' to detect which one is faulty. The authors of [16] present a new data-driven method to allow a robot to extract metrics which can distinguish between normal and faulty states.

The approach that is closest to the one we present in this paper is the work of Tarapore *et al.* [23, 24, 26, 27], where an exogenous fault detection approach exploiting a classification algorithm based on the Cross-Regulation Model (CRM) [24, 26] is presented. The CRM exploits six binary features obtained by processing and thresholding the measurements on an observed robot and returns a faulty or a non-faulty outcome. The results of the classification algorithm are shared between robots, using a majority voting scheme to decide collectively if a robot is faulty or non-faulty. The feasibility of this fault detection approach has been shown also on a physical robot swarm in [27]. Differently from this approach, we use numerical features and machine learning classifiers, as detailed in the next section.

We remark that, although the literature on fault detection in single-robot systems is quite rich [13], those methods are generally not applicable to SRSs. In fact, they usually are only endogenous fault detectors, and often require computational capabilities beyond those available in common SRS platforms [14]. Moreover, we remark that swarm robotic fault detection differs from the detection of Byzantine threats in multirobot and swarm systems, as these represent adversarial rather than malfunctioning robots [6].

In this work, we apply some of these solutions to the SRS field by proposing an exogenous fault detection approach based on binary and numerical features and exploiting machine learning methods for data-driven classification.

3 THE PROPOSED FAULT DETECTION APPROACH

The fault detection approach we propose requires every robot of the SRS to have the ability of sending and receiving messages and to feature sensors to detect the distance and heading to the neighboring robots. These requirements are often satisfied by most swarm robot platforms. For instance, Range And Bearing (RAB) sensors and actuators are available on e-pucks [18]. They can broadcast simple messages, and when they receive a message, they can detect the distance and the angle of the sender.

3.1 Overview

We present the proposed fault detection algorithm's skeleton, which is similar to the one described in [24], in Algorithm 1. It considers control cycles of T_{cycle} seconds and experiments of T_{exp} seconds.

Phase A is executed for the entire duration of the experiment: each robot observes its neighbors and acquires and processes the

Algorithm 1 Proposed fault detection algorithm's structure

```
t = 0
while t < T_exp do
    Phase A: Observe and Process
    if (t mod T_cycle) ≤ T_clas then
        Phase B: Classify
    else
        Phase C: Vote
    end if
    increase t
end while
```

raw measurements to calculate the features that are used to model them. In the first T_{clas} seconds of the control cycle T_{cycle}, *phase B* is executed: each robot uses the classification method to classify the other robots it has observed as faulty or non-faulty. After the first T_{clas} seconds of each control cycle, each robot consolidates its individual classifications on the other robots. Robot r_i individually classifies robot r_j as faulty if and only if, in the last T_{clas} seconds, r_i classified r_j as faulty for the majority of the times. In the remaining $(T_{cycle} - T_{clas})$ seconds, *phase C* is executed: the robots communicate their individual classifications of the observed robots to the other robots. Using a majority voting scheme, they collectively classify if a robot is faulty or non-faulty. Note that the collective classifications obtained in one cycle do not influence the classifications of the other cycles.

The three phases are detailed in the following.

3.2 Phase A: Observe and Process

In this phase, each robot takes measurements of its neighbors. We define as d_{ij} the distance between robot r_i and robot r_j, measured by r_i. We define as ϕ_{ij} the angle between r_i's and r_j's headings, measured by r_i. The raw measurements are then processed to obtain the features that model the observed robots' behavior. The proposed fault detection algorithm can be used with two types of features: binary and numerical.

3.2.1 Binary Features The six binary features ($F1, F2, \ldots, F6$) of the proposed fault detection approach are the same used in [24].

Features $F1$ and $F2$ model if, for the majority of the past observations collected in a time window W_l, the observed robot r_j had at least a neighbor in the ranges $[0, 15]$ cm and $(15, 30]$ cm, respectively. Robot r_i measures the distance between two robots, r_j and r_k, using the Law of Cosines: $(d_{ij}^2 + d_{ik}^2 - 2d_{ij}d_{ik}\cos(\phi_{ij} - \phi_{ik}))^{\frac{1}{2}}$.

Features $F3$ and $F4$ model the motor actions of the observed robot. Both features require estimating the distance traversed by the observed robot r_j in a time window using Algorithm 2. Feature $F3$ is equal to 1 if and only if $Dist[r_i, r_j, W_l] > 0.15 \cdot W_l \cdot v_{max}$, where v_{max} is the robot's maximum speed. Feature $F4$ models the proportion of times the observed robot r_j changed its heading. The change in heading of r_j, as observed by r_i, is encoded by a binary value M_j, which is equal to one when $\frac{\dot{\omega}_j}{\dot{\omega}_{max}} \cdot \frac{Dist[r_i, r_j, W_l]}{W_s v_{max}} > 0.1$, where $\frac{\ddot{\omega}_j}{\ddot{\omega}_{max}}$ is the normalized angular acceleration of robot r_j. The value $\frac{\ddot{\omega}_j}{\ddot{\omega}_{max}}$ is calculated by robot r_j and then sent to its neighbors. Feature $F4$ is equal to 1 if and only if the proportion between the number of times r_j is observed with $M_j = 1$ and the total number of times r_j is observed in W_l is larger than 0.05.

Features $F5$ and $F6$ model the relationship between the change in heading of the observed robot r_j and the presence of nearby robots. Feature $F5$ is equal to 1 if and only if the proportion between the number of times r_j is observed with $M_j = 1$ and at least one neighbor in the range $[0, 30]$ cm, and the total number of times r_j is observed in W_l is larger than 0.05. Feature $F6$ is equal to 1 if and only if the proportion between the number of times r_j is observed with $M_j = 1$ and no neighbors in the range $[0, 30]$ cm, and the total number of times r_j is observed in W_l is larger than 0.05.

These binary features and the thresholds used to calculate their values are rather ad hoc and require domain knowledge in order to be defined. Further details can be found in [24].

Voting to improve the estimation of binary features As in [24], when a robot r_i processes the binary features about robot r_j, it sends them to its neighbors. If robot r_i observes r_j, every binary feature about r_j that r_i processes is then updated using a majority voting scheme: each message sent by a neighbour r_k and received by r_i with the binary features about r_j is a vote,in addition to the binary features about r_j that r_i processed itself. This process exploits the distributed nature of the SRS to obtain more robust binary features.

3.2.2 Numerical Features Beyond the binary features of [24] we introduce numerical features $f1, f2, \ldots, f5$.

Features $f1$ and $f2$ represent the speeds of the left and right wheels of the observed robot, respectively. They are obtained from the speed of the wheels that every robot communicates to the neighboring robots. Note that these numerical features are used to compute binary features $F4$, $F5$, and $F6$. Indeed, $\dot{\omega}_j = \frac{f2 - f1}{h}$ (where h is the distance between the wheels), and $\ddot{\omega}_j \approx \frac{\dot{\omega}_j(t+dt) - \dot{\omega}_j(t)}{dt}$, where dt is a short time interval.

The distances of a robot's neighbors are computed using the Law of Cosines, as described in Section 3.2.1. Storing the distance of every robot from each of its neighbors has a worst-case space complexity of $O(n^2)$, where n is the number of robots in the SRS. To reduce this complexity to $O(n)$, for each observed robot only two features, $f3$ and $f4$ are computed: the distance of the nearest

Algorithm 2 Compute distance $Dist[r_i, r_j, W_l]$ traversed by robot r_j during time window W_l, as observed by robot r_i

$\overrightarrow{P^j_{1_i}} = (d_{ij}\cos(\phi_{ij}), d_{ij}\sin(\phi_{ij}))$
$\omega_i = 0$
$\overrightarrow{v_i} = (0, 0)$
for (t = current to t = current $+W_l$) **do**
  $\omega_i = \omega_i + \Delta\omega_i$, where $\Delta\omega_i = \frac{-v_l + v_r}{h} \cdot \Delta t$, v_l and v_r are the speeds of the left and right wheels, respectively, and h is the distance between the wheels
  $\overrightarrow{v_i} = \overrightarrow{v_i} + (\frac{v_l + v_r}{2} \cdot \Delta t \cdot \cos(\omega_i + \frac{\Delta\omega_i}{2}), \frac{v_l + v_r}{2} \cdot \Delta t \cdot \sin(\omega_i + \frac{\Delta\omega_i}{2}))$
end for
$\overrightarrow{P} = (d_{ij}\cos\phi_{ij}, d_{ij}\sin\phi_{ij})$
$\overrightarrow{P^j_{2_i}} = (P_x\cos(\omega_i) - P_y\sin(\omega_i), P_x\sin(\omega_i) + P_y\cos(\omega_i)) + \overrightarrow{v_i}$
return $((P^j_{1_i x} - P^j_{2_i x})^2 + (P^j_{1_i y} - P^j_{2_i y})^2)^{\frac{1}{2}}$

neighbor and the average distance of the neighbors, respectively. Note that these numerical features are used to compute binary features $F1$, $F2$, $F5$, and $F6$.

Feature $f5$ represents the distance traveled by each robot in the last W_l seconds and is computed using Algorithm 2. Note that this feature is used to compute binary features $F3$, $F4$, $F5$, and $F6$.

Differently from the binary features, it is not feasible for the robots of the SRS to share, compare, and vote on these numerical features. In fact, in our implementation, a numerical feature occupies 32 bits in single precision format. Thus, a message with the numerical features of a robot contains at least $32 \cdot 5 = 160$ bits of data, which can exceed the bandwidth available in most SRSs. For instance, the method of [24] limits the messages to 16 bits.

We argue that using less processed features, such as the numerical ones, can leave more freedom to the classification methods. For instance, suppose the machine learning methods with numerical features implicitly compute a threshold on $f3$, as for $F1$. Now suppose changing the environment where the SRS moves: in the case of binary features, domain knowledge about the new environment and the behavior of the robots in it is needed to decide if the range $[0, 15]$ cm is still good or if it should be changed. On the other hand, such domain knowledge is not needed when using machine learning methods and numerical features: the machine learning method automatically learns the best threshold for fault detection. Experimental results of Section 5 confirm this intuition. This represents one of the main advancements we provide w.r.t. the method of [24].

3.3 Phase B: Classify

In this phase, each robot classifies the other robots as either faulty or not based on the features computed as per Section 3.2. To classify robot r_j, robot r_i must have obtained k sets of features about r_j in the last W_c seconds. The machine learning classifier receives in input the k sets of features and return a label that classifies r_j as faulty or non-faulty. In this way, r_i not only considers the last observation for classification, but exploits data about the evolution of r_j's behavior during the W_c time window. This phase is completely different from the corresponding phase of [24], which is based on a CRM (see Section 2).

3.3.1 Machine Learning Classifiers We consider two supervised machine learning methods for our fault detection approach: a logistic regressors and Gradient Boosting Decision Trees (GBDT) [9]. The logistic regressor computes a linear combination of the features, then applies a sigmoid function to the result: the output is the probability that the observed robot is faulty [9]. GBDTs are an ensemble method composed of weak decision trees. The term "weak" means that the decision trees have low individual performance. The trees are ordered: the first one predicts the problem's solution; the subsequent ones predict the error made by the preceding ones. The learning process is iterative, the first weak result gets subsequent refinements to become, ideally, more and more correct [9].

In both cases, the classification is obtained by applying a threshold p_{thresh} to the output probability: if the probability is above the threshold, the observed robot is classified as faulty, non-faulty otherwise.

The choice of these two machine learning methods aims to give two very different options for the proposed fault detection approach. Indeed, they use completely different procedures for the classification task, giving them distinct strengths. The logistic regressor requires very few resources for training and inference. The GBDT, on the other hand, thanks to its ensemble nature, represents a very flexible alternative. In fact, depending on the computational capability of the SRS platforms considered, the number of trees and their depth can be customized to obtain the best trade-off between classification accuracy and computational complexity. Note that, for both methods, training can be performed off-line while inference will be performed onboard the robots.

3.3.2 Training Dataset Preparation The training datasets for the machine learning methods are generated in controlled experiments: the SRS, composed of the union of a set of non-faulty robots and a set S_{faulty} of faulty robots, executes a given task and phase A of the proposed fault detection algorithm. Given robot r_j, it is known if $r_j \in S_{faulty}$ and, in that case, its fault. At the end of the experiments, all the observations of the non-faulty robots are gathered, labeled, and added as data points to the training dataset. Every time robot r_i observed robot r_j k times in the time window of W_c seconds, a data point is added to the training dataset: it contains the k sets of features obtained from the k observations and a label indicating if $r_j \in S_{faulty}$. When numerical features are used, they are standardized to reduce the impact of their different scales [2]. Then, the dataset is balanced by random undersampling non-faulty data points, depending on the probability p_{faulty} of a robot to belong to S_{faulty}.

3.3.3 Model Evaluation and Training The model evaluation phase exploits the training dataset to choose the best classifier for fault detection in a given task; namely the best machine learning method with a specific combination of values of the hyperparameters. Recalling Section 3.3.1, this phase compares the logistic regressor and the GBDTs with different combinations of hyperparameters. It exploits the ten-fold cross-validation technique to obtain ten evaluations for each classifier [9]. The metric used for evaluation is the F-score, a nonlinear combination of precision and recall, computed as: $F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$. For each classifier, the mean of the ten evaluations is computed. The classifiers with the lowest means are discarded to have only four remaining. The evaluations of these remaining classifiers are compared with the Wilcoxon signed-rank test [28], a non-parametric statistical test. The obtained p-values are corrected with the Bonferroni-Holm method [11], necessary to maintain the desired level of significance α with multiple tests. When the number of statistical tests is high, the Bonferroni-Holm method can considerably reduce the number of results that refuse the null hypothesis. Thus, in model evaluation, the statistical tests are used only with four classifiers.

After this process, two scenarios are possible. In the first scenario, there is statistical evidence that one classifier is better than the others: in this case, it is chosen for fault detection in the given task. In the second scenario, there is a set of classifiers for which there is no statistical evidence to prove that one is better than the others. In this case, the most resource- and time-saving classifier in the set is chosen. In particular, the logistic regressor is preferred over the

GBDTs. If there are only GBDTs, the one with the lowest number of decisions is chosen. In a GBDT, the number of decisions is the product of the two hyperparameters, i.e., the number and depth of the trees.

The chosen classifier is then trained on the whole training dataset.

3.4 Phase C: Voting

In this phase, similar to the one of [24], each robot communicates to the other robots its individual classifications on the observed robots. Then, the SRS computes a collective classification on each of its robots. The individual classification of robot r_i on the observed robot r_j is computed by majority voting starting from the classifications of r_i on r_j made during phase B in the current cycle. For instance, suppose that during phase B, r_i classified r_j seven times as faulty and six times as non-faulty. At the beginning of phase C, the individual classification of r_i on r_j is faulty.

Each robot r_i maintains in memory two lists: L_i and L_c. L_i contains the robot's individual classifications and the ones it received. For each classification, r_i stores the observing robot, the observed robot, and the label (faulty or non-faulty). If a robot r_i receives a collective classification on r_j (illustrated below in this paragraph), it adds it to L_c and removes all the individual classifications of r_j from L_i. When a robot has 5 or more individual classifications from different robots on robot r_j, it generates a collective classification, obtained by a majority voting scheme on the individual classifications. Then, it adds the collective classification to L_c, sends it to the other robots, and removes all the individual classifications on r_j from L_i. This procedure exploits the distributed nature of the SRS to obtain robust classifications.

4 EXPERIMENTAL SETUP

We implement the proposed fault detection approach in ARGoS, a discrete-time multi-robot simulator [19]. The control cycles of each robot are executed every 0.1 s and the proposed fault detection algorithm is executed at each control cycle. In order to compare the results, we maintain the same experimental setup described in [24].

Environment and robots The environment is a square arena of side 3 m, delimited by walls. The SRS is composed of $n = 20$ e-pucks [18], equipped with eight infrared proximity sensors, two wheels with sensors and actuators, and RAB sensors and actuators. The infrared proximity sensors have a range of $[0, 10]$ cm and an additive noise with uniform distribution between -1 cm and +1 cm.
The RAB sensors have a range of [0,100] cm. For a more realistic simulation, a noise vector with module sampled from $\mathcal{N}(0, 1)$ cm and angle uniformly sampled between $-\pi$ and $+\pi$ is added to each RAB reading. The wheels sensors detect velocity and have an additive uniform noise within -0.1 cm/s and +0.1 cm/s, while a noise with distribution $\mathcal{N}(0, 0.1)$ cm/s is added to the wheels actuators.

Behaviors We consider four different behaviors (tasks) of the SRS: *aggregation*, *dispersion*, *flocking*, and *homing*.

In aggregation, the robots use the RAB sensors to detect other robots, then move toward them. The main objective of this behavior is to bring all the robots of the SRS close to each other.

In dispersion, the robots move randomly: at each control cycle, they go straight with a certain probability; with a complementary probability, they change their heading to a certain angle, sampled from a uniform distribution between $-\pi$ and $+\pi$. The main objective of this behavior is to make the robots move in the environment so that the SRS occupies the entire available space.

In flocking, each robot uses the RAB sensors to detect the other robots' speeds and headings. Then, it moves toward the other robots while trying to uniform its speed and heading to the ones of the other robots. The main objective of this behavior is to make the entire SRS move in formation.

In homing, the robots use the RAB sensors to receive messages from the beacon, a non-moving robot. When they receive the messages, they use the RAB sensors to localize the beacon and move toward it. While they do not receive messages from the beacon, the robots follow the dispersion behavior. The main objective of this behavior is to aggregate all the robots near the beacon.

In all the behaviors, the robots use the infrared proximity sensors to avoid collisions.

Faults The proposed fault detection approach is tested with seven different faults: *bact*, *lact*, *ract*, *pmax*, *pmin*, *prnd*, and *rofs*.

lact, *ract*, and *bact* stop the movement of the left wheel, the right wheel, and both wheels, respectively.

pmin makes all the proximity sensors of the faulty robot always signal that there are no obstacles nearby. *pmax* makes all the proximity sensors of the faulty robot always signal that the robot is colliding. *prnd* makes all the proximity sensors of the faulty robot return values sampled from a uniform distribution in the range of the proximity sensor.

rofs adds a value sampled from a uniform distribution in the range $[75, 100]$ cm to each RAB range measurement and a value sampled from a uniform distribution in the range $[-\pi, +\pi]$ to each RAB bearing measurement. Our robots use the RAB board both as a sensor and an actuator for the swarm behaviors and as a communication mean for the fault detection approach. In our experiments, we consider faults in the RAB board affecting only the swarm behaviors. This is not a big limitation, because robots that are not able to transmit well-formed payloads (or don't communicate at all) can be easily recognized.

Parameters The values of the parameters used in the implementation of the proposed fault detection approach are the following: T_{cycle} = 10 s, T_{clas} = 9 s, T_{exp} = 600 s, W_l = 10 s, W_s = 5 s, W_c = 1 s, k = 10, p_{thresh} = 0.5, p_{faulty} = 20%, α = 0.05, and β = 2.

The values of the parameters used in the experimental setup are the following: n = 20, v_{max} = 5 cm/s, and h = 5.3 cm.

Training datasets We generate a training dataset for each combination of behavior and type of features (binary or numerical). We obtain it by running several simulations and capturing the robots' observations from control cycle 1000 to control cycle 1200, i.e., 100 s after the beginning of the simulation and for 20 s. In this way, we reduce the effects of the random initial positions of the robots.

Model evaluation and training For each combination of behavior and type of features, we use the corresponding training dataset to obtain a classifier through the model evaluation and training phase of Section 3.3.3. We use the F-score for model evaluation. The main objective of the fault detection approach is to detect the faults in the SRS: β must be greater than 1. On the other hand, if β is too

large, false alarms lose importance. Therefore, we implement the F-score with $\beta = 2$. For each combination of behavior and type of features, we report in Table 1 the classifier chosen in the model evaluation phase.

Experiments We run two sets of experiments. In the first set, as in [24], there is a single faulty robot in the SRS. For each combination of fault, behavior, and type of features, we run 30 experiments of 6000 control cycles (T_{exp} = 600 s), ignoring the first 450 cycles. In this way, we reduce the effects of the random initial positions of the robots. We compare these results with the ones reported in [24]. In the second set, each robot can be faulty with a probability of p. The values of p used in the experiments are $p = 0.1, 0.2, \ldots, 0.9$. The fault is chosen randomly (uniformly) from the set $\{bact, lact, ract, pmin, pmax, prnd, rofs\}$. For each combination of the values of p, behavior, and type of features, we run 30 experiments of 6000 control cycles (T_{exp} = 600 s), ignoring the first 450 cycles. We analyze these experiments' results to assess if the value of p influences the performance of the proposed fault detection approach. The results of this second set of experiments cannot be directly compared against [24] because it has not been shown to deal with multiple faulty robots.

5 EXPERIMENTAL RESULTS

We call CRM-B, ML-B, and ML-N the fault detection approach in [24], the proposed fault detection approach with binary features, and the proposed fault detection approach with numerical features, respectively. The performance of the proposed fault detection approaches is based on the F-score with $\beta = 2$ calculated on the collective classifications computed in phase C.

5.1 Single Fault Experiments

In Figure 1, we report the results of the experiments with a single faulty robot. Each subplot represents the results of the experiments on a behavior. The x-axis contain the different faults; the y-axis contain the F-scores with $\beta = 2$. For each combination of behavior and fault, there are three boxplots representing the results of the three fault detection approaches. The leftmost one, in blue, is relative to CRM-B, the central one, in orange, to ML-B, and the rightmost one, in green, to ML-N.

We compare these three sets of results using permutation statistical tests [7] to assess the presence of a statistically significant difference in their means. We correct the obtained p-values with the Bonferroni-Holm method so that the overall significance level for all the tests on a single behavior is α. We report the full results of these tests in the Supplementary Material. The analysis of the results highlights that ML-B and ML-N outperform CRM-B in 60.71% (17 over 28) and 71.43% (20 over 28) of the combinations, respectively. ML-B and ML-N have, in most cases, similar performance. Even when there is a statistically significant difference in their means, it is often marginal.

	Binary features	Numerical features
Aggregation	Logistic regressor	GBDT, 14 trees of depth 4
Dispersion	Logistic regressor	Logistic regressor
Flocking	GBDT, 4 trees of depth 4	Logistic regressor
Homing	GBDT, 4 trees of depth 4	GBDT, 12 trees of depth 4

Table 1: Chosen classifiers.

There are only a few cases in which these two approaches perform very differently: in all of them, ML-N outperforms ML-B. For instance, in the aggregation behavior, ML-N is nearly perfect in detecting faults in the wheels actuators, while ML-B struggles in some experiments, reaching F-score values lower than 0.2. For some combinations, some fault detection approaches have bad performance. We further inspect them.

Aggregation behavior with pmin fault. From a visual inspection of the simulations, the main difference between the faulty robot and the non-faulty robots is that the former does not keep a minimum security distance from the other robots. Indeed, its faulty proximity sensors never detect other robots; therefore, it never tries to avoid collisions with them. The binary features cannot capture the difference between two robots keeping the minimum security distance and two colliding robots: feature F1 captures if the nearest neighbor is in the range $[0, 15]$ cm, but this feature is true in both situations. On the other hand, the numerical feature $f3$ measuring the minimum distance of a neighbor can capture this difference. Hence, ML-N has significantly better results than the two fault detection approaches based on binary features.

Dispersion behavior with rofs fault. The dispersion behavior does not exploit the Range And Bearing (RAB) sensor. Therefore, the behaviors of faulty and non-faulty robots are the same, making it impossible to detect the fault.

Flocking behavior with pmin fault. In the flocking behavior, all the robots try to move in the same direction with the same speed. This behavior makes collisions less frequent, making the faulty robot almost unrecognizable, and all the three fault detection approaches perform poorly.

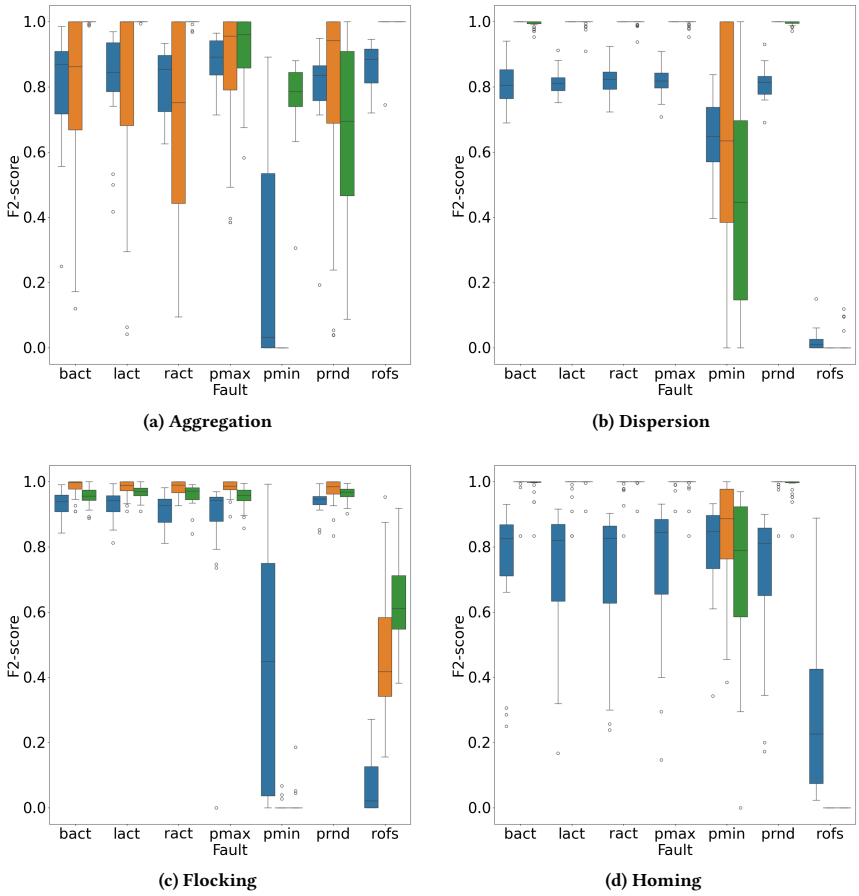
Homing behavior with rofs fault. When a robot does not receive the beacon message, it executes the dispersion behavior. Thus, with *rofs* fault, if the faulty robot does not receive the beacon message, the fault detection approaches perform as for the dispersion behavior. If the faulty robot receives the beacon message, the random offset of the RAB makes it move in a random direction, likely different from the correct one. From an external point of view, this random movement is is indistinguishable from the dispersion behavior of a robot that does not receive the beacon message. In both cases, it is hard for the fault detection approaches to detect the fault.

5.2 Multiple Faults Experiments

In Figure 2, we report the results of the experiments with multiple faulty robots. Each pair of plots represents the results of the experiments on a behavior and using ML-B (above) and ML-N (below). The x-axes contain the values of p, and the y-axes contain the F-scores with $\beta = 2$.

We run three sets of statistical tests:

I For each combination of behavior and type of features, we run permutation tests to assess the presence of a statistically significant difference in the means of the results with different values of p (the probability of a robot to be faulty).



(a) Aggregation (b) Dispersion

(c) Flocking (d) Homing

Figure 1: Experiments with a single faulty robot.

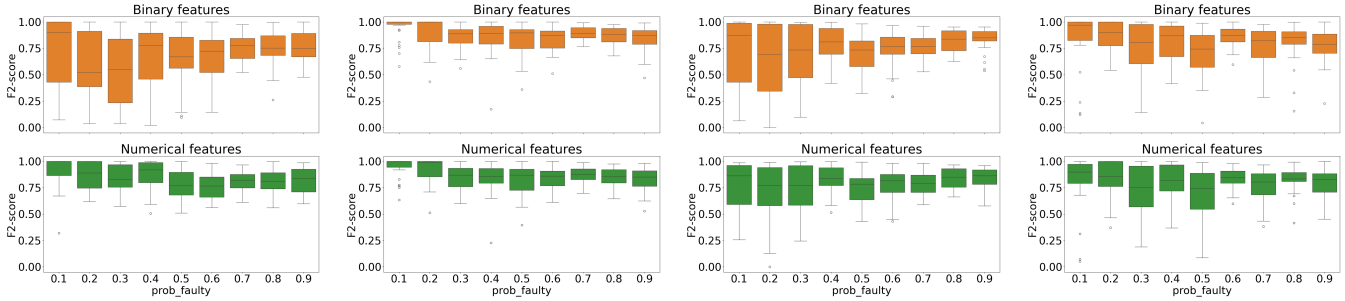
II For each combination of behavior and type of features, we run permutation tests to assess the presence of a statistically significant difference in the variances of the results with different values of p.

III For each combination of behavior and value of p, we run one-sided permutation tests to assess if ML-N obtains results with a higher mean than ML-B. The choice of one-sided tests stems from the results of single fault experiments and from visual inspection of the results in Figure 2. Indeed, they suggest that, in most cases, ML-N performs at least equally, if not better, than ML-B.

We correct the obtained p-values with the Bonferroni-Holm method so that, in sets I and II, the overall significance level for all the tests on a combination of behavior and type of features is α. In set III, the overall significance level for all the tests on a single behavior is α. We report the full results of these tests in the Supplementary Material.

The results on set I highlight that the value of p does not influence the mean of the performance of our fault detection approaches. The only exception is in the experiments on the dispersion behavior with $p = 0.1$, whose results are significantly higher than those with other values of p. By inspecting these experiments, we observe that, in 70% of them, the *pmin* and *rofs* faults are absent. When they are present, there are at least two other faults in other robots. Therefore, the effects of *pmin* and *rofs* faults on the results are marginal in these experiments. Both ML-B and ML-N have relatively bad results in single fault experiments with these two faults: thus, their marginal impact on these experiments explain why the results are significantly better than with other values of p.

The results on set II highlight that, in most cases, the value of p does not influence the variance of the performance of the fault detection approaches. There is a significant difference between the variance of the results with lower values of p and those with higher values of p in three combinations of behavior and type of features: aggregation with binary features and flocking with both binary and numerical features. We present the analysis for the first combination; the analysis of the other two is similar. The single fault experiments highlight that ML-B has low performance in detecting the *pmin* fault on aggregation behavior. When p is low, a small number of robots is faulty. If one of the faults is *pmin*, it considerably decreases the performance of the proposed fault detection approach. Since in the other experiments the performance is high, this increases the variability. When p is high, a large number of robots is faulty. Each faulty robot has a $\frac{1}{7}$ probability of having



(a) Aggregation (b) Dispersion (c) Flocking (d) Homing

Figure 2: Experiments with multiple faulty robots.

a *pmin* fault: the ratio of robots with *pmin* faults is low. Therefore, the *pmin* faults only marginally decrease the performance of the proposed fault detection approach and the variability remains low.

The results on set III highlight that ML-N significantly outperforms ML-B on aggregation and flocking behavior.

6 DISCUSSION

CRM-B [24] requires explicit domain knowledge to be designed correctly: the binary features are identified and thresholded in an ad hoc way by using information about the task, the robots, and the environment; and the setting of the parameters of the CRM [24, Table 2] requires deep knowledge about this model. On the other hand, the domain knowledge required by ML-N is minimal: the numerical features are simply derived from raw measurements of the robots, while the choice of the best machine learning methods and their hyperparameters occur in the model evaluation phase, which requires minimal human intervention. Indeed, this phase only requires the user to choose feasible combinations of hyperparameters. Also the parameters of the chosen machine learning method are computed automatically in the training phase. However, the model evaluation phase and the training require labeled data; on the contrary, the CRM does not require a training dataset. ML-B lies between CRM-B and ML-N: it still uses domain knowledge to identify the binary parameters and their thresholds, but it exploits data to automatically choose the best machine learning methods and their hyperparameters.

The statistical analysis of the results of the experiments highlights that the proposed fault detection approaches, both with binary and numerical features, often outperform CRM-B in the setup described in Section 4, when a single robot of the SRS is faulty. Thus, in this setup, it is possible to substitute domain knowledge with the knowledge automatically learned from data, keeping high performance. This is a promising result for real-world applications: given their complexity, eliciting domain knowledge could be hard. On the other hand, data can often be easily obtained by running controlled experiments. In addition, our approach can deal with multiple faulty robots.

However, further tests are necessary before the implementation of the proposed fault detection approach in real-world applications. The behaviors studied in this paper are simpler than those required in many real-world applications, where also combinations of simple behaviors could be needed. Also, results obtained in simulation do not necessarily transfer to the real world [10].

We implemented and tested the proposed fault detection approach only with two supervised machine learning methods. We speculate that further studies involving other machine learning methods could improve the flexibility of the proposed approach.

It is possible to easily extend the proposed fault detection approach to perform both exogenous and endogenous fault detection: the only requirement is that each robot can measure its own features. This extension could improve or worsen the performance of the fault detection approach, depending on the setup. Indeed, while it adds a new and unique point of view on each robot, it could also give some incorrect information if the robot observing itself has faulty sensors. For instance, a faulty robot with broken speed sensors and actuators could detect it is moving and classify itself as non-faulty, even if it is not actually moving.

7 CONCLUSION

In this paper, we proposed a novel distributed fault detection approach for SRSs, which exploits machine learning methods to classify a robot as faulty or non-faulty. Also, we proposed new numerical features, more primitive than the binary features used in [24] to leave more freedom to the machine learning methods. The proposed approach relies on data to select the best machine learning method (and its hyperparameters) and to train it. This reduces the amount of required explicit domain knowledge. Also the processing of the measurements to obtain numerical features requires minimal domain knowledge. The statistical analysis of the results, exploiting permutation tests, shows that the proposed fault detection approach outperforms the one described in [24] in several settings and that its performance remains robust also for multiple faulty robots in the SRS. Also, the performance of the proposed approach improves when using numerical features instead of binary features.

Acknowledgments

This paper is supported by PNRR-PE-AI FAIR project funded by the NextGeneration EU program.

References

[1] A. Barua and K. Khorasani. 2011. Hierarchical Fault Diagnosis and Fuzzy Rule-Based Reasoning for Satellites Formation Flight. *IEEE Trans. Aerosp. Electron.*

Syst. 47, 4 (2011), 2435–2456.
[2] C. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.
[3] J. Bjerknes and A. Winfield. 2013. On fault tolerance and scalability of swarm robotic systems. In *Proc. DARS*. 431–444.
[4] D. Bossens, S. Ramchurn, and D. Tarapore. 2022. Resilient Robot Teams: a Review Integrating Decentralised Control, Change-Detection, and Learning. *Curr. Robot. Rep.* 3, 3 (2022), 85–95.
[5] A. L. Christensen, R. O'Grady, and M. Dorigo. 2009. From Fireflies to Fault-Tolerant Swarms of Robots. *IEEE Trans. Evol. Comput.* 13, 4 (2009), 754–766.
[6] G. Deng, Y. Zhou, Y. Xu, T. Zhang, and Y. Liu. 2021. An investigation of byzantine threats in multi-robot systems. In *Proc. RAID*. 17–32.
[7] P. Good. 2005. *Permutation, Parametric and Bootstrap Tests of Hypotheses* (3 ed.). Springer.
[8] H. Hamann. 2018. *Swarm Robotics: A Formal Approach.* Springer.
[9] T. Hastie, R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.
[10] S. Höfer, K. Bekris, A. Handa, J. Gamboa Higuera, F. Golemo, M. Mozifian, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Liu, J. Peters, S. Song, P. Welinder, and M. White. 2020. Perspectives on Sim2Real Transfer for Robotics: A Summary of the R:SS 2020 Workshop. *CoRR* abs/2012.03806 (2020). https://arxiv.org/abs/2012.03806
[11] S. Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scand. J. Stat.* 6, 2 (1979), 65–70.
[12] A. Khadidos, R. Crowder, and P. Chappell. 2015. Exogenous Fault Detection and Recovery for Swarm Robotics. In *Proc. INCOM*. 2405–2410.
[13] E. Khalastchi and M. Kalech. 2018. On fault detection and diagnosis in robotic systems. *ACM Comput. Surv.* 51, 1 (2018), 1–24.
[14] E. Khalastchi and M. Kalech. 2019. Fault detection and diagnosis in multi-robot systems: A survey. *Sensors* 19, 18 (2019), 4019.
[15] H. Lau, I. Bate, P. Cairns, and J. Timmis. 2011. Adaptive data-driven error detection in swarm robotics with statistical classifiers. *Rob. Auton. Syst.* 59, 12 (2011), 1021–1035.
[16] S. Lee, E. Milner, and S. Hauert. 2022. A data-driven method for metric extraction to detect faults in robot swarms. *IEEE Robot. Autom. Lett.* 7, 4 (2022), 10746–10753.
[17] A. Millard, J. Timmis, and A. Winfield. 2013. Towards exogenous fault detection in swarm robotic systems. In *Proc. TAROS*. 429–430.
[18] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. 2009. The e-puck, a Robot Designed for Education in Engineering. In *Proc. ICARSC*. 59–65.
[19] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. Gambardella, and M. Dorigo. 2012. ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems. *Swarm Intell.* 6, 4 (2012), 271–295.
[20] L. Qin, X. He, and D. Zhou. 2014. A survey of fault diagnosis for swarm systems. *Syst. Sci. Control. Eng.* 2, 1 (2014), 13–23.
[21] E. Şahin. 2005. Swarm Robotics: From Sources of Inspiration to Domains of Application. In *Proc. SAB*. 10–20.
[22] M. Soori, B. Arezoo, and R. Dastres. 2023. Artificial intelligence, machine learning and deep learning in advanced robotics, A review. *Cognitive Robotics* (2023), 54–70.
[23] D. Tarapore, A. Christensen, and J. Timmis. 2015. Abnormality detection in robots exhibiting composite swarm behaviours. In *Proc. ECAL*. 406–413.
[24] D. Tarapore, A. Christensen, and J. Timmis. 2017. Generic, scalable and decentralized fault detection for robot swarms. *PLOS ONE* 12, 8 (2017), 1–29.
[25] D. Tarapore, R. Groß, and K.-P. Zauner. 2020. Sparse robot swarms: moving swarms to real-world applications. *Front. Robot. AI* 7, 83 (2020), 1–9.
[26] D. Tarapore, P. Lima, J. Carneiro, and A. Christensen. 2015. To err is robotic, to tolerate immunological: fault detection in multirobot systems. *Bioinspir. Biomim.* 10, 1 (2015), 1–43.
[27] D. Tarapore, J. Timmis, and A. L. Christensen. 2019. Fault Detection in a Swarm of Physical Robots Based on Behavioral Outlier Detection. *IEEE Trans. Robot.* 35, 6 (2019), 1516–1522.
[28] F. Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.
[29] A. Winfield and J. Nembrini. 2006. Safety in numbers: fault-tolerance in robot swarms. *Int. J. Model. Identif. Control.* 1, 1 (2006), 30–37.