# Anomaly Detection in Swarm Robotics with Security Constraints

Daniel Cronce, Dr. Andrew Williams, Dr. Debbie Perouli

***Abstract*—A common problem in swarm robotics is fault detection. Many solutions rely on internal diagnostics to diagnose these problems and, therefore, inherently trusts information from all members. However, our research adds the possibility of hacked swarm members. To accommodate this new constraint, we must accurately obtain diagnostics exogenously. In this paper, we propose an external diagnostics system, a simple monitoring algorithm, and a rule-based anomaly detection algorithm.**

## I. Introduction

Fault detection in swarm robotics is still a large issue despite both active research and fault tolerance of swarms. While many papers show progress in this area, they generally suffer specific weaknesses. These include not taking into account security of the systems or having such general fault detection that only faults in the detection modules or a complete robot failure can be detected. This research introduces security constraints which prevent robots from directly trusting information from the suspect. This research does not significantly expand on the granularization of fault or anomaly detection.

### A. Background

Anomaly detection is relative to the properties of the algorithm being monitored. In this case, we are using the Boid flocking algorithm [5]. This algorithm attempts to simulate flocking by summing velocity vectors derived from the positioning of other Boids within a neighborhood, or radius. It follows that our anomaly detection will be centered around determining whether a Boid's positioning is anomalous. As an additional factor, small imperfections in real-life robotics will naturally add noise and cause discrepancies between the sensors and actuators. The final area of concern is how we can gather information. Since we cannot trust the health status and positional information sent from the suspect, we must find a way to acquire this information externally.

## II. Rule-based Anomaly Detection

Generally, anomaly detection is determining whether a behavior is outside of a norm. To do this, we need to define the norm using information indicative of normal function. Then, we need to specify a threshold, so small, circumstantial value errors are not anomalous. Here we define three rules:

- If the suspect broadcasts false information
- If the suspect is in the incorrect position
- If the suspects stops communicating

However, we need to know the suspect's current position, the position it should be in, the last time it communicated, and the information it communicates to use these rules. The information it communicates and when it communicates is readily available, but where it actually is and where it should be requires information from external sources.

## III. Using Wi-Fi to Trilaterate Positions

In order to guarantee accurate positioning, we needed to create a shared frame of reference for all robots in the swarm. Following Khadidos et al. [1], we used signal strength of the robots' communication to judge the distance from them and the receiver. We created a service that monitored all packets, recorded the MAC addresses and signal strength, and responded to requests with the average distance within a time-frame.

### A. Calculating Distance

Electromagnetic signals in transit experience a phenomenon known as path loss [6]. Path loss for our problem is simply the loss in signal strength over distance. Using the signal strength gathered from the packets, we rearrange the Free Space Path Loss formula to calculate the distance:

$$d = 10^{((27.55-(20*log_{10}(f))+|s|))/20)}$$

Where $f$ is the frequency in megahertz, $s$ is the signal strength in decibels, and result $d$ is in meters.

### B. Determining Initial Coordinates

The last component of the trilateration service is the coordinates of each server. Since the Cartesian place only had to be relative to the servers in question, we could pick the initial coordinates and orientation.

The algorithm runs as follows: first, the servers discover each other by looking for Robot Operating System (ROS) services within the multi-master system; second, they all request distance between each other; third, they sort all the servers alphabetically; fourth, the first server becomes (0,0), the second becomes ($a$, 0), where $a$ is the distance between the first and second server; and finally, the third server's position is calculated using the three known distances and two known coordinates of the triangle.

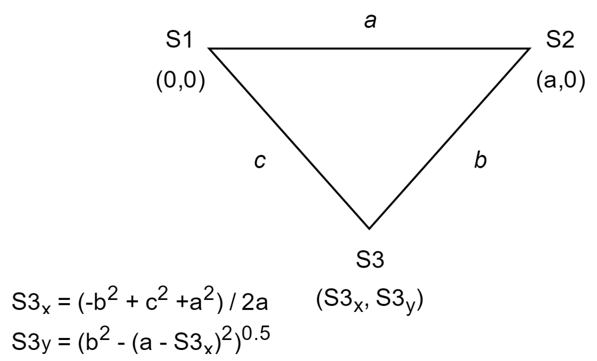

Fig. 1. Representation of the triangle used to build the frame of reference.

**Algorithm 1** Building the Cartesian Plane

```
$a = distance(sortedServs[0], sortedServs[1])$
$b = distance(sortedServs[1], sortedServs[2])$
$c = distance(sortedServs[0], sortedServs[2])$

$x_3 = (-b^2 + c^2 + a^2)/2a$
$y_3 = sqrt(b^2 - (a - x_3)^2)$

$myInd = sortedServs.index(me)$
$servCoords[0] = (0, 0)$
$servCoords[1] = (a, 0)$
$servCoords[2] = (x_3, y_3)$

$myCoords = servCoords[myInd]$
```

## IV. Simulating Flocking of the Suspect

To calculate where the suspect should be, we need to run their inputs through the flocking algorithm. We gather its inputs by simply taking all robots within its neighborhood. Then, we continuously input the information into a separate Boid. From here, we can determine discrepancies between the actual location and the location the suspect should be.

## V. Suspicion Algorithm

The suspicion algorithm uses threshold-based rules. If any discrepancy passes the respective threshold, a series of events fires. First, the robot broadcasts its suspicion. Then, a designated, third-party confirmer runs the same algorithm. If this confirmer also broadcasts suspicion, the suspect is declared anomalous.

### *A. Selecting a Suspect and Who to Confirm For*

To prevent every member from having to monitor every other member, we needed a selection algorithm that fulfilled the following constraints:

- A member cannot monitor or confirm for itself
- A member cannot monitor or confirm the same robot twice in a row
- A member cannot monitor a robot that another member is already monitoring
- A member cannot confirm for a robot that another member is already confirming for
- A member cannot monitor a member it's confirming for and vice versa

Despite the list of constraints, the final algorithm is relatively simple. First, the robot pulls all hosts using the swarm's topic and sort them by name. If this is the first time i.e. we haven't had a suspect, then the robot finds its own position in the array. The position plus one is the suspect, and the position plus two is the host the robot will confirm for. Otherwise, we use the positions of the last robots that were our suspect/confirmee and add one to each of their positions.

**Algorithm 2** Selection of Suspect and Confirmee

```
**if** $firstTime$ **then**
    $suspect = myPos + 1 \bmod numberOfRobots$
    $confirmFor = myPos + 2 \bmod numberOfRobots$
**else**
    $suspect = suspect + 1 \bmod numberOfRobots$
    $confirmFor = confirmFor + 1 \bmod numberOfRobots$
**end if**
```

## VI. Experiment

The physical experimentation has yet to be performed. It is planned that an empty room with three, dedicated Wi-Fi-trilateration servers set up along the walls. Their wired interface would be connected to an access point (AP). When the service is run, it will require information to connect to the AP's wireless. A robotic swarm of no less than three, nonholonomic robots running the Robot Operation System will be subjects. The first experiment will be the control group, and no anomalies will be injected. The next experiments will be all possible combinations of the detection rules. All of these experiments should run for at least

$$(numRobots + 1) * monitorPeriod$$

seconds where $monitorPeriod$ is the duration in which the robots monitor their suspects.

## VII. Related Work

Khadidos et al. [1] used a similar fault detection method as well as using signal strength to estimate distance. The most significant difference is the inclusion of security constraints. While their experiments trusted information from the possibly-anomalous robot, the possibility of that robot being hacked constrains how we procure our information and the extent to which we can diagnose the fault.

Millard el al. [4] also used a simulation and periodic reinitialization approach for detecting anomalies in their robots. However, our detection system works at a higher level and uses a selection algorithm to cut the complexity from O($n^2$) to O($n$).

## Acknowledgment

The authors would like to thank the Humanoid Engineering & Intelligent Robotics (HEIR) Lab for their equipment and support and our cohorts for their assistance.

This work was supported in part by a National Science Foundation REU Site grant, #ACI-1461264, 'Computation Across the Disciplines,' and NSF National Robotics Initiative grant #1427399 at Marquette University.

## References

[1] Adil Khadidos, Richard M. Crowder, Paul H. Chappell *Exogenous Fault Detection and Recovery for Swarm Robotics*

[2] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, Marco Dorigo *Exogenous Fault Detection in a Collective Robotic Task*

[3] Anders Lyhne Christensen, Rehan O'Grady, Marco Dorigo *From Fireflies to Fault-Tolerant Swarms of Robots*

[4] Alan G. Millard, Jon Timmis, Alan F.T. Winfield *Run-time Detection of Faults in Autonomous Mobile Robots Based on the Comparison of Simulated and Real Robot Behaviour*

[5] Craig W. Reynolds *Flocks, Herds, and Schools: A Distributed Behavioral Model*

[6] Nation Telecommunications & Information Administration *Path Loss* http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm