Endogenous Fault Detection for Circle Following Behavior

Andrew Euredjian
Robotics Department
Worcester Polytechnic Institute
Worcester, MA 01609
Email: ageuredjian@wpi.edu

Harrison Saperstein
Robotics Department
Worcester Polytechnic Institute
Worcester, MA 01609
Email: hksaperstein@wpi.edu

Revant Mahajan
Robotics Department
Worcester Polytechnic Institute
Worcester, MA 01609
Email: rmahajan@wpi.edu

***Abstract*—Robots in a swarm can achieve complex objectives by splitting them into smaller tasks and achieving these sub tasks with decentralized coordination. This decentralized behavior is responsible for the inherent robustness and scalability of swarm systems. But swarm systems are not devoid of faults. Like all robot systems, they can fail. In order to realize the full robustness potential, the swarm systems need to be able to detect and isolate faults within themselves. In this paper, we implemented an endogenous fault detection strategy on a swarm of robots exhibiting a circle following behavior.**

I. Introduction

Swarm Intelligence is very prevalent in nature. Bees, ants, and birds are all known to show this behavior. In fact, the field of swarm robotics is inspired from nature and these animals. Despite their small size, these animals are able to accomplish some impressive and intelligent behaviors. They are able to overcome their limited individual capabilities by acting as a collective. The field of swarm robotics aims to draw inspiration from this collective cooperation and implement it for a system of robots. This way, robotic systems would be able to overcome the challenge of limited capability of a single robot to perform complex tasks. This would help in driving down costs for these robots and would also help to formulate solutions in an intrinsically scalable manner.

In the recent decade, swarm systems have come a long way in terms of the complexity of tasks they are being considered for. The most popular example nowadays is using a swarm of drones to display specific patterns. Some of the other use cases being researched are for the military, search and rescue, surveillance, agriculture, etc [1]. These complex use cases have a very strong need for robust systems. Swarm systems are inherently robust and scalable due to the decentralized nature of control. In swarm systems, global patterns or behaviors emerge from local interactions. The quality of the local interactions is thus critical for the effectiveness of the swarm systems. If the quality of these local interactions degrades over time, due to internal or external faults, the system can depict significant aberrations. Common examples of these faults include sensor failures, communication failures, software bugs, etc. These faults are very common in robotic systems and thus any swarm robotic system should be expected to depict some of them eventually. Thus, in order to realize the full robustness potential of swarm systems, there is a general need for good fault detection and isolation algorithms.

In this project, we implemented a simple circle following behavior [2] in simulation and implemented a simple endogenous fault detector to detect faults for this behavior.

II. Background

A. Fault Detection

Fault detection for swarm systems can be classified as endogenous or exogenous [3]. Endogenous fault detection refers to the act of robots detecting faults within themselves. This fault detection scheme has been shown to detect faults such as broken sensors or actuators [4]. While relatively simpler to implement as compared to the exogenous counterparts, this fault detection scheme cannot detect catastrophic faults such as power source failure, computational hardware failure, etc. On the other hand, exogenous fault detection refers to the act of robots detecting faults in other robots of the swarm system. An exogenous fault detection scheme can potentially detect any kind of faults, including catastrophic faults. This approach has certain limitations for when a robot of the swarm is performing a task far away from other robots. In this case, the other robots of the swarm might not necessarily be able to detect failures because of communication restrictions or other limitations. For such a case, exogenous and endogenous fault detection schemes can be combined and this combined approach is referred to as multi-layered fault detection.

One method for performing exogenous fault detection is using a Cross Regulation Model as introduced by [3]. This method was inspired by the autoimmune response to foreign and domestic antigen present in the body. Each robot observes its neighbor's behaviors and determines their abnormality over a 9 second period. Within these 9 seconds, the observations are calculated 90 times and the decision to mark the robot as normal vs abnormal is determined by the state that was calculated the most in the 90 cycles. After the 9 seconds, the robots share their observations with the remainder of the swarm and a voting coalition is formed within the swarm to then vote on the abnormality of the observed robots behaviors.

Another method that has been of interest is model-based detection. Model-based detection consist of analyzing the *residual* deviation between the expected model behavior and that of the actual behavior of the system [5]. System noise can cause a substantial effect on the *residual*, creating motivation for techniques that include artificial neural networks for this method of fault detection [6]. Kalman filters [7] and particle filters [8] have also been implemented to perform multi-model fault detection with robustness against noise in the system. The major challenges with these methods is the computational requirement and the need to reduce the cost to calculate the *residual* for fault detection.

B. Behavioral Patterns

In order to test the effectiveness of fault detection algorithms, researchers developed various behaviors for robotic swarms to perform. These behaviors are typically classified into two categories: homogeneous behaviors and heterogeneous behaviors. Homogeneous behaviors are those that all robots in a swarm perform at the same time, whereas heterogeneous behaviors are defined as groups of robots within a swarm performing different tasks as part of an overall objective.

Much of the research in swarm robotics focuses on performing homogeneous behaviors such as dispersion, aggregation, flocking and homing [3] [9]. In dispersion, robots will move in the opposite direction of the center of mass of their neighbors. Aggregation is the opposite of dispersion where robots move towards the center of mass of their neighbors. Homing is similar to aggregation where robots will move towards a beacon. Lastly, flocking deals with robots that adjust their velocities in accordance with their neighbors [3]. Though these examples make up much of the basis of homogeneous behaviors, they are not the only examples. [2] discusses how a swarm of robots can create and maintain a circular movement pattern with a front-facing binary proximity sensor.

An example of a heterogeneous behavior is foraging which involves multiple tasks such as cooperative exploration, navigation, resource localization, and recruitment [3]. With foraging, the swarm will delegate tasks for groups of robots to perform based on the status of the environment and the current goal of the swarm. As the swarm's goal or environment changes, the sizes of groups may also change to compensate or comply with the changing requirements. Continuing with the example of foraging, an environment that has a resource pool far away from the swarm's nest will have more explorer robots and less transport robots than an environment in which the resource pool is close to the nest [3]. Due to the increased complexity of a heterogeneously behaving swarm, the robots require a wider variety of sensors that are more sophisticated than those used by homogeneously behaving robots [10].

III. Methodology

Our project was split into two phases: 1) Implementation of the circle following behavior and 2) Implementation of the endogenous fault detection algorithm.

A. Circle Behavior

We implemented a simple non-computational circle formation algorithm from [2] on a differential drive robot (Figure 1).

d	α	V_L^0	V_R^0	V_L^1	V_R^1
1.0m	0 deg	9.0	1.0	3.0	9.0

TABLE I: Parameters for the Algorithm 1

This algorithm does not require communication between robots but only depends on a binary input from the front proximity sensor. The proximity sensor returns $S = 1$ if there is a robot within the proximity sensor's range and aperture as shown in Figure 2. It returns $S = 0$ otherwise. The turning behavior based on the proximity input S is shown in Figure 3. Based on the value of S and the behavior in Figure 3, V_L and V_R are set. Algorithm 1 describes this approach. The values for V_L and V_R are shown in Table I

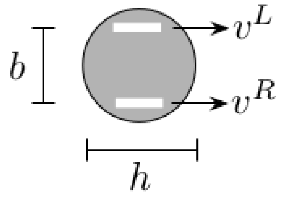


Fig. 1: Differential drive robot

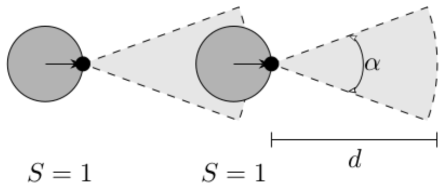


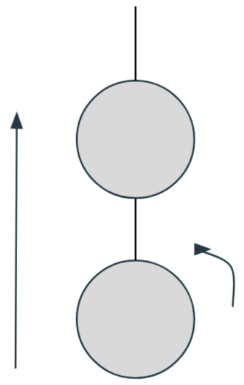
Fig. 2: Aperture (α) and distance (d) of the proximity sensor

Algorithm 1 Control algorithm

```
1: if $S == 1$ then
2:     set wheels speed ($v_1^L$, $v_1^R$)
3: else
4:     set wheels speed ($v_0^L$, $v_0^R$)
5: end if
```

We made a slight modification to the proximity sensor parameters. We used $d = 1m$ for the proximity sensor's range instead of the optimal value $d = 3m$. This is based of the observation made in the paper that $d = 1m$ gave more uniform detection range across robots. We were also constrained by the placement of the proximity sensors on the Khepera IV which affected our aperture size. We could only use the front sensor as shown in Figure 4. The adjacent sensors gave us a wider view than we wanted to use.

Using Algorithm 1 and the values mention in Table I, we were able to consistently achieve a good circle formation behavior (Figure 5


(a) Robot turns CCW when $S = 1$

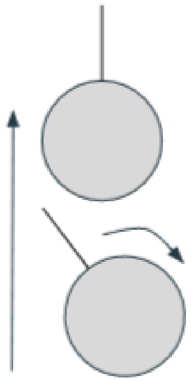

(b) Robot turns CW when $S = 0$

Fig. 3: Robot behavior based on proximity input

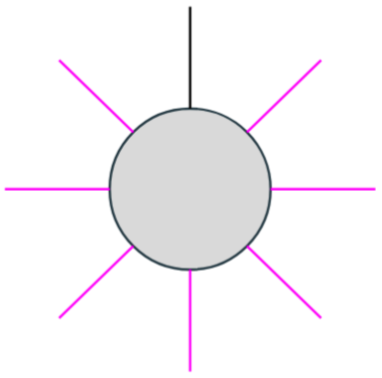
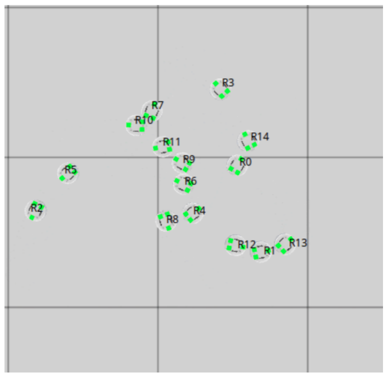
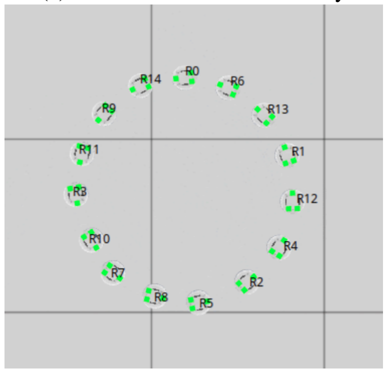


Fig. 4: Proximity sensor being used represented by black and the ones not used represented by pink



(a) Robots initialized randomly



(b) Robot form a circle after some time

Fig. 5: Circle formation using Algorithm 1

B. Fault Detection Algorithm

Initially, our team planned to use a collective polling strategy inspired from [3]. However, due to time constraints, we were unable to implement it and decided to instead develop our own simple fault detection algorithm. In our algorithm, the robots are classified as $faulty$ and $non-faulty$. There is one robot in the swarm that is randomly chosen to be classified as $faulty$. Initially, all robots are represented with green LEDs, as $non-faulty$. Once the fault is introduced into the swarm, the randomly selected faulty robot's LEDs changed to blue to indicate the fault has been successfully induced. The robot that identifies itself as faulty after the fault has been introduced then changes its LEDs to red to indicate a self-fault. Note that the robot that identifies itself as faulty may not be the robot randomly chosen to fault, resulting in a false positive. This is due to the nature of our algorithm. Using the range-and-bearing sensors on the robots, we create a simulated field of view (FOV) in front of and behind the robot. The FOV is defined by specifying a range $distance$ and angle from the center line $alpha$ shown in Figure 6, resulting in a cone-shaped area that the robot uses to identify close neighbors as seen in Figure 7. After a circle formation is established, the robots begin checking for the closest neighboring robots within their front and rear FOVs. The robots only start monitoring front and rear close neighbors after a circle formation is created since the process of creating the circle formation involves many robots not having close front or rear neighbors at times. While monitoring close neighbors in the circle formation, if a robot were to lose sight of both its front and rear neighbors, it would identify itself as faulty. The robot then broadcasts a message to all its neighbors close or far that it has faulted. Upon receiving the fault message, the neighbors relay the message to all their neighbors such that the message cascades throughout the swarm and reaches every robot. A robot that identifies itself as faulty or receives a fault message will also stop moving such that eventually all robots in the swarm stop moving when a fault is detected.

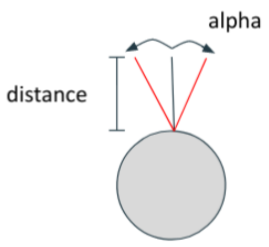


Fig. 6: Visualization of FOV defined by *distance* and *alpha*

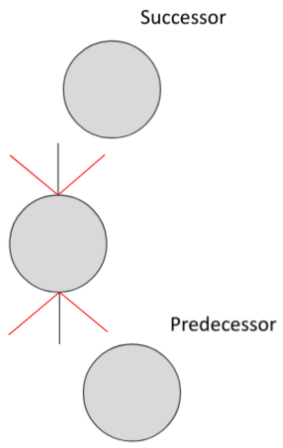


Fig. 7: Visualization of front and rear neighbors in range of FOV

IV. Experiments

To conduct experiments, our team used the Argos simulator with the Buzz language. We also created loop functions in C++ for the purpose of collecting data and selecting a random robot in the swarm to fault. The loop functions were also used to select random wheel speeds for the faulty robot to use when the fault is induced in the swarm. The robot we used in our simulation was the Khepera IV as seen in Figure 8. It features 8 proximity sensors located around its periphery, range-and-bearing sensors for getting the locations of neighboring robots, wireless communication protocols to other robots, and a 2-wheel differential drivetrain.

Our algorithm's performance is evaluated based on the correct detection of the faulty robots and the number of false positives it produces based on our experiments. An experiment starts with robots initialized in random positions in the simulated arena. They then proceed to perform the circle formation behavior. After an experimentally determined number of time steps after which a circle is formed, we introduce a fault randomly in one of the robots in the swarm by changing the V_L and V_R to a random integer in the range [-6,6]. At the same time, our fault detection algorithm is enabled and begins monitoring for faults in the swarm. Once the algorithm identifies a fault, it broadcasts a fault detected message to all robots in the swarm with the ID of the faulty robot. Every robot upon receiving the message then stops its movement and relays the message to its neighbors such that all robots in the swarm eventually stop moving. We performed a total of 105 experiments with different starting parameters to evaluate the algorithm's performance. 90 of the experiments ran were considered normal experiments with 9 different combinations of starting parameters with 10 runs each. 15 experiments were considered to be special case experiments with 3 different combinations with 5 runs each. The starting parameters that were varied in the experiments included the number of robots in the swarm R, the range threshold for the front and rear FOV *distance*, and the aperture of the FOV α. In the normal experiments (Table II), *distance* was kept constant at 45 centimeters while R and α were varied between 15, 20, and 25 robots, and 15, 20, and 30 degrees respectively. Note that an α value of 15 degrees corresponds to a total FOV of 30 degrees. For the special case experiments (Table III), R and α were kept constant at 50 robots and 30 degrees respectively while *distance* was varied between 45, 50, and 55 centimeters. In addition to the varying parameters, we assume that the errors encountered in a faulty robot are only introduced in the actuators. Therefore, the sensors and communications with other robots are undisturbed and will not encounter any issues in the faulty robot during our experiments.

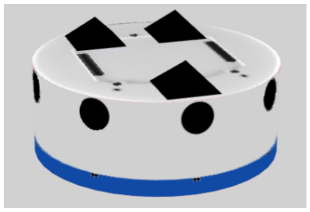


Fig. 8: The Khepera IV robot in the Argos simulator

R	*alpha* (degrees)	*distance* (cm)
15	15	45
15	20	45
15	30	45
20	15	45
20	20	45
20	30	45
25	15	45
25	20	45
25	30	45

TABLE II: Starting parameters for normal experiments

R	$alpha$ (degrees)	$distance$ (cm)
50	30	45
50	30	50
50	30	55

TABLE III: Starting parameters for special case experiments

V. Results

The results from the experiments provided us with significant insight into the performance of our fault detection algorithm. Over the 105 experiments recorded, 93 trials resulted in a true positive detection, where the fault detected was in the expected robot, and 12 trials resulted in a false positive detection, where a fault was detected, but not in the expected robot. There were zero negative detections, resulting in a 100% detection rate. This also demonstrates the robustness of the fault detection algorithm as changing R and α did not affect the detection rate. The parameters did however affect the detection time, as seen in Figure 9.

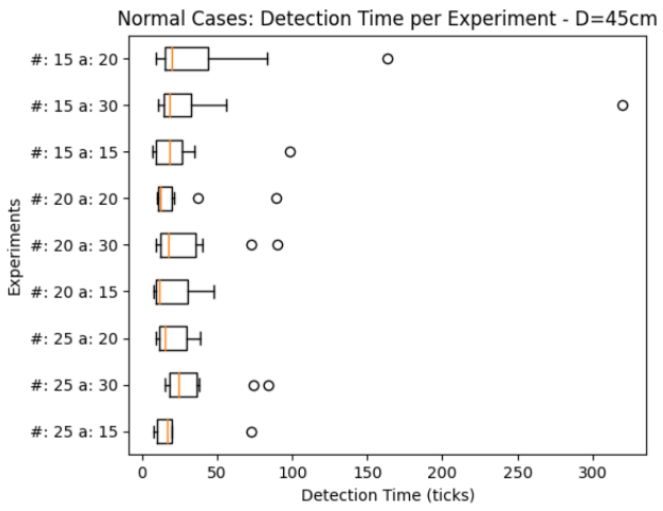


Fig. 9: Detection time of normal experiments

In the special case experiments (Figure 10) it is clear there is some correlation between size of $distance$ and detection time. When $distance$ was the largest at 55 centimeters, the detection time was also the largest, with a median of about 175 ticks. On the other hand when the $distance$ was the smallest at 45 centimeters for the special cases the detection time was actually much smaller and happened almost immediately in just 15 ticks. These are expected results as when the $distance$ is larger, the robot would need more time to lose sight of it's neighbor and detect faulty behavior, and when its smaller, the neighboring robots would only be detected for a much shorter period as a faulty robot turns direction.

The detection time was also affected by the type of detection present in the trial Figure 11. When comparing the speed of detection of true positive and false positive detections, we found that while trials that detected true positives mostly detected them quicker, there were also a majority of outliers within that dataset, as compared to the speed for a false positive to be detected. These outliers can be associated with the speed of the wheels of the faulty robot. Both Figure 12 and Figure 13 display the effects of the wheel speed on the types of false positives. For false positive detection a linear regression was applied and found that when the wheels were set to be the same speed, both positive and negative, a false positive would occur. When the robot drove backward, with a negative wheel speed, it would tend to crash into the robot behind it hindering that robots performance and position, causing the false positive.

The outliers of Figure 11 can also be correlated with wheel speed. Figure 13 shows that when the wheels are both positive, and when the left wheel is larger than the right wheel, more time is required to detect a fault. This is due to the reason that when the left wheel is a larger speed than the right wheel, the robot would turn into the circle and be more likely to see 2 neighbors within the threshold of the experimental setup.

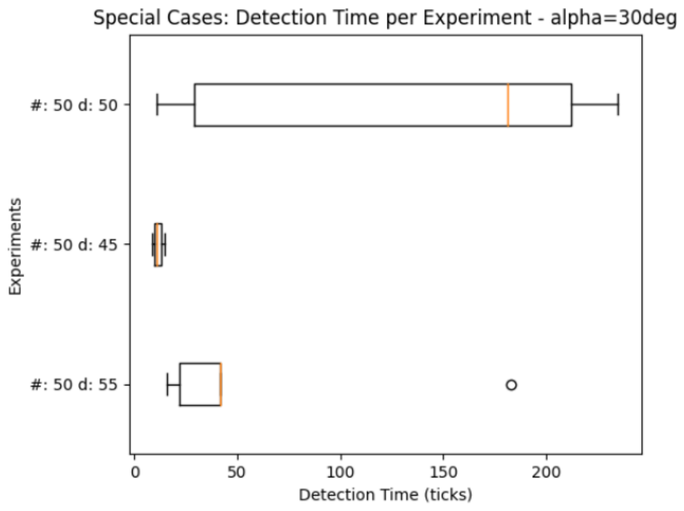


Fig. 10: Detection time of Special case experiments

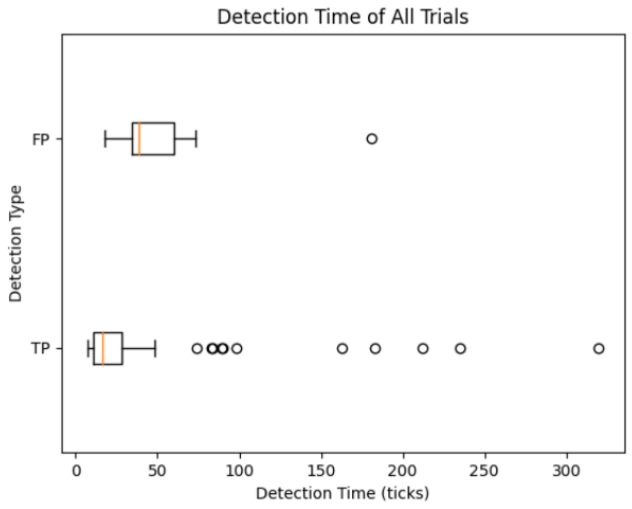


Fig. 11: Detection time of True Positive and False Positive detections

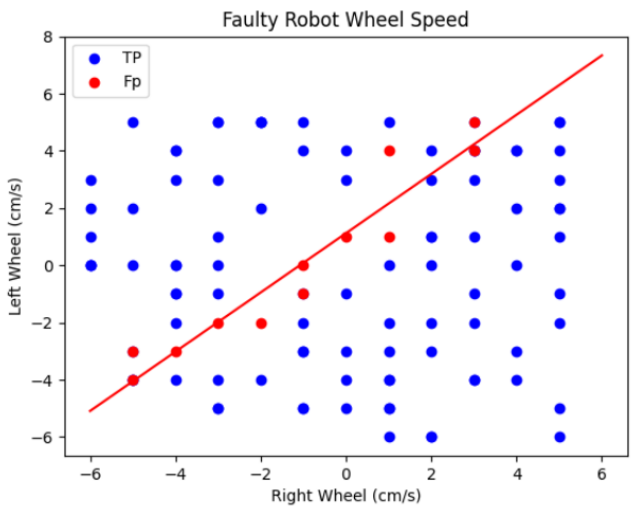


Fig. 12: Wheel speeds of the faulty robots

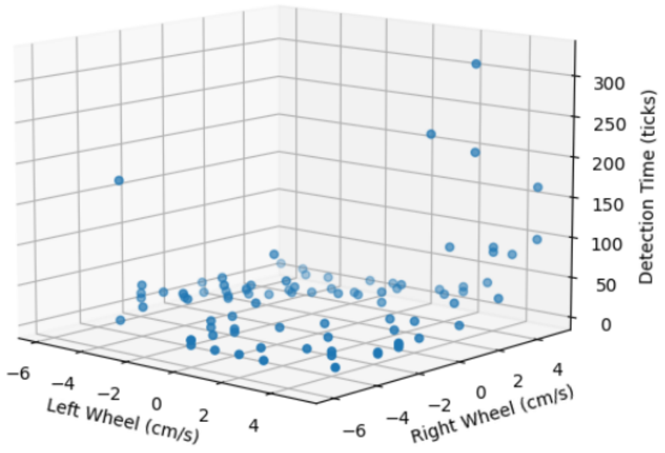


Fig. 13: Detection time with respect to wheel speed of faulty robot

VI. Conclusion and Future Work

In this project, our team aimed to implement an endogenous fault detection system for a swarm of Khepera IV robots executing a circle formation behavior in simulation. We implemented and modified an existing circle formation algorithm from [2]. We also successfully developed a simple fault detection algorithm that uses the range-and-bearing sensors of the Khepera IV to create a simulated FOV in front of and behind the robot. The robot then looks for close neighbors within those FOVs and if there are none, signals a fault to the rest of the swarm. With these algorithms in place, our team ran a total of 105 experiments to evaluate the effectiveness of our fault detection algorithm.

There are many different directions through which this project could be expanded. One significant expansion to our work would be to introduce different types of faults that can occur. In our experiments, we only induced a single type of fault in the actuators of the robots. It would be an interesting exercise to observe how the algorithm would perform if perhaps the proximity sensors were to fail or give false readings. Another area for expansion would be to implement some form of circle formation detection in the swarm. Currently, our experiments wait for an experimentally derived number of time steps to start the fault detection algorithm. Implementing a circle formation detector would eliminate the need to hard-code a trigger time step and allow the swarm to determine when to enable the fault detection algorithm dynamically. Additionally, a circle formation detector would enable the ability to implement dynamic reconstruction of the swarm's circle formation in the event that a faulty robot disrupts the formation. Another point of expansion for this project could be to dynamically adjust the criteria for flagging a fault in a robot. When running our experiments, we had to experimentally find a minimum value for our starting parameters that would not cause the swarm to fault instantly when fault detection was enabled. Figuring out a way to calculate minimum required starting parameters based on known values in the simulation would be a useful feature. Lastly, our team feels that the project could benefit from having the algorithm tested on different movement patterns and behaviors, such as flocking, more complex leader follower behaviors, etc.

References

[1] B. Khaldi, F. Harrou, F. Cherif, and Y. Sun, "Monitoring a robot swarm using a data-driven fault detection approach," *Robotics and Autonomous Systems*, vol. 97, pp. 193–203, 2017.

[2] D. St-Onge, C. Pinciroli, and G. Beltrame, "Circle formation with computation-free robots shows emergent behavioural structure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5344–5349.

[3] D. Tarapore, A. L. Christensen, and J. Timmis, "Generic, scalable and decentralized fault detection for robot swarms," *PloS one*, vol. 12, no. 8, p. e0182058, 2017.

[4] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, 2008.

[5] ——, "Exogenous fault detection in a collective robotic task," in *European Conference on Artificial Life*. Springer, 2007, pp. 555–564.

[6] A. T. Vemuri and M. M. Polycarpou, "Neural-network-based robust fault diagnosis in robotic systems," *IEEE Transactions on neural networks*, vol. 8, no. 6, pp. 1410–1420, 1997.

[7] S. Roumeliotis, G. Sukhatme, and G. Bekey, "Sensor fault detection and identification in a mobile robot," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, vol. 3, 1998, pp. 1383–1388 vol.3.

[8] P. Li and V. Kadirkamanathan, "Particle filtering based likelihood ratio approach to fault diagnosis in nonlinear stochastic systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 31, no. 3, pp. 337–343, 2001.

[9] D. Kengyel, H. Hamann, P. Zahadat, G. Radspieler, F. Wotawa, and T. Schmickl, "Potential of heterogeneity in collective behaviors: A case study on heterogeneous swarms," *PRIMA 2015: Principles and Practice of Multi-Agent Systems*, vol. 9387, pp. 201–217, 2015.

[10] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugniere, G. Di Caro, F. Ducatelle, E. Ferrante, A. Forster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Retornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stutzle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard, "Swarmanoid: A novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics Automation Magazine*, vol. 20, no. 4, pp. 60–71, 2013.