Flocking Behavior in the Presence of Robot Failure

Charles Yoo [cyoo28@seas.upenn.edu]

***Abstract*— Multi-robot system (MRS's) are currently used for a wide range of problems, ranging from coverage to flocking to stigmergy. When using an MRS there are many considerations including coordination, communication, and task allocation. One important factor that has not been discussed much is robot failure. With a single robot, there are a multitude of sources for potential faults, including from sensors, actuators, and software errors. An MRS introduces even more possibly for failure with the necessity for coordination and communication between the individual robots. There is also the fact that there are simply more robots subject to potential failure. Given how prevalent fault risk is, it is important to have a good understanding of the different types of faults and how to address them. The purpose of this paper is to investigate fault detection and diagnosis for multi-robot systems, specifically in the context of flocking. In the field of flocking, there are various models that have been developed using fault-tolerant techniques (FTTs). A few of these methods are explored and compared to one another. To assess the impact of FTTs, one of these techniques is adapted to the Boids algorithm that has been covered in the Reynolds paper. The performance of this FTT model is then compared to the unaltered Boids model to gain some insights into the effectiveness of FTTs.**

I. Introduction

A. Flocking

Flocking is the coordinated movement and behavior of multiple robots. In nature this occurs as schools of fish or flocks of birds. These animal formations are highly organized and would seem to suggest the presence of a centralized control. However, the overall behavior of the flock is actually the result of the combined behaviors of the individual members [1]. In the context of robotics, MRS's seek to emulate this behavior by using robots that follow relatively simple control laws that give rise to coordinated movement. This type of system is highly distributed and does not require any centralized controller. Some benefits of such an algorithm is that it is highly scalable and can achieve complex behaviors with relatively simple mechanisms.

This paper will focus on fault detection and diagnosis in the context of flocking. The Reynolds Model for bird-like objects (boids) will be used as the base model. In the next section we will discuss some failure tolerant techniques for flocking. One of these techniques will be applied to the Reynolds Model and its performance will then be compared to the base model.

B. Fault Detection and Diagnosis

In robotics, failures (or faults) can arise from a variety of different sources. For an MRS, this not only includes the individual robots but also communication, planning, and coordination. These failures can have a significant impact on the overall system: reducing performance and efficiency, compromising the safety of the robots and their environment, and potentially resulting in failure to complete a task [2]. The purpose of fault detection and diagnosis (FDD) is to detect faults when they occur and to mitigate the effects of robot failure. This can either be done through fault recovery, in which the faulty robot is restored to a non-faulty state, or through the change in behaviors of non-faulty robots. To understand FDD, we will first discuss failure in the context of a single robot system. We will then discuss failure in the context of a multi-robot system and explore the challenges of dealing with multiple robots.

1) Failure for Single Robot Systems: An individual robot is composed of both physical (hardware) and virtual (software) components. The basic hardware includes sensors, actuators, and a computer. The computer runs the software that dictates the behavior of the robot. A robot first interacts with its environment by gathering new data through its sensor(s). The computer then processes this data into usable information about the environment. Based on this information, the robot then develops a plan for how to behave. Finally, the robot executes this plan and the actuator(s) act on the environment. This cycle occurs over and over until otherwise stated (i.e. the goal has been completed or is no longer achievable). This cyclic behavior can be seen in figure 1. While this cycle is running, faults can occur in both the hardware and software side of the robot. For hardware, this includes sensor failures that prevent the robot from accurately sensing its environment and actuator failures that prevent the robot from correctly acting as it has planned. For software, this includes planning failures which require the robot to perform tasks that are infeasible and processing failures in which the data is not accurately converted. Regardless of the source, faults can be compounded as the cycle continues and can seriously impact the performance of the robot. The purpose of FDD is to quickly identify and repair these faults so that the robot can effectively achieve its goal.

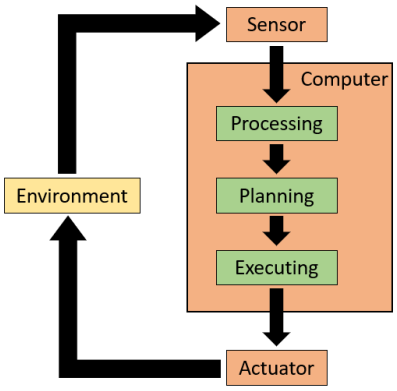

Fig. 1. Robot Behavior Cycle. Adapted from [**?**].

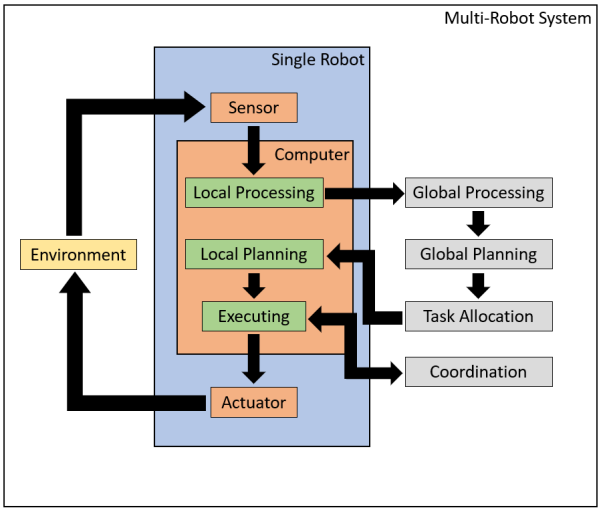

Fig. 2. Robot Behavior Cycle in an MRS. Adapted from [**?**].

2) Failure for Multi-Robot Systems: The types of failures that are present for single robot systems are further compounded in MRS's, simply because of the introduction of more that one robot. Since the single robot is now working with other robots there is a slight change in the overall behavior structure. The part of the environment that the robot interacts with is now on a local scale when compared to the global scale of the overall system. This means that the data collected by the robot's sensor(s) only reflect the robot's local perspective. For an effective MRS, this local data must then be aggregated to create global information that provides a better perspective of the overall environment. Using this data, a global plan for all of the robots can then be created and tasks within this plan can be allocated to individual robots. Once the robot receives a task, it can create a local plan for how it wants to complete the task. Finally, the robot executes this plan. However, during execution, the robot must now coordinate with the other robots within the system. The new cylic behavior of a single robot in an MRS can be seen in figure 2. As stated previously, the failure problem for MRS's is much more complex than for single robot systems. For starters, the global information that is created for the robots relies heavily upon the local data collected by the robots. Any discrepancies with one robot's local data can produce errors in the global information. Additionally, each robot creates its local plan to achieve its assigned tasks. Sometimes these local plans can interfere with each other and create more inefficiencies. There is also the fact that robots must now communicate in order to coordinate with the other robots. This introduces another source of failure in which a robot becomes unable to communicate. Performing FDD becomes increasingly more difficult because it must be able to identify failures on both a local and global scope. Additionally, for failures at a global scope, FDD must be able to trace the source of a failure back to an individual robot.

It is evident that FDD is more difficult for MRS's than for single robot systems. However, there is one interesting benefit that MRS's has for performing FDD. In a single robot system, FDD must be conducted either onboard or through an external watcher (in many cases this is simply a human). For real-world applications, relying on an external watcher to identify and correct faults is slow and cannot be used for cases that require real-time solutions. This means that robots must now monitor its own systems, which requires that FDD be computationally light so as to not interfere with other processes. For MRS's, especially when there is a high degree of collaboration, each robot can monitor other robots and identify when there is abnormal behavior. This inter-robot monitoring can be used in conjunction with intra-robot monitoring and aid with FDD.

C. Failure Tolerant Techniques for Flocking

1) Offline Optimization Technique: The first paper creates a simple leaderless flocking algorithm that can perform flocking behaviors [3]. The robots are spawned randomly in one corner of the map and try to move as a flock to a target destination in a different corner of the map. Each robot employs 3 main behaviors: collision avoidance, velocity matching, and flock centering. The collision avoidance behavior is used when as robot detects a potential obstacle and steers away to avoid a collision. The velocity matching and flock centering behaviors are used together and rely on the centroid of the flock. For flock centering, the robot seeks to fly towards the centroid of the flock. For velocity matching, the robot determines how the centroid is changing and matches its velocity with the velocity of the centroid.

In this paper, when a robot fails, it stops moving (i.e. crashes) but is still detectable to the other robots.

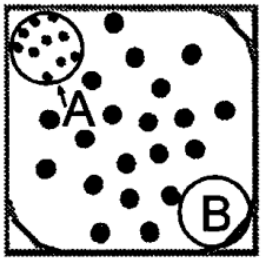

Fig. 3. Robots spawn randomly at location A and seek to flock towards location B. Copied from [3].

This paper does not use an explicit technique for managing individual robot failure. Rather it uses reinforcement learning to optimize the algorithm so that is able to flock, even with the inclusion of robot failures. Tuning parameters associated with collision avoidance is one method for creating an algorithm that can avoid collisions with failed robots.

2) Crash Fault Tolerant Technique: The next paper seeks to create a flocking algorithm that is able to perform well in the presence of crash failures [4]. When a robot crashes, it is no longer able to move for the remainder of the simulation. In this model, robots are unable to directly communicate with each other. Rather they can only communicate visually and each robot relies on visual sensors to monitor other robots. The robots begin in a flocking pattern and seek to maintain a regular polygon formation. Each robot is assigned a unique identifier and one robot is selected as the unique leader of the formation. Based on the leader and its neighbors, each robot will calculate a target position that will maintain the formation.

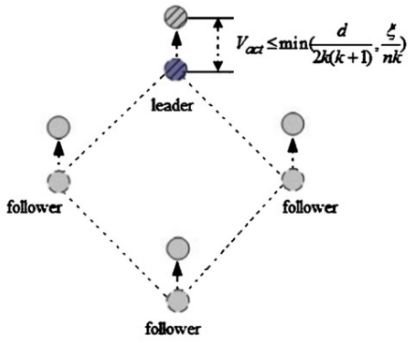

Fig. 4. Robots move to keep formation consistent. Copied from [4].

The failure tolerant technique used on this paper relies on visual data. In order to maintain the formation, each robot must move to a new position. However, crashed robots are unable to move to a new position. Therefore, robots will determine that another robot has crashed when it has not moved for too many time-steps. When a failed robot is detected, the remaining robots reconfigure themselves to continue maintaining the desired formation.

3) Actuator Fault Tolerant Technique: The goal of this paper is to create an algorithm that is free of collisions, even in the presence of actuator failures [5]. The actuator faults implemented in this paper cause the robot to fly at a fixed velocity and is irreversible. For the model used in this paper, robots communicate its state information with all other robots. The state information includes the robot's current position and target, previous position and target, and neighbors it suspects of having failed. Based on the information of the other robots, each robot will then calculate a target position and adjust its velocity accordingly. When a flocking pattern emerges, the flock will then fly to its final destination.

The failure tolerant technique used in this paper requires each robot to monitor its neighbors. A robot is able to use the received information about current and previous states to estimate the velocity of its neighbors. If a neighbor moves when it is not supposed to or has the incorrect velocity, then it is added to the list of suspected failed robots. Since this list is shared between neighbors, if one robot suspects another robot of failing, then the whole flock does as well. The movement of the robots in the flock is simplified to multiple lanes. Non-faulty robots will move to different lanes to avoid robots that it suspects of failing.

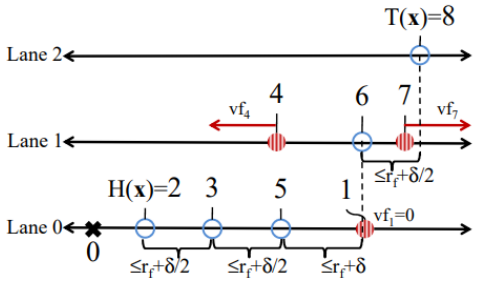

Fig. 5. Robots 1, 4, and 7 are suspected of failing. The remaining non-faulty robots switch lanes to avoid colliding with these failed robots. Copied from [5].

D. FTT Comparison

The algorithms discussed in the previous section provide failure-tolerance to 2 important types of failures in flocking problems: actuator and crash failures. This paper has discussed 2 techniques for crash failures and 1 technique for actuator failures. Between the 2 techniques

for crash failures, the crash fault tolerant technique will provide better performance than the offline optimization technique. This is because the crash fault tolerant technique uses an explicit module for detecting and handling crash failures. On the other hand, the offline optimization technique requires parameter tuning to handle crash failures. A model that uses this technique must be re-optimized whenever it is used for a new environment and is therefore not very robust. The FTT for actuator failures is similar to the FTT for crash failures in that it also includes an explicit module for detecting and handling actuator failures.

A direct comparison between FTT designed for actuator failures and FTT designed for crash failures is difficult because they are fundamentally different. Therefore, it would not make sense to try and determine which FTT is better. That being said, it can still be stated that both FTT models are limited in their application as they can only be applied to one type of failure. It could be possible that these FTTs could be used in tandem to create an even more robust model that can handle both crash and actuator failures.

II. Experimental Setup

A. Problem Overview

The flocking problem implemented in this paper is fairly simple. The simulation takes place in a grid world shown in figure 6. A group of robots will be spawned in random points at location A, which itself is located in the bottom left corner of the world. This group will then fly to an intermediate location B, which is located in the opposite corner of the world. This problem serves as a simplification of a migration behavior of birds.

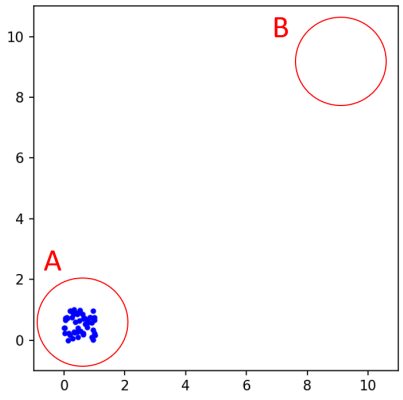

Fig. 6. Simulation Set Up with Flocking

2 models will be compared for this experiment. One is the Reynolds flocking algorithm, created by Craig Reynolds. The model has been slightly adjusted to include target following. The second model builds upon the first model and includes an FTT discussed in the previous section.

B. Reynolds Model

The Reynolds Model model has 3 main principles that lead to flocking formation: flock centering, obstacle avoidance, and velocity matching [1]. These principles are also known as cohesion, separation, and alignment respectively. Under the cohesion rule, robots seek to move closer to neighboring robots:

$$u_i = k_{coh}(\sum_j x_j - x_i) \tag{1}$$

where k_{coh} is the cohesion gain and x_i is the position of the i^{th} robot and x_j is the position of the j^{th} neighboring robot of x_i. Under the separation rule, robots aim to avoid collisions with obstacles, which includes neighboring robots:

$$u_i = k_{all}(\dot{x}_j - \dot{x}_i) \tag{2}$$

where k_{all} is the alignment gain. Under the alignment rule, robots look to adjust their velocity to match neighboring robots:

$$u_i = k_{col}(\sum_j x_i - x_j) \tag{3}$$

where k_{col} is the collision gain. For obstacles, this equation is:

$$u_i = k_{col}(x_i - x_{obst}) \tag{4}$$

For this paper, there are no obstacles in the world so this term can be ignored. The overall control term that dictates the motion of a robot is:

$$\begin{aligned} u_i = k_{coh}(\sum_j x_j - x_i) + k_{all}(\dot{x}_j - \dot{x}_i) \\ +k_{col}(\sum_j x_i - x_j) \end{aligned} \tag{5}$$

The robots act in discrete time, with a time-step of Δt. At the beginning of each time-step, each robot sends their current position to their neighbors. Each robot then calculates its control term based on equation 5 and applies this term to its velocity. Finally, the position of each robot is updated by multiplying the new velocity by Δt and applying this to its position. This relatively simple model has already been proven to exhibit flocking behavior.

C. Communication Adjustment

In the Reynolds Model, each robot communicates its position with its neighbors. For the model used in this paper, robots will not only communicate their position but also their velocity. While robots cannot communicate whether or not they have experienced actuator failure,

they can communicate which robots they suspect of failing.

D. Target Implementation

Each robot is given the centroids of the intermediate target locations as well as of the final target location. The current target location affects the overall movement of the robots as follows:

$$u_i = k_{tar}(x_{tar} - x_i) \tag{6}$$

where k_{tar} is the target following gain and x_{tar} is the position of the target location. The new equation is now:

$$\begin{aligned} u_i = k_{coh}(\sum_j x_j - x_i) + k_{all}(\dot{x}_j - \dot{x}_i) \\ +k_{col}(\sum_j x_i - x_j) + k_{tar}(x_{tar} - x_i) \end{aligned} \tag{7}$$

Each robot has a flag, F_{reach}, which indicates whether or not it has reached the current target location. This flag is triggered when the robot is within a region of ϵ about the centroid (representing the overall target location). When this flag is set, the robot brakes until it reaches a gradual stop.

E. Failure Implementation

The type of failure that is explored is actuator failure. Initially, all robots are non-faulty and their failure flags, F_{fail}, are set to False. This flag is not communicated between robots and is used for assessing flocking performance. A robot has a probability, p_{fail}, of experiencing an actuator failure. When this occurs, F_{fail} is set to true. The robot is no longer able to accelerate or change direction and must continue traveling at a constant velocity. For this paper, this failure is permanent and cannot be reversed.

F. FTT Algorithm

The FTT algorithm designed in this paper is very similar to the actuator fault technique discussed in the prior section. Each robot has a list, G_{sus}, which is a list of neighboring robots that it suspects have experienced failure. A robot that experiences actuator failure is no longer able to accelerate and travels at a constant velocity. Therefore, if a neighboring robot is traveling at constant velocity for multiple time steps, it will be suspected of failing. Normally under Reynolds model, a separation term is enough to prevent collisions between robots. However, this assumes that all the robots are able to adjust their velocities to avoid collision. Since a failed robot cannot change its velocity, the correct robots must compensate:

$$u_i = k_{cor}(\sum_{js} x_i - x_{js}) \tag{8}$$

where k_{cor} is the correction gain and x_{js} is the position of the js^{th} neighboring robot that is suspected of failing. The full equation modeling the behavior of the robots is now:

$$\begin{aligned} u_i = k_{coh}(\sum_j x_j - x_i) + k_{all}(\dot{x}_j - \dot{x}_i) \\ +k_{col}(\sum_j x_i - x_j) + k_{tar}(x_{tar} - x_i) \\ +k_{cor}(\sum_j x_i - x_j) \end{aligned} \tag{9}$$

G. Performance Indicators

At each time step, there is a performance heuristic, which indicates how strong the flocking formation is. This heuristic is the average distance between an individual robot and its neighbors, which is averaged over all robots:

$$H = \frac{1}{N_i c} \sum_i c(\frac{1}{N_j c} \sum_j c|x_i c - x_j c|) \tag{10}$$

where $N_i c$ is the total number of correct (non-faulty) robots and $N_j c$ is the total number of correct neighboring robots for the ic^{th} robot. This heuristic will only evaluate the flocking performance of correct robots that have not experienced actuator failure. A lower heuristic value indicates that robots are flying closer together and are likely to exhibit flocking behavior. On the other hand, a higher heuristic value indicates that robots are flying farther apart and may not be flocking. During the experiments, the number of collisions between robots and the successful navigation of the flock will be recorded. A good model will generate flocking behavior that successfully navigates between target locations with little to no collisions.

H. Variables of Interest

One variable of interest will be the number of robots used in the simulation. For these experiments, flocks of 5, 10, 25, 35, and 50 will be used. Another variable that will be monitored will be the number of failures that occur over the course of the simulation. While this will not be changed directly, it will be influenced by the failure rate. It will be interesting to see how the flock performance changes as the simulation progresses and the number of failures increases.

III. Results and Discussion

A. Comparing Algorithm Performance

One easy way to assess the effectiveness of FTTs is to compare the number of collisions that occur with the base model to the number that occur with the FTT model. Tables 1 and 2 show the comparison of collisions between the base and FTT models, respectively. The

failure probability for these experiments is held constant at 0.05%. For each configuration, 20 trials are run and the data is then averaged across these 20 trials. One drawback to using this metric is that it does not reflect how many failures have occurred. As more failures occur, it is also more likely that collisions occur. Therefore, the tables also provide the normalized number of collisions per failure.

TABLE I
CHANGING ROBOT COUNT FOR BASE MODEL

Robot Count	Time (s)	Collisions	Collisions per Failure
5	6.1	0	0
10	7.5	0	0
25	7.9	5.5	1.6
35	13.7	8.2	2.4
50	24.6	15.4	4.3

TABLE II
CHANGING ROBOT COUNT FOR FTT MODEL

Robot Count	Time (s)	Collisions	Collisions per Failure
5	6.2	0	0
10	7.7	0	0
25	13.6	5.2	1.4
35	19.9	7.9	2.7
50	33.5	15.7	4.5

For configurations with a smaller flock (i.e. 5 and 10), there are no collisions for either model. It is observed that over the 20 trials, the number of failures averaged below 0.2 for 5 robots and below 0.5 for 10 robots. For configurations with a bigger flock, both algorithms experienced some collisions. However, there is no discernable difference between the performance of the base and FTT models. Therefore, this FTT model did not have the desired effect of reducing the number of collisions caused by robot failure. Another thing to note is that the FTT model takes longer than the base model to complete each configuration. The difference in completion time between the 2 models increases as the robot count increases. This suggests that there is some added overhead for checking for robot collisions. To be more specific, this overhead is likely due to the fact that the base model does not have to account for robots that have failed and robots can fly more directly to the target locations.

In the previous experiments, the number of robots was the main variable that was changed. As the number of robots increased, the number of failures also increased. Therefore, increasing the number of robots is an indirect way of increasing the number of failures. A more direct approach for increasing the number of failures is to increase the failure rate of the robots. Tables 4 and 6 show the comparison of collisions between the base and FTT models, respectively. The number of robots is held constant at 15 robots. Again, for each configuration, 20 trials are run and the data is then averaged across these 20 trials.

TABLE III
CHANGING FAILURE RATE FOR BASE MODEL

Failure Rate (%)	Time (s)	Collisions	Collisions per Failure
0	7.6	0	0
0.05	7.5	5.2	1.7
0.10	7.7	5.1	1.5
0.15	7.9	5.7	1.6

TABLE IV
CHANGING FAILURE RATE FOR FTT MODEL

Failure Rate (%)	Time (s)	Collisions	Collisions per Failure
0	7.9	0	0
0.05	7.8	4.9	1.6
0.10	8.1	5.4	1.9
0.15	8.0	5.9	1.7

Just like in the previous experiments, for these configurations, the base model generally reaches the final destination faster than the FTT model. The time it takes to reach the final destination does not seem to increase as the failure rate increases. However, there is a noticeable relationship between the failure rate and the number of collisions. As the failure rate increases, the number of collisions increases. It is also worth noting that as the failure rate increases, the number of robot failures increases. This explains why the number of collisions per failure does not really increase as the failure rate increases.

B. Further Characterizing Flocking with Failure

The previous section looked at the robustness of the base and FTT models. It presented the completion time and number of collisions for these models in different configurations. Particularly, it looked at how changing the number of robots and the failure rate of each robot affected the performance of these models. This section seeks to develop a deeper understanding of the different variables that characterize flocking behavior.

The first characteristic is flocking spread as a function of time. As time progresses, the number of failures will inevitably increase. Plotting flocking spread as a function of time is therefore a proxy for flocking spread as a function of the number of failures. Figures 7 and 8 are plots showing flocking spread vs time for the base model and FTT model, respectively. The size of the flocks for these figures are 15 robots and the failure rate is 0.05%.

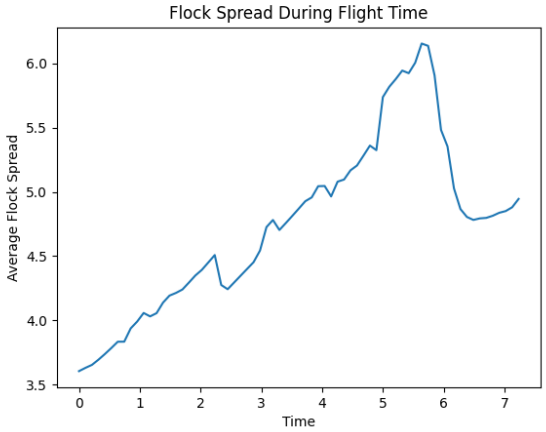

Fig. 7. Flocking Spread vs Time for Base Model

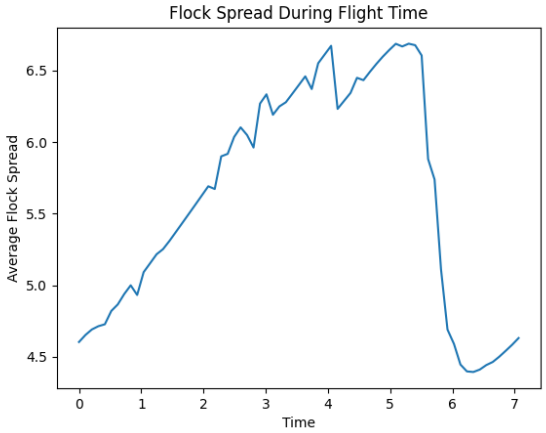

Fig. 8. Flocking Spread vs Time for FTT Model

Based on these plots, it is observed that the flocking spread increases as time increases. At the end, there is a decrease in flocking spread, which is indicative of the flock landing. Since the number of failures also increases as a function of time, the increase in spread can be attributed to the rise in failures. For both models, the flocking spread increases as you increase the number of robots in the system. When there are more robots, there are more factors influencing the cohesion term of the control rule. The contradicting influences on a robot may affect cohesion, which would explain why the flocking spread increases.

It is also worth exploring the affect that the number of robots has on the number of collisions and time to reach the final destination. Figure 9 is a plot showing the number of collisions as a function of the number of robots for the base and FTT models. Figure 10 is a plot showing the completion time as a function of the number of robots for the base and FTT models.

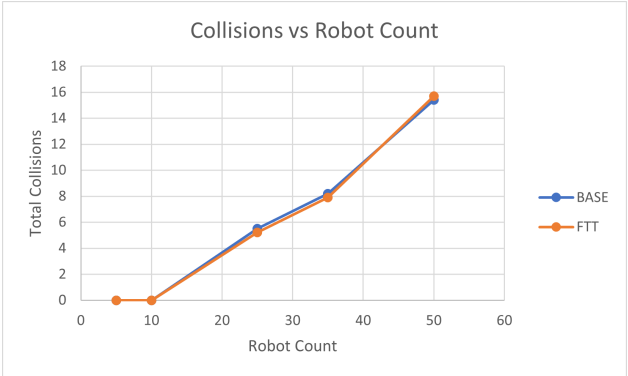

Fig. 9. Number of Collisions vs Number of robots

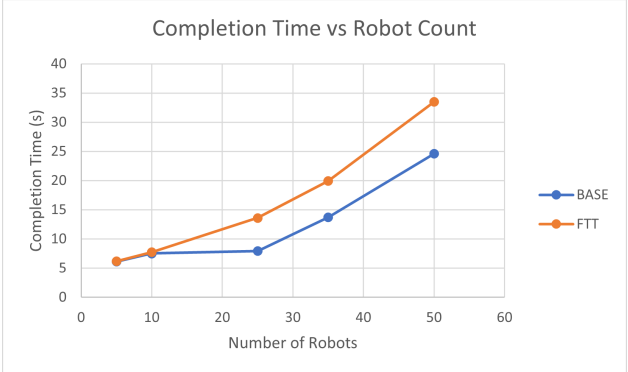

Fig. 10. Completion Time vs Number of robots

Based on figure 9, the number of collisions increases as the number of robots increases, which is what should be expected. As the number of robots in the system increases, the number of failure sources also increases. Additionally, when a robot fails, it has the potential to collide with more robots in a larger system than a smaller one. The base model does not experience significantly more collisions than the FTT model, which seems to indicate that this particular implementation of FTT is not very effective at preventing actuator based failures. Based on figure 10, the time to completion increases as the number of robots increases. With more robots, the target following term becomes less important compared to the other control terms. This would explain why it takes longer to reach the target locations. The base model has better timing than the FTT model because robots do not have to alter their trajectories to accommodate failed robots. This difference becomes more pronounced as the number of robots in the system increases.

IV. Conclusion

This paper has provided a foundational introduction to robotic failures and fault-tolerant techniques. As part of this introduction, we have discussed fault detection and diagnosis and its role for detecting failures in both single robot and multi-robot systems. The focus of this paper has primarily been on fault-tolerant techniques in the context of flocking. By narrowing the scope of review, it becomes easier to understand how fault-tolerant techniques may be applied. As such, 3 different methods for fault-tolerance have been presented. Previous works have shown how effective fault-tolerant techniques are at addressing robot failure but have not really shown how much more effective fault-tolerant algorithms are over their non-tolerant counterparts. To better characterize this relationship, simulations have been conducted comparing a modified Reynolds model to a fault-tolerant model. The fault-tolerant algorithm has proven more difficult to implement than initially anticipated. As a result, the experiments currently presented in this paper have not yielded clear-cut evidence that fault-tolerant techniques increase the robustness of the model in the presence of robot failure. The main issue appears to be with how the algorithm detects robot failure. The current method seems to under-detect robot failure, making it not much better than an algorithm without fault detection. More work will need to be done in order to tweak the fault-tolerant model and make it better at handling robot failure.

V. Future Work

The scope of the simulations presented in this paper is relatively narrow as it only explores the use of one FTT for a relatively simple flocking problem. One area of further exploration would be to try implementing the other FTTs discussed in this paper. Another area of interest would be to test the effectiveness of FTTs in different, possibly more complex flocking problems. Finally, different types of faults do not occur within a vacuum. For instance, in this paper we have discussed failures in which a robot is no longer able to accelerate and failures in which a robot completely crashes. This paper has only covered these failures when they occur independently of each other, however, it is still unknown how effective FTTs are at dealing with situations in which both types of failures occur.

This paper does not offer exhaustive coverage of every FTT. Therefore, further work can be done investigating other FTTs that have not been explored thus far. The FTTs we have covered focus on only one kind of robot fault. There is definitely an opportunity to explore how one might be able to develop an FTT that can cover multiple types of faults (if one has not already been developed). Perhaps the different FTTs that are available could be used in tandem with each other.

We have discussed the identification of robot faults and how to mitigate the impact these faults on the overall system. Some faults are permanent, such as when a robot crashes and is unable to be recovered. Other faults are not as permanent, such as actuator failures, and have the potential to be corrected. It would be of great interest to be able to correct faulty robots whenever possible, which would increase robustness of the system and further mitigate the impact of robot failures. Therefore, further exploration can be done to identify existing correction techniques and to develop an understanding of how non-faulty robots may help correct faulty ones once they are detected.

Additionally, this paper focuses solely on FTTs in the context of flocking. The effectiveness of FTTs can also be explored in the context of other problems, such as search or coverage. Based on the findings of this paper, it is expected that FTT algorithms will still offer better performance over their non-FTT counterparts. Even though this is likely the case, it could be interesting to see the degree to which it improves performance and which methods work best for dealing with failure. There is also the opportunity to see if the same FTT models that work for flocking will also work with other problems.

References

[1] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 25–34, New York, NY, USA, 1987. Association for Computing Machinery.

[2] Eliahu Khalastchi and Meir Kalech. Fault detection and diagnosis in multi-robot systems: A survey. *Sensors (Basel, Switzerland)*, 19, 2019.

[3] A.T. Hayes and P. Dormiani-Tabatabaei. Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 4, pages 3900–3905 vol.4, 2002.

[4] Yan Yang, Samia Souissi, Xavier Défago, and Makoto Takizawa. Fault-tolerant flocking for a group of autonomous mobile robots. *Journal of Systems and Software*, 84(1):29–36, 2011. Information Networking and Software Services.

[5] Taylor Johnson and Sayan Mitra. Safe flocking in spite of actuator faults. In *Proceedings of the 12th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'10, page 588–602, Berlin, Heidelberg, 2010. Springer-Verlag.