Towards Fault Mitigation in a Robot Swarm Using Neuroevolution

Suet Lee[1], and Sabine Hauert[2]
Department of Engineering Mathematics, Bristol Robotics Laboratory,
University of Bristol, United Kingdom
Email: [1]suet.lee@bristol.ac.uk, [2]sabine.hauert@bristol.ac.uk

***Abstract*— As robot swarms are increasingly deployed in the real-world, making them safe will be critical for improving adoption and trust. A robot swarm is composed of many individual robots each susceptible to failure at any given time, which may decrease the performance of the swarm as a whole. The ability to mitigate such faults is therefore necessary. The difficulty with designing a good mitigation strategy lies in the complexity of the swarm as a system, where individual interactions give rise to emergent behaviour. In this paper, we aim to learn a mitigation strategy using neuroevolution in a *fault-discriminatory metric space*. We demonstrate the strategy in a realistic intralogistics use-case.**

I. INTRODUCTION

Swarms have potential in the real-world: they are decentralized systems where an individual only has access to knowledge of its local environment [1]. The overall behaviour of the swarm emerges from local interactions. Examples of real-world applications include search and rescue [2], construction [3], and space exploration [4], [5] [6]. However, in taking these applications from a research environment to real-world implementation, there will need to be a consideration of safety and, in particular, a method to mitigate faults.

Traditionally, swarms have been considered to be robust through redundancy of a large number of individuals [1] although recent work has shown this assumption to be false with the presence of partial failures in a swarm-taxis behaviour [7]. If robustness does not hold for this particular example, what other behaviours could be susceptible to faults? This is a serious consideration if a swarm is to be deployed in the real-world and motivates a need to develop strategies for fault mitigation in the swarm.

The complexity of swarm dynamics means that learning any behaviour to achieve a specific goal, indeed any mitigation strategy, in an optimal way is non-trivial. We ask ourselves the following questions:

1) Can we assume that removal of a fault is always the optimal strategy? Can partially failed robots still contribute to the swarm?
2) If removal is not possible, what other strategies are there?
3) Is there a threshold for tolerance, under which faulty robots do not significantly affect swarm performance and can be ignored?
4) What level of granularity should we consider for mitigation actions?
5) How can we evaluate the effectiveness of a learned strategy?
6) What approach to learning should we take: evolutionary algorithms, reinforcement learning or other?

In this paper, we take some initial steps to providing answers to these questions. In particular for question 1, we find that it may be beneficial to keep partially failed robots in the swarm dependent on the *mode of failure*. We focus on two variations of an intralogistics task, selected to test how mitigation strategies might vary with the interdependence of members of the swarm: random walk behaviour for box retrieval and delivery, and collective transport where multiple robots must coordinate to do the same. We select collective transport as a task as it requires a level of coordination within the swarm that usually requires direct communication, and thus presents an interesting challenge for fault mitigation.

Motivated by previous work, we propose a method to learn a mitigation strategy with the following properties:

1) Mitigation occurs on the level of the individual - each robot selects the best action for overall swarm performance.
2) Learning incorporates knowledge of faulty states of the system - we would like to generate good strategies in a "smart" way, to reduce computation.
3) Strategies are robust and adaptive.

Property (2) can be achieved without explicit fault detection: in previous work, we considered endogenous fault detection in a swarm - robots identifying their own faults - based on what individuals could sense in their local neighbourhoods [8]. We were able to extract metrics based on local-sensing which were highly indicative of a fault. By learning strategies on this metric space, which has high discriminatory power between faulty and normal states of a robot, we can reduce the search space of strategies.

In line with property (3), we turn to neuroevolutionary algorithms: neuroevolution evolves a network to optimise for a given criteria and is effective for searching continuous, high-dimensional solution spaces [9]. It also maintains a population of solutions during search, enables extreme exploration and may be more robust than some reinforcement learning methods. A simple random search of static linear policies, effectively a genetic algorithm applied to a single layer neural network, has been shown to be competitive in the model-free reinforcement learning space [10].

In the method we propose, we learn a strategy for decen-

tralized fault mitigation optimizing for swarm performance (rate of box delivery) for the intralogistics task. We map between local metrics of an individual robot to available actions it may take. Although strategies determine how *individual* robots behave, they are learned with respect to overall swarm performance and this allows for implicit learning of co-ordination between individuals. Finally, we will evolve the strategy offline in simulation, evaluating performance at the end of each simulation run.

II. RELATED WORK

Fault mitigation for swarms is a relatively new area of research whereas there is extensive work for single robot systems. We can identify two approaches in current work: 1) strategies which target a known fault and 2) strategies which maximise the performance of a swarm without explicit knowledge of fault occurrence. For the latter approach, faulty states do not need to be defined nor detected in order for mitigation to occur. The swarm adapts to overcome sub-optimal states for a given performance metric. A diverse set of behaviours can be generated or defined offline and this allows the swarm to adapt online by selecting the best performing behaviours. Previous work applies an "illumination" evolutionary algorithm called MAP-Elites to generate a sufficiently diverse set of behaviours to this end [11][12]. Another work pre-defines sets of behaviours which may be activated or de-activated by individuals in a multi-robot system depending on an activation score [13]. In both these works, behaviours are not designed to mitigate faults directly but rather allow the system to *tolerate* the presence of faults.

In the first category where faults are targeted explicitly, work has been done in learning mitigation strategies with reinforcement learning and evolutionary algorithms [14]. Immune system inspired approaches have considered granuloma formation for energy transfer to recover a robot with low power supply [15]. In both cases, non-faulty robots take direct action to mitigate a fault. The identity of the faulty robot is also assumed to be known. Mitigation actions are behaviour-based, for example a robot may lead a faulty robot to a repair station.

The "fault tolerance" approach has the advantage of being independent of knowledge of faults. The system is able to adapt without any method of fault detection. However, the performance of this approach depends heavily on the *quality* and *diversity* of pre-defined behaviours. To generate such behaviours requires either expert knowledge of the system or extensive computation. Where faults are targeted explicitly, some method of fault detection will be required in a real-world implementation of a fault mitigation pipeline.

The method we propose falls in the space between the two approaches: we are not explicitly detecting faults but we hope to target faulty states by learning mitigation actions on a fault discriminatory metric space.

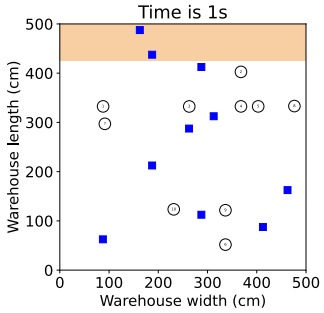

Fig. 1. Warehouse setup: boxes are represented by blue squares, robots by circles. The drop-off zone is shaded orange at the top of the arena.

III. PROPOSED METHOD

A. Scenario

Swarms have the potential to be used out-of-the-box for intralogistics in areas that have not yet adopted robotics, such as SMEs, or in messy real-world environments [16]. In our use-case scenario, the robots operate in a 5m x 5m bounded arena, which we refer to as the warehouse: robots must retrieve and deliver boxes to the drop-off zone, a 75cm-length horizontal strip extending along the width of the upper boundary. Additionally, the boxes are raised up on tables which the robots can detect and navigate under in order to lift. We abstract the table and refer only to the boxes to be lifted. Robots are able to detect objects (robot, box or wall) via ArUco tags. Following previous work in our team, the parameters of the scenario have been selected to match as closely as possible the robot platform and the arena we have available, our aim being to close the reality gap with real-world tests [16] [17]. Table I and Fig. 1 summarize the configuration.

TABLE I
CONFIGURATION

	Property	Value
Warehouse	Dimensions	500 x 500 cm
	Number of boxes	10
	Number of robots	10
	Box diameter	25 cm (task A), 50cm (task B)
Robot	Diameter	25 cm
	Cameras	4 x 120° FOV video cameras equidistant on perimeter, 100 cm range, used to detect objects in the arena at short/mid-range; 1 x 120° FOV video camera upward-facing for precise positioning under boxes
	Proximity	16 x IR laser time-of-flight, 3 m range, used for short-range collision avoidance
	Communication	Bluetooth, 100 cm range
	Robot max speed	200 cm/s (real-time)

B. Tasks and Base Behaviour

1) Task A: One robot is required to lift a single box.

2) Task B: Each box requires a team of four robots to lift. We assume one robot to be positioned at each corner and say the team is *complete* if the team size is exactly four.

If the team is **not complete**:

- Any robot in proximity of the box may join the team.
- Robots in the team will send signals to request other robots to join.

If the team is **complete**:

- No additional robots may join the team.
- Robots must reach consensus before the box is lifted. We then refer to the team as *active*.
- Robots must reach consensus before the box is deposited.
- The team is assumed to move at the velocity of the slowest robot.

Additionally, robots may choose to leave a team at any time and will communicate their intention to the rest of the team. Such a decision will break any consensus to lift a box in an active team and the team will attempt to drop the box if possible.

C. Simulation Environment

The experiments are run on a C++ simulator developed for the purpose of studying the intralogistics use-case scenario. It adheres to real-world physical constraints. Further considerations made in the simulation are:

- **Initialization**: Boxes and robots are placed at random in the simulated arena.
- **Object ID**: Each object in the scenario (robot or box) is given a unique ID.
- **Robot states**: These are states describing the active goal of the robot: (1) a robot is searching for a box, and (2) a robot is carrying a box ready for drop-off.
  Task A: A robot in state (2) will immediately deposit a box upon entering the drop-off zone and the box is removed from the arena. It will then return to state (1).
  Task B: Robots in a team will check for consensus to deposit on entering the drop-off zone. Once deposited, the team disbands and robots return to state (1).
- **Robot movement**:
  Task A: Robots move stochastically in all states and they select a new heading at random at a rate of once per 0.2 seconds.
  Task B: Robots move stochastically unless they receive a signal to join an incomplete team, in that case they move towards the team. If robots are in a team, they are considered to move together as one unit and the chosen heading is the average of proposed headings communicated from each robot in the team. Proposed headings are also selected stochastically. All robots must be able to move in the new direction in order for the team to move.
- **Obstacle avoidance**: Robots have an avoidance margin of 5cm to avoid collisions with each other, boxes and walls.

D. Fault Injection

In previous work which required fault injection in simulation, we considered a realistic sample of faults with respect to the capabilities of the robots and the scenario [8]. We consider the same set of faults in this work with the addition of communication faults (e.g. failure to send or receive messages) which may occur in task B.

When evolving the network between input metrics and output actions, we aim for wide coverage of the input metric space of values. This ensures that a robot is able to determine the best action given any state encountered, hence producing a robust strategy. In practice, this means covering all possible number of faults (from zero faulty robots to all faulty robots) across all fault types. We consider the following faults:

F1: 0% max speed
F2: 10% max speed
F3: 50% max speed
F4: Can't lift boxes
F5: Can't deposit boxes
F6: Radial camera (RC) failure (all cameras)
F7: Upwards-facing camera (UFC) failure
F8: IR laser failure (all lasers)
F9: Can't broadcast messages
F10: Can't receive messages
F11: Spam messages
F12: Sending incorrect messages
F13: Receiving incorrect messages

F6: refers to the cameras on the perimeter of the robot. Faults *F9*:-*F13*: are communication faults relevant to task B.

The faults are injected at the beginning of each simulation and last for the entire duration. Due to computational time constraints, we also inject a *single type* of fault in each simulation to reduce the number of scenario configurations. In reality, there may be a correlation between fault types with subsets of different faults that are likely to occur together. Although exhaustive coverage of configurations may be ideal, our main goal is to demonstrate successful neuroevolution of mitigation strategies given a particular scenario.

E. Mitigation Actions

Mitigation actions available are based on an individual's capabilities (sensing, actuation and communication). It is important to choose the right level of granularity in these actions to match the input metric space. We consider the following set of actions:

A1: Decrease speed
A2: Stop moving
A3: Don't join team
A4: Leave team
A5: Move lifter up (pick up box)
A6: Move lifter down (drop box)
A7: Attract nearest robot
A8: Repel nearest robot
A9: Move in a fixed direction
A10: Stop broadcasting messages
A11: Stop receiving messages
A12: Do nothing

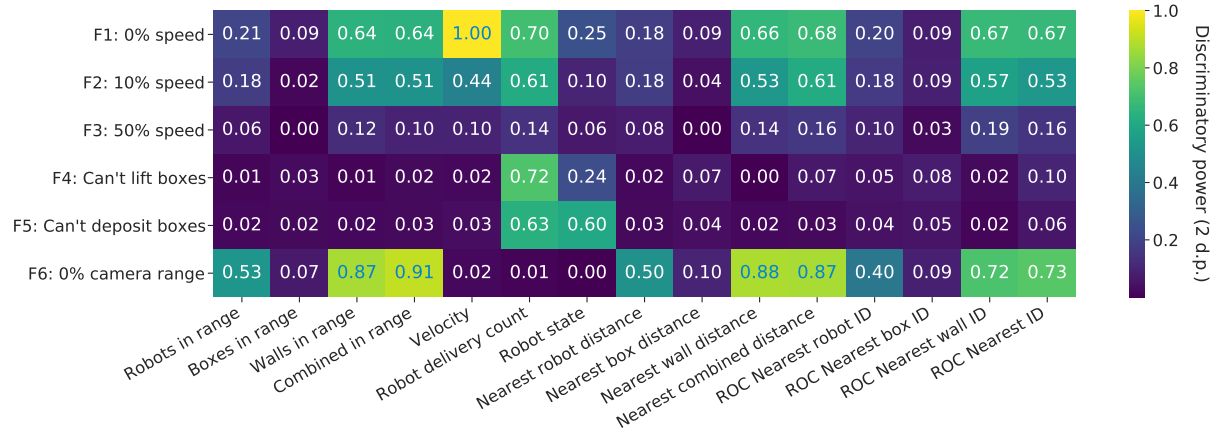

Fig. 2. Discriminatory power matrix for faults types vs. metrics: values sit in the range [0, 1] with 1 being the highest discriminatory power. We can see a metric "signature" emerging where a correspondence between fault type and metrics with high discriminatory power may be identified [8].

(Actions cont.)

A13: Move towards nearest object (robot/box/wall)
A14: Move away from nearest object (robot/box/wall)

Here we assume that taking no action is also a valid strategy - the motivation being that a robot moving slower may still contribute positively to swarm performance, for example.

F. Learning on a Fault-Discriminatory Metric Space

Our aim is to learn a strategy such that an individual robot can act based on knowledge of its local environment, local action which contributes positively to overall swarm performance. The local environment can be represented by metrics which capture the internal state of the robot or the presence and proximity of other objects near the robot. In particular, we are interested in metrics which can discriminate between "faulty" and "normal" states of a robot, so that we can learn the best mitigation action to take for each state. In previous work we presented a statistical method for identifying local metrics with high discriminatory power: in summary, we applied the Mann-Whitney U test (a statistical test of sample group difference) in combination with effect size analysis to take into account sample size [8]. Figure 2 shows the metrics with highest discriminatory power for different types of fault. We call the space of discriminatory metrics, a *fault-discriminatory* metric space.

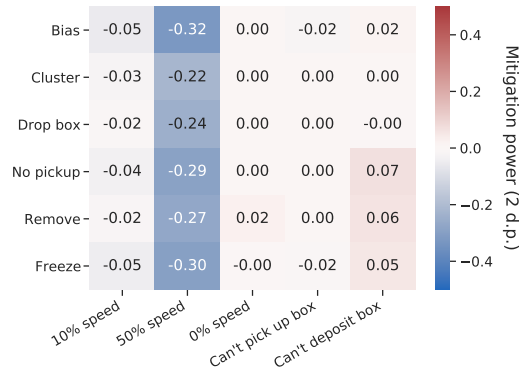

Fig. 3. Mitigation power for various strategies against fault types: these preliminary trials hardcode the mitigation action to be taken when a robot is faulty. The "mitigation power" is computed using the same statistical method as for metric extraction albeit with values in the range [-0.5, 0.5], computing the difference between a sample where no action is taken versus a single mitigation strategy applied to all faulty robots.

1) Preliminary Results: As an initial evaluation of the selected mitigation actions, we performed preliminary trials where the actions were tested against each type of fault. We assumed that faults always occurred at the beginning of each trial and that a faulty robot would choose the mitigation action without delay. A second assumption was that only a single fault type would be present in each trial ranging from 1 to 10 faulty robots.

Figure 3 shows the *mitigation power* of various actions against fault types. The mitigation power is computed applying the same statistical method as previously used for metric extraction, albeit with values in the range [-0.5, 0.5], comparing the swarm performance with mitigation and without. A negative score for mitigation power indicates that a mitigation strategy has worsened performance and a positive score indicates an improvement. From this brute force first pass, we get a sense of the effectiveness of particular actions. In particular, all actions for fault "50% speed" have a negative impact on swarm performance. This suggests that partially failed robots may still contribute positively to swarm performance as even removing these robots, for example, causes the swarm to perform worse.

2) Neuroevolution: We will evolve a network using neuroevolution with the fault-discriminatory metrics as the input and a probability of choosing any mitigation action (from section III-E) as the output. The full list of input metrics is to be determined for each task but we use metrics extracted from previous work as a starting point [8]. The network represents the controller of the robot. It will have a fixed topology of two hidden layers, with ten neurons each. The aim is to evolve the weights of the neural network. We will employ parameters outlined in [18].

IV. DISCUSSION

We have taken initial steps towards answering the questions posed at the beginning of the paper. We have found in preliminary results that a partial fault (e.g. robot moving at half speed) may not be catastrophic and may still contribute positively to swarm performance. If there is a threshold for tolerance to faults, this will depend on the type of fault together with the number of occurrences. Concerning choice of mitigation actions: we have opted for a combination of simpler motor actions and behavioural actions as a first pass. These are actions at the level of the individual but, in combination with the actions of other individuals, they may create an emergent strategy. The trick then is in learning local strategies which may propagate in such a way. One way to drive emergent properties is to select an evaluation metric for the performance of the *collective*. Here we choose box delivery over time to evaluate how mitigation strategies perform as part of the swarm behaviour.

Towards approach to learning: we believe that neuroevolution offers much potential in learning mitigation strategies for a swarm. To begin with, we are only exploring a simple, fixed-topology network but may expand our scope depending on necessity. For a task where robots in the swarm have low interdependence (task A), the mapping between metrics and actions may have low complexity. For task B, it will be interesting to see if actions correlate to metrics which indicate a robot belongs to a team. We can in fact extend task B so that the scenario contains a mixture of single-robot boxes and multi-robot boxes. The extended task could provide a test for whether co-operative behaviours can be learned using our method. Future work will be to implement this neuroevolutionary approach and evaluate its effectiveness. In particular, we would like to assess how alternative approaches such as reinforcement learning might compare.

Finally, in this case we are mapping local metrics to local actions but we can also ask whether we can map group metrics to group actions. This is not at odds with the decentralized nature of the swarm, as groups can be formed and communicate locally. We may even find a mapping between mappings (from local-local to global-global or vice versa) which may pave the way to more powerful learning methods for fault mitigation strategies.

V. ACKNOWLEDGEMENTS

This work was supported by the US Air Force Office of Scientific Research, European Office of Aerospace Research and Development, the UKRI Trustworthy Autonomous Systems Node in Functionality (EP/V026518/1), and the European Union under Grant Agreement 101070918 and UKRI grant number 10038942.

REFERENCES

[1] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics*, E. Şahin and W. M. Spears, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 10–20.

[2] J. Penders, L. Alboul, U. Witkowski, A. Naghsh, J. Saez-Pons, S. Herbrechtsmeier, and M. Habbal, "A robot swarm assisting a human fire-fighter," *Advanced Robotics*, vol. 25, pp. 93–117, 01 2011.

[3] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science (New York, N.Y.)*, vol. 343, pp. 754–8, 02 2014.

[4] E. Vassev, R. Sterritt, C. Rouff, and M. Hinchey, "Swarm technology at NASA: Building resilient systems," *IT Professional*, vol. 14, no. 2, pp. 36–42, 2012.

[5] D. Carrillo-Zapata, E. Milner, J. Hird, G. Tzoumas, P. J. Vardanega, M. Sooriyabandara, M. Giuliani, A. F. T. Winfield, and S. Hauert, "Mutual shaping in swarm robotics: User studies in fire and rescue, storage organization, and bridge inspection," *Frontiers in Robotics and AI*, vol. 7, 2020.

[6] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Frontiers in Robotics and AI*, vol. 7, 2020.

[7] J. Bjerknes and A. Winfield, *On Fault Tolerance and Scalability of Swarm Robotic Systems*, 01 2013, vol. 83, pp. 431–444.

[8] S. Lee, E. Milner, and S. Hauert, "A data-driven method for metric extraction to detect faults in robot swarms," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 746–10 753, 2022.

[9] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, 01 2019.

[10] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," 03 2018.

[11] D. M. Bossens and D. Tarapore, "Rapidly adapting robot swarms with swarm map-based bayesian optimisation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9848–9854.

[12] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *ArXiv*, vol. abs/1504.04909, 2015.

[13] L. Parker, "Alliance: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[14] O. Oladiran, "Fault recovery in swarm robotics systems using learning algorithms learning algorithms," November 2019.

[15] J. Timmis, A. R. Ismail, J. Bjerknes, and A. Winfield, "An immune-inspired swarm aggregation algorithm for self-healing swarm robotic system," *Biosystems*, vol. 146, 05 2016.

[16] S. Jones, E. Milner, M. Sooriyabandara, and S. Hauert, "Distributed situational awareness in robot swarms," *Advanced Intelligent Systems*, vol. 2, p. 2000110, 08 2020.

[17] ——, "DOTS: An open testbed for industrial swarm robotic solutions," 2022. [Online]. Available: https://arxiv.org/abs/2203.13809

[18] S. Hauert, J.-C. Zufferey, and D. Floreano, "Evolved swarming without positioning information: An application in aerial communication relay," *Autonomous Robots*, vol. 26, 10 2009.