EXOGENOUS FAULT DETECTION IN SWARM ROBOTIC SYSTEMS

AN APPROACH BASED ON INTERNAL MODELS

ALAN GREGORY MILLARD

Doctor of Philosophy

University of York
Computer Science

March 2016

ABSTRACT

Swarm robotic systems comprise many individual robots, and exhibit a degree of innate fault tolerance due to this built-in redundancy. They are robust in the sense that the complete failure of individual robots will have little detrimental effect on a swarm's overall collective behaviour. However, it has recently been shown that partially failed individuals may be harmful, and cause problems that cannot be solved by simply adding more robots to the swarm. Instead, an active approach to dealing with failed individuals is required for a swarm to continue operation in the face of partial failures.

This thesis presents a novel method of exogenous fault detection that allows robots to detect the presence of faults in each other, via the comparison of expected and observed behaviour. Each robot predicts the expected behaviour of its neighbours by simulating them online in an internal replica of the real world. This expected behaviour is then compared against observations of their true behaviour, and any significant discrepancy is detected as a fault.

This work represents the first step towards a distributed fault detection, diagnosis, and recovery process that would afford robot swarms a high degree of fault tolerance, and facilitate long-term autonomy.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I must thank my supervisor Jon Timmis, for all of his support and guidance over the past few years. My journey into the field of swarm robotics first began when I applied for his final-year MEng project titled "Self-repairing collective robotic systems", and I haven't looked back since. If it were not for that initial opportunity to break into the field, which directly led to writing the research proposal for my PhD, this thesis surely would not have come about.

Despite numerous setbacks throughout the course of my research, Jon's belief in my abilities remained steadfast, instilling me with the confidence that I needed to see things through to the end. Jon also helped to find the funding required to send me to numerous summer schools and international conferences, where I was able to present and discuss my work, and for that I feel very privileged.

The Engineering and Physical Sciences Research Council funded the first three years of my doctoral studies, but it was thanks to Jon's repeated efforts to keep me employed that allowed me to remain in York and complete my PhD — first as a Teaching Fellow, and now as a Research Associate. I am incredibly grateful for all of the opportunities that Jon has sent my way, as they have been instrumental in the continued development of my academic career.

I must also thank my co-supervisor Alan Winfield, whose contagious enthusiasm for my research has been a constant source of inspiration throughout my time as a PhD student. Although our meetings were infrequent, each one contributed immeasurably to shaping the direction of my research, which is a testament to Alan's great wisdom and attention to detail.

I would like to thank my assessors, Anders Christensen and Fiona Polack, for their constructive criticism of my work, and an enjoyable viva rife with thought-provoking discussion. I greatly appreciate the time they took out of their busy schedules to read my thesis and provide such thoughtful feedback. Thanks especially to Anders, for making the journey to York for my viva, and to Fiona, for her motivating encouragement at every Thesis Advisory Panel meeting.

Thanks also to my line manager David Halliday, for his understanding while I finished writing my thesis during the start of my post-doc.

Special thanks my parents, for their encouragement throughout my time at university, and financial support that kept me afloat between jobs, for which I am very grateful.

I would also like to thank Georgina for her patience during the final throes of my PhD. Without her emotional support to get me through the hard times, this thesis may never have materialised.

Of course, I would not have made it this far without a network of incredible friends and colleagues, too numerous to mention by name. To everyone who helped me along the way, I hope that you know who you are, and please know that I'm very appreciative of the advice and support that you've given me over the past few years.

DECLARATION

I declare that the work presented in this thesis is my own, except where attributed to another author. Some of the ideas, figures, and text have previously appeared in the following publications:

[1] **Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour**. Alan G. Millard, Jon Timmis, and Alan F. T. Winfield. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[2] **Towards exogenous fault detection in swarm robotic systems**. Alan G. Millard, Jon Timmis, and Alan F. T. Winfield. *Towards Autonomous Robotic Systems*, 2014.

[3] **A low-cost real-time tracking infrastructure for ground-based robot swarms**. Alan G. Millard, James A. Hilder, Jon Timmis, and Alan F. T. Winfield. *9th International Conference on Swarm Intelligence*, 2014.

[4] **A low-cost real-time tracking infrastructure for ground-based robot swarms**. Alan G. Millard, James A. Hilder, Jon Timmis, and Alan F. T. Winfield. *University of York Technical Report YCS-2014-489*, 2014.

Part I

INTRODUCTION

1 INTRODUCTION

Swarm robotic systems comprise many individual robots, and exhibit a degree of innate fault tolerance due to this built-in redundancy. They are robust in the sense that the complete failure of individual robots (mechanical or behavioural) will have little detrimental effect on the collective behaviour [5]. However, recent work [6, 7, 8] has shown that partially failed individuals can adversely affect swarm behaviour. Overall system reliability may even decrease with increasing swarm size [7], so this is a problem that cannot simply be solved by adding more robots to the swarm. Instead, swarm robotic systems must take an active approach to dealing with failed individuals if they are to achieve a high degree of fault tolerance.

Despite the fact that recovery mechanisms can only be initiated once a fault has been detected, the problem of fault detection in robot swarms has received little attention. It is often assumed that robots are able to proprioceptively detect faults in themselves (referred to as *endogenous* fault detection), and then signal this to the rest of the swarm. Unfortunately, it may be impossible for a robot to do so in certain situations, such as those where its communications hardware has failed. There has been some recent interest in developing methods of *exogenous* fault detection [9, 10, 11, 12] (where robots attempt to detect the presence of faults in each other), but existing approaches are often only capable of detecting pre-specified faults, or a small number of failed individuals relative to the size of the swarm.

This thesis attempts to address these problems by presenting a novel method of exogenous fault detection for swarm robotic systems, which is based on the comparison of expected and observed behaviours. This approach allows individual robots to detect when other robots in the swarm are behaving in an unexpected manner, and to use this information to determine whether those robots have developed faults. This is an important first step towards engineering fault-tolerant robot swarms, and is necessary if swarm robotic systems capable of long-term autonomy are ever to become a reality.

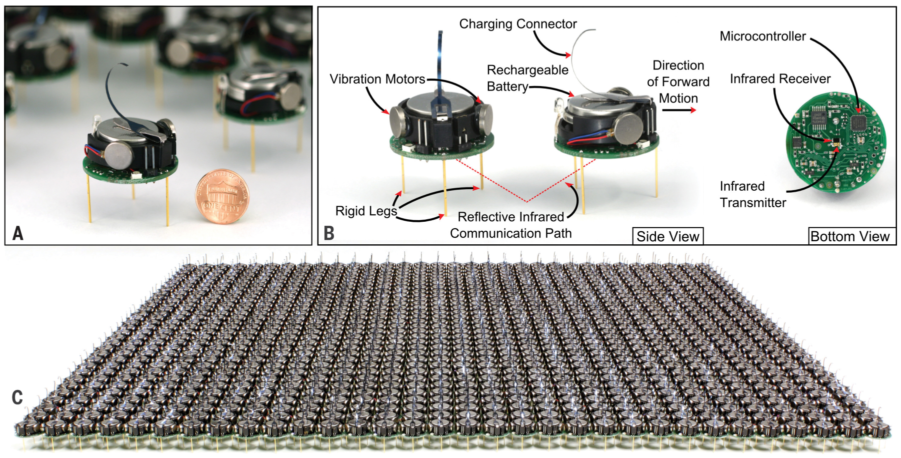


Figure 1.1: (A) Kilobot robot, with a U.S. penny for scale. (B) Each Kilobot is able to move around using vibration motors attached to rigid legs, and can communicate with neighbouring robots by reflecting infra-red light off the ground. (C) 1,024 Kilobot swarm. Taken from [13].

This chapter first outlines the field of swarm robotics, before discussing claims of robustness and how fault tolerance can truly be achieved. The proposed exogenous fault detection approach is then explained at an abstract level, and positioned within the field based on the imposed constraints. Finally, the research hypothesis is given, followed by an overview of the thesis structure and contributions.

1.1 SWARM ROBOTICS

Swarm robotics is an approach to the coordination of robot collectives comprising a large number of relatively simple individuals, inspired by the principles of swarm intelligence [5]. It has become an increasingly popular field of research in recent years [14], in part due to advances in miniaturisation that have facilitated the mass production of simple robots at relatively low cost (compared to single-robot systems). The Kilobot robot platform [15] shown in Figure 1.1 is a prime example — it is not much larger than a coin, and the total cost of parts is under $15. This has allowed a record-breaking swarm of 1,024 robots to be built [13], albeit with quite limited capabilities.

In contrast to multi-robot systems, which typically employ hierarchical or centralised control for coordinating the behaviour of robots, swarm robotics adopts a decentralised approach in which the desired collective behaviours are an emergent consequence of self-organisation and local interactions between robots and their environment [16].

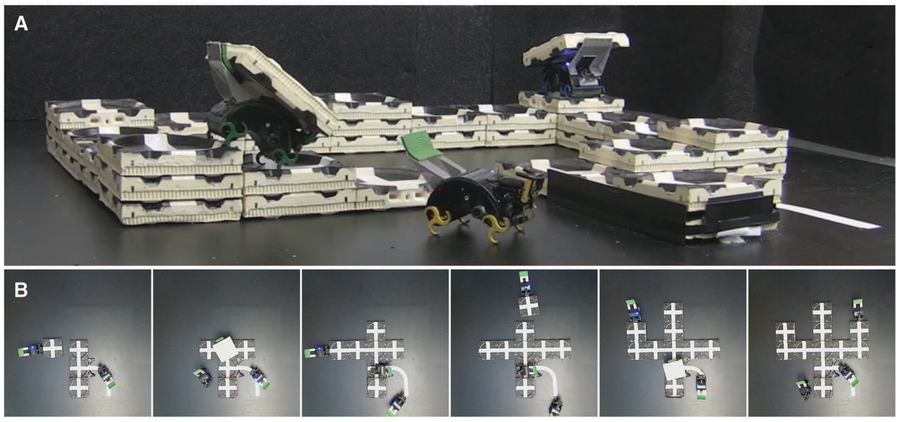


Figure 1.2: (A) Three termite-inspired TERMES robots [18] autonomously building a 3D structure as a collective. (B) Sequence of overhead views showing the progress of collective construction. Taken from [19].

In order to help distinguish swarm robotics research from that focused on multi-robot systems, various different criteria have been proposed to characterise swarm robotic systems [5, 17, 14]. Although each definition differs slightly, the following criteria are generally agreed upon:

- Each robot must act autonomously, rather than under the influence of centralised control. Coordination between robots should be distributed and decentralised.
- The robots should only have local sensing and communication capabilities, with no access to global knowledge.
- The robots must be physically embodied in the environment, and able to interact with it.
- Individual robots should be relatively incapable or inefficient with respect to the task at hand, thus necessitating cooperation.
- The swarm must either consist of a large number of robots, or the swarm behaviour must scale with increasing swarm size.

Complex collective behaviours may emerge from robot systems conforming to this definition, including aggregation, flocking, foraging, cooperative transport, and collective decision-making [20, 21]. Decentralised systems are not without their limitations though, and may lead to sub-optimal solutions due to a lack of global perspective.

Various domains of application have been identified for swarm robotic systems, including environmental monitoring, search and rescue, automated construction (see Figure 1.2), mine clearance, and deep-sea/space exploration [5]. However, such real-world applications are still a long way off — the majority of swarm robotics research to date has been carried out in controlled laboratory environments, and the verification and validation of the emergent and self-organising behaviour of robot swarms remains an open problem [22].

1.1.1 *Robustness*

Bayındır and Şahin [23] argue that any swarm robotic system should, by definition, exhibit three desirable qualities — *robustness*, *flexibility*, and *scalability*, which they define as:

“ ROBUSTNESS: the degree to which a system can still function in the presence of partial failures or other abnormal conditions.

FLEXIBILITY: the capability to adapt to new, different, or changing requirements of the environment.

SCALABILITY: the ability to expand a self-organised mechanism to support larger or smaller numbers of individuals without impacting performance considerably. ”

While it may be true that swarm robotic systems are flexible and scalable, Winfield and Nembrini [6] criticise the general swarm intelligence literature for often claiming that swarms exhibit a high degree of robustness, despite a lack of supporting empirical evidence or theoretical analysis. They also raise questions about what robustness really means, and how it can be quantified and measured, such that the fault tolerance of a swarm robotic system may be assessed. It is important that these questions are answered if robot swarms are ever to make the transition from laboratory experiments to real-world safety-critical applications.

Winfield and Nembrini [6] argue that the term ‘robust’ is often casually used in the context of swarm robotics without explicitly defining it, and in cases where it is defined, there is some confusion over its meaning. They state that a robot swarm may exhibit all of the following forms of robustness, and more:

“

1. it is a completely distributed system and therefore has no common-mode failure point

2. it is comprised of simple and hence functionally and mechanically reliable individual robots

3. it may be tolerant to noise and uncertainties in the operational environment

4. **it may be tolerant to the failure of one or more robots without compromising the desired overall swarm behaviours**

5. it may be tolerant to individual robots who fail in such a way as to thwart the overall desired swarm behaviour ”

The fourth form of robustness — tolerance to the failure of individual robots — is particularly relevant to this thesis. Winfield and Nembrini [6], and later Bjerknes [7, 8], have shown that this form of robustness does not come for free just by using local communication and decentralised control, and that partially failed individuals can compromise collective swarm behaviours. Therefore, additional mechanisms are required if swarm robotic systems are to become truly fault-tolerant.

1.1.2 *Fault tolerance*

Fault tolerance is defined as a system's ability to continue operation, perhaps at a degraded level of performance, despite the presence of faults [24]. In the context of swarm robotic systems this refers to the swarm's ability to cope with failed robots, which is a desirable trait, especially if a robot swarm is required to operate autonomously for extended periods of time without human intervention.

While robot swarms may be tolerant to certain types of faults due to inherent redundancy in the system, Christensen et al. [9] argue that an implicit approach is generally infeasible. Instead an explicit process of fault *detection*, *diagnosis*, and *recovery* is required to make a swarm robotic system tolerant to a wide range of faults. Once the presence of a fault has been detected, diagnosis aims to determine the cause and location of the failure so that appropriate action can be taken to remove, isolate, or mitigate the effect of the fault [25]. This may involve physically removing a faulty robot from the swarm or repairing its failed hardware components, so that the swarm is able to continue unhindered.

1.2 FAULT DETECTION VIA INTERNAL MODELS

Fault detection is a critical first step in this explicit approach to fault tolerance, as diagnosis and recovery can only proceed once a fault has been detected. This thesis proposes a novel method of exogenous fault detection that allows robots to detect the presence of faults in each other, via the comparison of expected and observed behaviour. Each robot predicts the expected behaviour of its neighbours by simulating them online in an internal replica of the real world. This expected behaviour is then compared against observations of their true behaviour, and any significant discrepancy is detected as a fault.

Although beyond the scope of this thesis, the proposed internal simulation approach could also be used for fault diagnosis and predicting the outcome of possible recovery actions, thus providing an architecture for every stage in an explicit approach to fault tolerance.

1.2.1 *Swarm robotics and minimalism*

The use of an internal simulation may seem like a heavyweight approach to fault detection, particularly in the context of swarm robotic systems, where individual robots are typically assumed to be simplistic. However, Sharkey [26] argues that during its short history, the field of swarm robotics has developed beyond its roots in swarm intelligence, and questions whether biological constraints are still relevant, and whether it is still important to strive for minimalism. Various constraints have been adopted by different researchers for a variety reasons, and it is no longer clear when it is appropriate to enforce minimalism and/or biological constraints [26]. In an attempt to resolve this confusion, Sharkey [26] proposes a simple categorisation of swarm robotics studies into sub-areas, as shown in Table 1.1.

A distinction is made between *Scalable* Swarm Robotics and *Minimalist* Swarm Robotics. Scalable Swarm Robotics corresponds to approaches that place emphasis on local communication and decentralised control, to ensure the scalability of the methods developed, but that ignore minimalism and individual simplicity. There are no biological constraints imposed on the ability of individuals, which can be as sophisticated as necessary, provided that their interactions with other robots remain local and decentralised.

Similarly, Minimalist Swarm Robotics enforces constraints on local communication and control, and so too benefits from scalability. How-

Sub-area	Scalable	Minimalist	Natural
Scalable SR	✓	✗	✗
Practical Minimalist SR	✓	✓	✗
Nature-inspired Minimalist SR	✓	✓	✓

Table 1.1: Sharkey's taxonomy of swarm robotics (SR) research [26].

ever, unlike Scalable Swarm Robotics, a commitment is made to the use of simple robots [26]. The motivation for individual simplicity may either come from practical or nature-inspired standpoint. *Practical* Minimalist Swarm Robotics practitioners are interested in using simple robots, because they are cheap, efficient, expendable, and their simple design (software and hardware) affords them inherent robustness. *Nature-inspired* Minimalist Swarm Robotics researchers impose biological constraints on individual robots, based on those found in natural self-organising systems such as social insect colonies [26].

When designing a swarm robotic system, it is therefore only necessary to impose constraints that are relevant to the research question being answered. Over-constraining the system unnecessarily may preclude the discovery of otherwise valid solutions. Conversely, it is necessary to ensure that sufficient constraints are imposed such that the developed system is fit for purpose.

Under Sharkey's taxonomy [26], the concepts presented in this thesis fall into the category of Scalable Swarm Robotics, as the cognitive capabilities of robots are not limited, but decentralised control and local communication are still enforced to ensure scalability. This allows the potential of robots with internal models to be explored, free from the shackles of minimalist constraints.

1.3 RESEARCH HYPOTHESIS

The aim of this thesis is to investigate the problem of exogenous fault detection in swarm robotic systems, with a focus on the use of internal models. The general research hypothesis is as follows:

> *Individual robots in a swarm robotic system can use internal simulations to predict the behaviour of their neighbours, and through the comparison of expected and observed behaviour, can exogenously detect the presence of faults in those robots.*

Although the prediction of robot behaviour is tackled quite differently in Parts II and III of this thesis, the same overarching hypothesis guides the research presented.

1.4 THESIS STRUCTURE

The remainder of this thesis is structured as follows:

Part I - Introduction

CHAPTER 2 - BACKGROUND & RELATED WORK:

This chapter begins with a review of the fault tolerance and reliability of robot swarms, or lack thereof, that motivates this thesis. Approaches to engineering fault tolerance are then discussed in the context of natural/artificial immunity, and the more traditional process of explicit fault detection, diagnosis, and recovery. The chapter concludes with a review of endogenous and exogenous fault detection approaches, with a focus on swarm robotic systems.

Part II - Fault detection via prediction of future behaviour

CHAPTER 3 - PREDICTING FUTURE BEHAVIOUR:

This chapter outlines the proposed exogenous fault detection system, which is based on the prediction of future behaviour. Related work concerning the use of internal models for predicting behaviour is reviewed, and the experimental infrastructure required to implement the fault detection system is discussed.

CHAPTER 4 - SINGLE ROBOT FAULT DETECTION:

This chapter presents the results of initial experimental work carried out to investigate the viability of fault detection based on simulated predictions of a single robot's future behaviour, as an intermediate step towards implementing the exogenous fault detection system proposed in Chapter 3 in a swarm context. The chapter concludes with a discussion of open problems with fault detection based on the prediction of future behaviour, and their proposed solutions.

Part III - Fault detection via analysis of past behaviour

CHAPTER 5 - ANALYSING PAST BEHAVIOUR:

This chapter presents a variation on the exogenous fault detection system originally proposed in Chapter 3, which is instead based on the analysis of past behaviour. Details of the implementation are given, and failure modes designed to test the fault detector's performance are defined.

CHAPTER 6 - FAULT DETECTION PERFORMANCE:

This chapter presents the results of experimental work carried out to assess the performance of the fault detection system proposed in Chapter 5. The fault detector's ability to detect various failure modes is tested, as well as its tolerance to multiple faults of random types. The results of scalability and global sensitivity analyses are also presented.

Part IV - Evaluation and conclusions

CHAPTER 7 - EVALUATION AND CONCLUSIONS:

This chapter summarises the work presented in this thesis, and discusses its limitations. Various potential avenues of future work are suggested, and concluding remarks are made.

1.5 THESIS CONTRIBUTIONS

The main contributions of this thesis are as follows:

CHAPTER 3 - PREDICTING FUTURE BEHAVIOUR:

The fault detection system proposed in this chapter is the first known example of predicting future behaviour via internal simulation for the purpose of exogenous fault detection in swarm robotic systems. This work brings together existing research concerning robots with internal models, and anomaly detection techniques, to produce a novel fault detection system that represents a step towards engineering fault tolerant swarms. This chapter also includes a discussion of key issues that must be considered when attempting to implement embedded simulations on physical robot hardware with the aim of predicting future behaviour. These ideas were published at TAROS 2013 [2].

CHAPTER 4 - SINGLE ROBOT FAULT DETECTION:

The experimental results presented in this chapter show that simulation can be used to successfully predict real robot behaviour, however drift between simulation and reality occurs over time due to the reality gap. This necessitates periodic reinitialisation of the simulation to reduce false positives. It is shown that selecting the length of this reinitialisation time period is non-trivial, and that there exists a trade-off between minimising drift and the ability to detect the presence of faults. The open problems discussed at the end of this chapter also apply to other researchers interested in predicting the future behaviour of individual robots in a swarm. This work was published at IROS 2014 [1].

CHAPTER 5 - ANALYSING PAST BEHAVIOUR:

The revised fault detection system presented in this chapter is the main contribution of this thesis. This chapter details an architecture for exogenous fault detection based on the internal simulation of past behaviour that builds on the novel contributions of Chapter 3, and reduces uncertainty in the predictions of robot behaviour. This architecture could also potentially be used to perform fault diagnosis once a fault has been detected, thus solving the first two stages of an explicit fault detection, diagnosis, and recovery process that would afford swarm robotic systems a high degree of fault tolerance.

CHAPTER 6 - FAULT DETECTION PERFORMANCE:

The experimental results presented in this chapter show that the revised fault detection system proposed in Chapter 5 is able to reliably detect multiple different failure modes, and can cope with multiple simultaneously failed robots. It is also shown that the fault detection performance scales with increasing swarm size, and that the true positive rate is robust to changes in parameter values. This work also represents the first known application of consistency analysis to swarm robotics research, which provides greater confidence in the results obtained.

CHAPTER 7 - EVALUATION AND CONCLUSIONS:
: This thesis exemplifies what can be achieved by robots that use internal simulations to reason about their surroundings, within a fault tolerance context. However, there is still plenty of scope for further work. This chapter suggests potential avenues for future research that extend far beyond the work presented in this thesis.

APPENDIX A - TRACKING INFRASTRUCTURE:
: A tracking infrastructure is a useful tool for swarm robotics research, and has many potential applications. Unfortunately, for many research laboratories, constructing one may be prohibitively expensive. This appendix presents a low-cost infrastructure for tracking ground-based robot swarms in real-time, which has many applications beyond the research presented in this thesis. The information provided in this appendix may be of use to others in the wider swarm robotics research community who are interested in building a similar tracking infrastructure. This work was published at ANTS 2014 [3], and as a technical report [4].

2 | BACKGROUND & RELATED WORK

This chapter begins with a review of the fault tolerance and reliability of swarm robotic systems, and the potential problems caused by partially failed robots, which motivate this thesis. Approaches to engineering fault tolerance are then discussed in the context of natural/artificial immunity, and how concepts from these natural systems map to the traditional process of explicit fault detection, diagnosis, and recovery. Finally, endogenous and exogenous approaches to fault detection are reviewed, with a focus on swarm robotic systems.

2.1 FAULT TOLERANCE AND RELIABILITY

In order to investigate the fault tolerance of swarm robotic systems, Winfield and Nembrini [6] carried out qualitative Failure Mode and Effects Analysis (FMEA) [28] on a collective containment behaviour that causes a swarm to physically surround a beacon in the environment, encapsulating it as shown in Figure 2.1. This section first describes the swarm algorithm used by Winfield and Nembrini [6], before discussing the results of their FMEA.

The robots are able to send messages via range-limited wireless communication, and are considered 'connected' when they are within communication range of each other. Each robot broadcasts its own ID, plus the IDs of neighbouring robots that it is connected to, which allows other robots to determine which of their own neighbours are shared neighbours of the sender [6]. Whenever a connection is lost, a robot checks how many of its remaining neighbours still have the lost neighbour in their own neighbourhoods. If this number is less than or equal to a predefined threshold β (usually set to 2 or 3), then the robot assumes it is moving away from the swarm, and reacts by turning 180°. When a new connection to a neighbour is formed, the robot chooses a random heading. These simple rules give rise to an emergent swarm aggregation behaviour, which allows the robots to maintain a stable *ad hoc* wireless network [6].

Each robot also has a long-range beacon sensor, which only provides binary information — whether or not the robot is illuminated by the beacon. This beacon sensor is deliberately minimal, as it en-

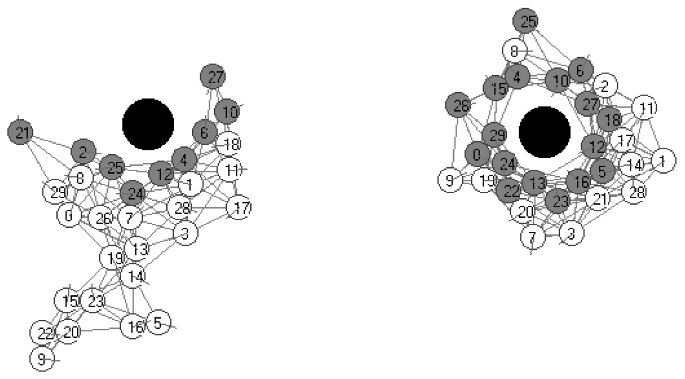


Figure 2.1: A swarm of 30 simulated robots performing beacon encapsulation using the β-algorithm. The beacon is shown in black. Robots illuminated by the beacon are shown in grey. Taken from [27].

sures that a single robot is incapable of completing the task alone, necessitating the use of emergent collective behaviours [6]. When a robot senses that it is illuminated by the beacon, it sets its β threshold value to ∞. If a robot's connection to an illuminated neighbour is lost, then the robot will react (by turning 180°), ignoring the value of the β parameter. This additional mechanism results in the emergence of phototaxis and beacon encapsulation behaviours [6].

The FMEA begins with the identification of all possible hazard conditions. Winfield and Nembrini [6] only consider internal hazards (faults in the robots) in their analysis, as external hazards (environmental disturbances) have already been investigated by Nembrini [27]. The following internal hazards were identified for the emergent beacon encapsulation behaviour:

- H_1: Motor failure
- H_2: Communications failure
- H_3: Avoidance sensor(s) failure
- H_4: Beacon sensor failure
- H_5: Control systems failure
- H_6: All systems failure

For each of these hazards in turn, the effect on the collective swarm behaviour was considered, under the assumption that the hazard would only occur in a small number of individual robots [6]. Many of the hazards were determined to have a similar effect on behaviour of the swarm, and only three distinct effects were identified:

- E_1: Motor failure anchoring the swarm (serious)
- e_2: Lost robot(s) loose in the environment (non-serious)
- e_3: Robot collisions with obstacles or target (non-serious)

Swarm behaviour	H_1	H_2	H_3	H_4	H_5	H_6
Aggregation	–	e_2	–	–	e_2	–
Ad hoc network	–	e_2	–	–	e_2	–
Beacon taxis	E_1	e_2	–	–	E_1	–
Obstacle avoidance	E_1	e_2	e_3	–	E_1	–
Encapsulation	E_1	e_2	e_3	–	E_1	–

Table 2.1: Summary of the FMEA carried out by Winfield and Nembrini [6].

The full results of the FMEA are reproduced in Table 2.1. It was found that hazard H_1 (motor failure) would have the most detrimental effect on the collective behaviour, causing the swarm to become anchored around a faulty robot, thus preventing the beacon encapsulation task from being completed [6]. This is because the robots collectively attempt to move towards the beacon, but may turn back when connections to a stationary robot are lost (depending on the β threshold). Motor failure itself does not inherently cause problems, it is the fact that the failed robot is unaware that it has developed a fault, and continues to communicate with other robots [6].

Every other hazard analysed was shown to have non-serious effects, apart from hazard H_5 (control systems failure). This may also have an anchoring effect on the swarm if it causes the robot to become stationary, or to turn on the spot, however this is unlikely to happen given the simplicity of the robot controller, and that the default behaviour is just to move forward [6].

A surprising conclusion from the work of Winfield and Nembrini [6] is that despite their initial criticism of the swarm robotics literature for unsubstantiated claims of robustness, their study has shown that swarms are indeed generally quite robust, with only 6 out of 30 the hazard scenarios considered having a serious effect. Counter-intuitively, while hazard H_6 (complete failure) is the most serious failure that can occur in an individual robot, it is the most benign in terms of its effect upon the swarm's collective behaviours [6].

In summary, this FMEA shows that while swarm robotic systems are remarkably tolerant to the complete failure of individual robots, they may be vulnerable to partially failed robots that continue to operate in some limited capacity. Furthermore, the effect of these partially failed individuals may be so severe that they prevent the swarm from completing its task.

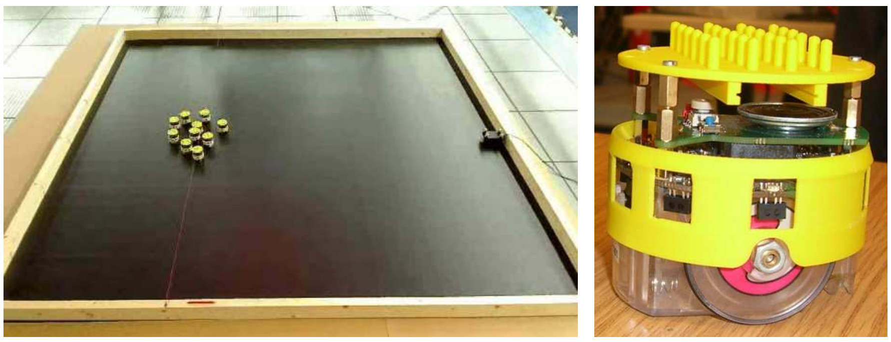


Figure 2.2: (a) Swarm of 10 e-puck robots using the ω-algorithm to perform emergent phototaxis towards an IR beacon on the right hand side of the arena. (b) e-puck robot fitted with a 'skirt' designed to prevent IR light from passing through its transparent body. Taken from [8].

2.1.1 *Swarms of real robots*

Bjerknes [7, 8] extended the work of Winfield and Nembrini [6] by validating their FMEA of the β-algorithm using empirical data from experiments with real robot swarms. Unfortunately, the β-algorithm depends on range-limited wireless communication with a well-defined boundary, which is difficult to achieve in physical hardware. To overcome this, Bjerknes [7] developed the ω-algorithm, which allows a swarm of real e-puck robots [29] to aggregate and perform phototaxis towards an infra-red (IR) beacon (as shown in Figure 2.2), through the combination of IR sensing and a simple timer. The ω-algorithm is explained in detail here, as it is used as a case study for testing the exogenous fault detection approach proposed later in Chapter 5.

Each robot in the swarm constantly emits IR light, allowing neighbouring robots to sense their presence. The robots are able to distinguish between IR light received from other robots, and IR light from the beacon, through the use of on-board signal processing [7]. Although the e-puck robots are actually capable of determining the distance and angle to the beacon, this information is deliberately discarded. As with the β-algorithm, each robot is only allowed to sense whether or not it is illuminated by the beacon, thus ensuring that the phototaxis behaviour is the result of emergent self-organisation [7].

The default behaviour of each robot is simply to move forward in a straight line. Each robot uses an *aggregation timer* to time the duration since it last avoided another robot, resetting its timer to zero whenever it makes an avoidance manoeuvre [7]. If the aggregation timer

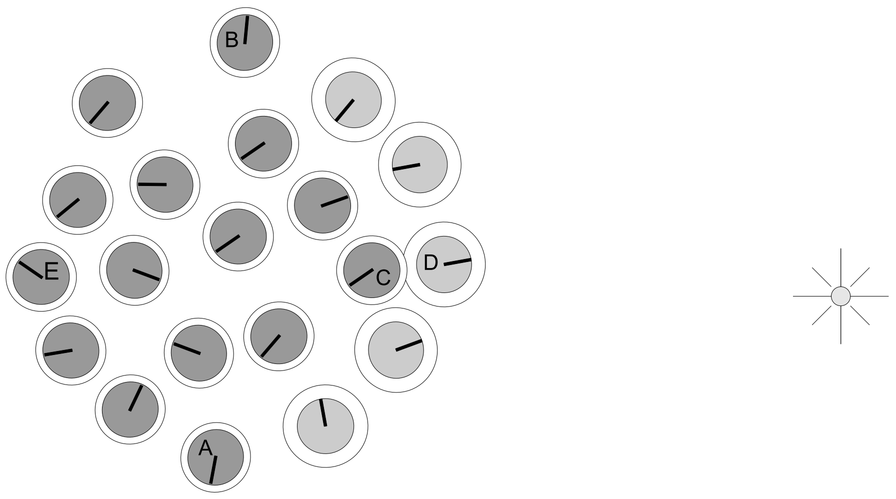


Figure 2.3: Illustration of symmetry breaking in the ω-algorithm. The robots illuminated by the beacon on the far right use a larger avoidance radius than occluded robots. Taken from [7].

ever exceeds a predefined threshold value ω, then the robot assumes it is moving away from the swarm and should turn back. When this happens the robot resets its aggregation timer to zero and turns to face its perceived centre of the swarm. This is only an estimate of the swarm's true centroid, calculated based on which of the IR sensors situated around the robot's body are activated [7]. If the robot travels beyond sensor range before attempting a coherence manoeuvre, there will be no sensory information to help it determine the direction back to the swarm. This should only ever occur if the value of ω is set too high, however to cope with borderline cases, the robot will simply turn 180° in the hope that it will eventually rejoin the swarm [7].

These two attraction and repulsion mechanisms are sufficient for the swarm to maintain stable aggregation, given an appropriate value for ω. However, in order for the behaviour of phototaxis to emerge, an additional *symmetry breaking* mechanism is required — robots alter their own avoidance radius according to illumination status, with robots illuminated by the beacon using a larger avoidance radius than those in shadow [7], as shown in Figure 2.3. In this example, robot *C* is occluded, so uses a smaller avoidance radius that allows it to move closer to other robots before an avoidance manoeuvre is triggered. Robot *D* is illuminated by the beacon, and its larger avoidance radius causes robot *C* to trigger the avoidance behaviour. This simple symmetry breaking mechanism results in the emergence of phototaxis at the collective level, despite individual robots being unaware of the distance or angle to the beacon [7].

Failure Mode and Effects Analysis

As with the work of Winfield and Nembrini [6], Bjerknes [7] only considered internal hazards for the emergent aggregation and phototaxis behaviours of the ω-algorithm, when implemented on real robots:

- H_1: Complete failure (equivalent to a total loss of battery power)
- H_2: Sensor failure (one IR sensor on one or many robots fails)
- H_3: Motor failure (IR sensors and emitters remain operational)

Consistent with the FMEA carried out by Winfield and Nembrini [6], Bjerknes [7] determined that complete failure and sensor failure would have little impact on the collective swarm behaviour, while motor failure could potentially be very harmful — again, having an anchoring effect on the swarm. This anchoring effect is due to the method that the robots use for estimating the centroid of the swarm, based on local IR sensing. When non-faulty robots perform a coherence manoeuvre and turn towards their perceived centre of the swarm, the IR light emitted by the stationary robot afflicted with motor failure contributes to the calculation of this estimate, as it is still considered part of the swarm [7]. This causes the swarm to remain tethered by the partially failed robot, preventing the non-faulty robots from making progress towards the beacon.

Bjerknes [7] validated this analysis by carrying out experimental trials with a swarm of 10 real e-puck robots. It was found that even in the case of a single robot experiencing motor failure, the swarm would often break apart due to the influence of the partially failed robot. The tracking data from an experiment shown in Figure 2.4 merits special attention. At time $t = 300$ a motor failure was introduced into the robot marked with a star. This has the effect of anchoring the swarm, causing two robots to break away. These robots partially occlude the rest of the swarm, making the symmetry breaking mechanism less effective, and hampering their escape from the partially failed robot [7]. At $t = 600$ the swarm remains anchored, but the two runaway robots have drifted slightly to one side, such that the swarm is now properly illuminated. This allows the symmetry breaking mechanism to work properly, and the rest of the swarm breaks free of the partially failed robot. At $t = 1100$ the swarm has managed to break away (leaving one anchored robot behind), and rejoins the two that initially broke away at the beacon [7].

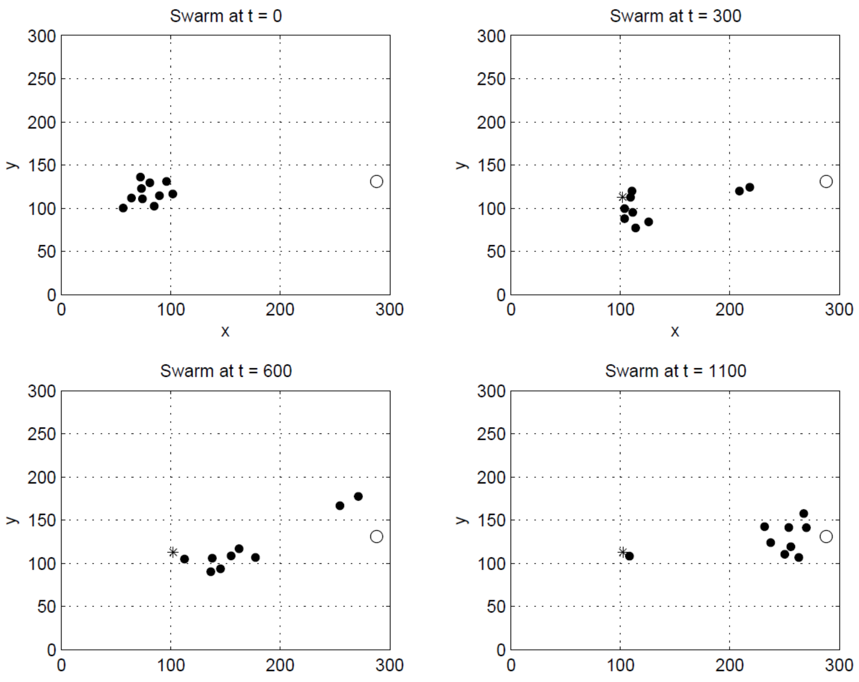


Figure 2.4: Tracking data from an experiment with real robots, demonstrating the effect of a partially failed robot on ω-algorithm phototaxis. A single robot with failed motors causes the swarm to become anchored and break apart. Taken from [7].

Bjerknes [7] refers to the ability of the swarm to eventually escape the influence of failed robots as 'self-repair', in the sense that the collective behaviour has recovered and is no longer affected. In order for the swarm to break away, the 'force' of phototaxis must overcome the anchoring force of the partially failed robots [8]. This is somewhat misleading terminology, since the partially failed robots have not been actually repaired — the swarm has simply left them behind. This behaviour is therefore a form of implicit fault tolerance that is inherent in the design of the ω-algorithm.

Unfortunately, the situation worsens when two robots simultaneously suffer motor failure — the swarm only reached the beacon in 6 out of 10 experiments conducted, and in those cases where the swarm succeeded, their progress was slowed significantly [7]. This demonstrates that implicit approaches to fault tolerance may be unable to cope with increased numbers of faulty individuals, thus necessitating an explicit approach to detecting and dealing with failed robots.

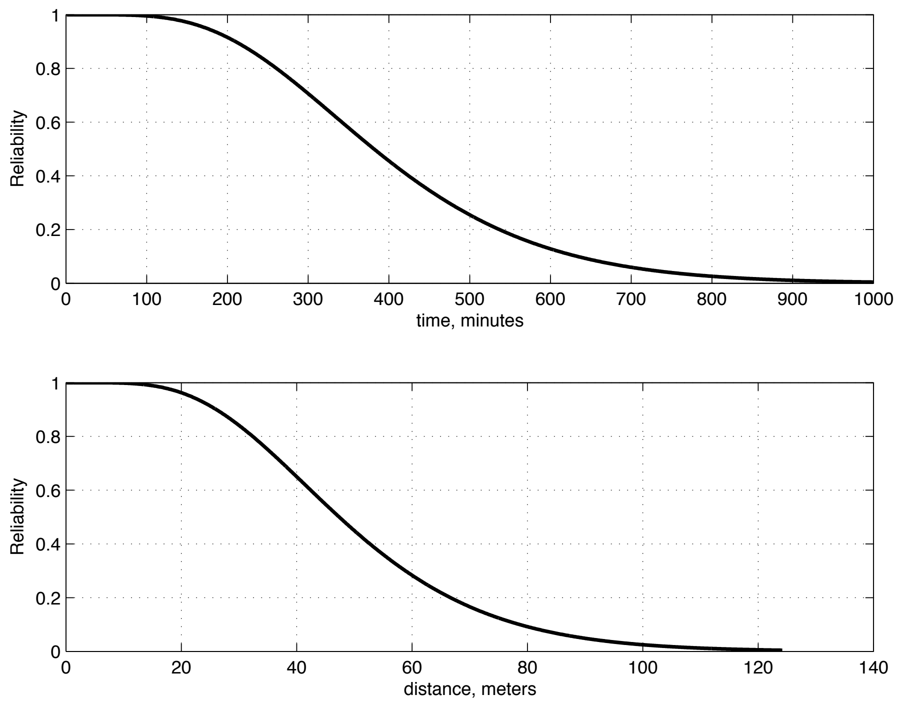


Figure 2.5: *Top*: Reliability as a function of time for a robot swarm modelled as a k-out-of-N system, where $k = 5$, $N = 10$, and a MTBF of 8 hours (480 minutes). *Bottom*: Reliability of the same swarm of 10 robots as a function of distance travelled, based on a mean phototaxis velocity of 12.42 cm per minute. Taken from [7].

Reliability analysis

In addition to performing experimental trials with swarms of real robots, Bjerknes [7] used reliability analysis to investigate the theoretical fault tolerance of swarm robotic systems. In the context of the ω-algorithm the *reliability* of the system is defined as the probability that the swarm will remain operational despite the failure of individual robots. Carlson and Murphy [30] collected failure type and frequency data from thirteen robots, representing three different manufacturers and seven robot models over a period of two years, and found that the mean time between failures (MTBF) was 8 hours. Although a follow-up study [31] with more recent data showed that the MTBF had improved to 24 hours, this still suggests that the reliability of mobile robots is concerningly low, and is especially problematic if a swarm robotic system is to be left unattended for several days.

Bjerknes [7] used a k-out-of-N reliability model to assess how the reliability of a swarm running the ω-algorithm will vary as a function of time. The swarm was mathematically modelled as a parallel system, where at least k robots must be operational for the swarm of

N robots to complete its task. Although partially failed robots were shown to have the most detrimental effect on the swarm behaviour, only completely failed robots were considered in this initial model. A value of $k = 5$ was used, as this was the minimum number of robots with which phototaxis was observed to be reliable during experimentation [7].

It was shown by Bjerknes [7] that with a MTBF of 8 hours, the reliability of a swarm of $N = 10$ robots declines rapidly after its first 100 minutes of operation, as shown in Figure 2.5. This figure also shows the reliability of the swarm as a function of the distance travelled, based on the mean velocity of phototaxis measured during experimental trials. The projected reliability shows that the swarm will only be able to travel just over 100 metres before completely failing, due to there being an insufficient number of functional robots left for phototaxis to emerge [7].

This is an important result, as it shows that even if partially failed robots could be detected by other robots in the swarm, leaving failed robots behind is not a scalable strategy. Instead, an explicit approach of fault detection, diagnosis, and recovery may allow partially failed robots to be repaired in some way, such that they may continue to contribute to collective behaviours that rely on emergence.

Scalability

Based on the results of this simple k-out-of-N reliability model, it seems as though the swarm could be made reliable (in the sense that it will still reach the beacon) for an arbitrarily long period of time simply by adding more robots, due to increased redundancy. Unfortunately, as shown by Bjerknes [7], this is not the case.

When partial failures are considered, the notion of swarm 'self-repair time' becomes important — defined as the time between the occurrence of a robot failure, and the time when the trailing non-faulty robot in the swarm escapes the influence of the failed robot. Bjerknes [7] argues that larger swarms will take longer to self-repair, because the velocity of phototaxis is inversely proportional to swarm size, and a larger swarm must move a greater distance before it self-repairs. Therefore, the *rate* of self-repair remains constant regardless of swarm size. However, although the MTBF for each individual robot also remains constant, the probability of failures occurring in the swarm will increase as the size of the swarm increases. Above a certain swarm

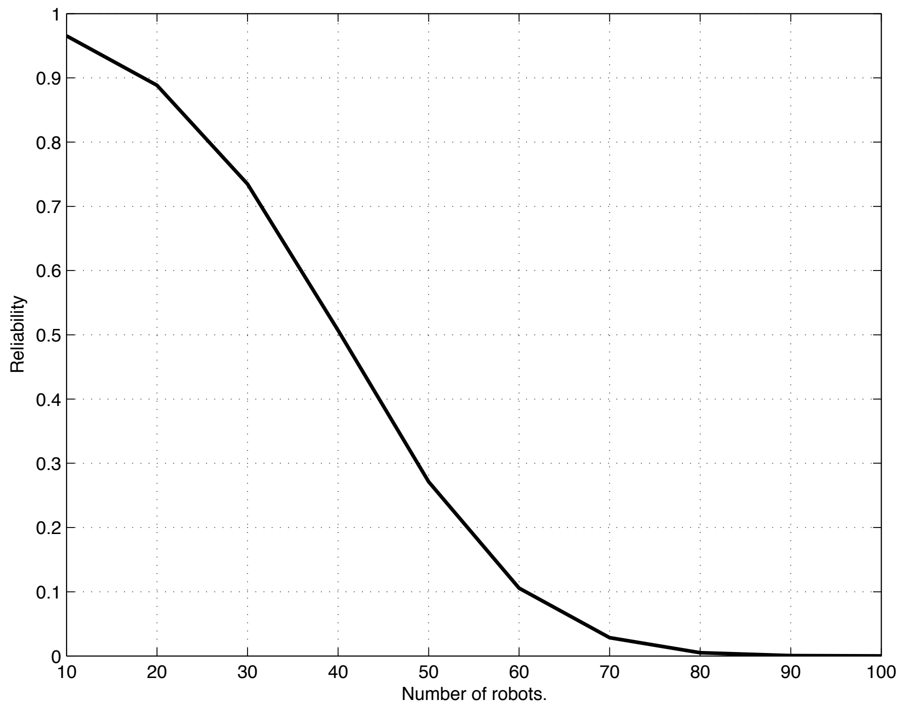


Figure 2.6: Reliability as a function of the number of robots in the swarm, based on a k-out-of-N model with $k = 0.9N$, a self-repair rate of 87.9 seconds, and a MTBF of 8 hours. Taken from [7].

size, the failure rate will overtake the self-repair rate, and the swarm will grind to a halt before it makes progress towards the beacon [7].

Bjerknes [7] demonstrates this analytically, by extending the k-out-of-N model to estimate how reliability would vary as a function of the number of robots in the swarm. The experimental trials with real e-pucks showed that a swarm of 10 robots could only self-repair with at most one partially failed robot, so k was set to 90% of N in the extended model. The average self-repair time during experiments with a swarm of 10 robots and a single robot experiencing motor failure was found to be 879 seconds, so the self-repair rate was set to 87.9 seconds. Figure 2.6 shows the projections of the extended reliability model. This analysis suggests that the reliability of the swarm would rapidly decrease as the number of robots increases, and that a swarm comprising as few as 100 robots would completely fail to operate.

As noted by Bjerknes [7], a number of assumptions are made in these reliability models, so the exact values will likely be different in practice. However, the general trend is that as the number of robots increases, the longer it will take the swarm to escape the anchoring effect of partially failed robots, to the point that it will be completely non-functional if the swarm is too large. Although the measure of self-repair time is specific to the emergent swarm phototaxis case study, the results generalise to other swarm robotic systems where the failure rate may overtake some notion of swarm 'self-repair time' as the number of robots increase [8]. The use of high reliability hardware components will result in a longer MTFB, thus improving the overall reliability of the swarm, but the system will eventually fail without an explicit approach to fault tolerance.

2.1.2 *Summary*

The work of Winfield and Nembrini [6] and Bjerknes [7] show that it is not safe to assume that fault tolerance and scalability are inherent properties of all swarm robotic systems, simply by virtue of local sensing and decentralised control, as is often claimed by swarm robotics researchers. However, such assertions are perhaps due to a common assumption that 'failure' only means complete failure, which has been shown to have little influence on swarm behaviour. When partial failures are considered, such as motor failure, problems may arise that cannot simply be solved by adding more robots.

Bjerknes and Winfield [8] argue that by demonstrating assumptions of swarm robustness and scalability are false in one specific case, it casts their general validity into doubt. It is important to note, however, that the effect of partial failures will be more severe in swarms whose behaviours rely on self-organisation and emergence. In swarm robotic systems where individual robots work independently in parallel to increase the speed of task completion, the effect of partial failures may be less extreme [8]. Nevertheless, aggregation and collective motion are fundamental swarm behaviours [20], therefore partial failures are likely to cause serious problems for many robot swarms. Winfield and Nembrini [6] conclude that:

“
1. analysis of fault-tolerance in swarms critically needs to consider the consequence of partial robot failures.
2. future safety-critical swarms would need designed-in measures to counter the effect of such partial failures.
”

2.2 IMMUNE-INSPIRED FAULT TOLERANCE

Bjerknes and Winfield [8] suggest that implementing some kind of distributed Artificial Immune System (AIS) [32] may be an appropriate and systematic approach to solving the problem of partially failed robots harmfully influencing a swarm, citing initial work in this direction by Timmis et al. [33]. Winfield and Nembrini [6] also encourage the development of immune response models, as a method of affording swarms increased fault tolerance. Bjerknes and Winfield [8] argue that this is an important new direction in the field of swarm robotics, and that any large-scale self-organising system must have an active approach to dealing with failed or rogue units.

2.2.1 *Natural immunity*

Before discussing the application of AIS to robot swarms, it is instructive to first examine immunity in natural systems. This section will begin with a brief overview of individual immunity, before discussing social/collective immunity in social insect colonies, and how general principles might be extracted from these natural systems that can be applied in an engineering context.

Individual immunity

The vertebrate immune system comprises a variety of biological components and mechanisms that fight against harmful microorganisms to protect the body from disease [34]. The immune system can broadly be split into two components — innate and adaptive — with respect to the ability of each to respond and react to invading pathogens. The innate immune system acts as the body's first line of defence, and is able to provide a fast response to invading pathogens. However, its immune repertoire is limited because it is coded into the organism's genome at birth via heredity [34].

In contrast, the adaptive immune system offers what is often referred to as *acquired* immunity — it is able to learn and adapt over the lifetime of the individual, to protect the body against pathogens not known about at the organism's birth, which the innate immune system is unable to detect [34]. However, the response time of the adaptive immune system is much slower than that of the innate immune system, due to the way in which it detects pathogens. It is therefore necessary for the innate and adaptive immune systems to work together, to provide a proper defence against disease [34].

Cohen [35] gives a dynamical systems perspective of the vertebrate immune system, describing it as a *cognitive* system that not only fights infection, but also performs maintenance, helping an organism maintain homeostasis. Cohen [35] argues that the immune system provides three properties that allow it to carry out maintenance:

“
1. Recognition - see what is right and what is wrong
2. Cognition - interpret signs, evaluate results, and make decisions
3. Action - actually do the job
”

These three properties of the immune system map closely to an explicit fault tolerance approach of fault detection, diagnosis, and recovery. If similar properties could be replicated for a robot swarm, then it should be possible to achieve swarm-level homeostasis, and thus fault-tolerant behaviour. A homeostatic swarm would maintain a stable state until perturbed by a faulty robot, then attempt to return to a stable, functioning state.

A swarm robotic system could therefore be considered as a multicellular ‘organism’, that requires an ‘immune system’ to maintain fault tolerance at the collective level. Innate immune defences would correspond to pre-programmed fault tolerance mechanisms that are based on prior knowledge of possible failure modes and effective recovery actions, whereas adaptive immunity would allow the swarm to recover from failure modes not anticipated before deployment.

Social/collective immunity

Cremer et al. [36] were the first to describe *collective immunity* in swarms of social insects, instead of focusing on individual immunity. Over millions of years, social insect colonies have evolved collective immune defences against parasites. These ‘social immune systems’ result from the cooperation of the individual group members to combat the increased risk of disease transmission that arises from sociality and group living. Collective defences may be both prophylactic (preventative measures) and activated on demand, and consist of behavioural, physiological and organisational adaptations of the colony that prevent parasite entrance, establishment and spread [36].

Cremer et al. [36] explain that the colony must select an appropriate response based on external information about the parasite (type, dose,

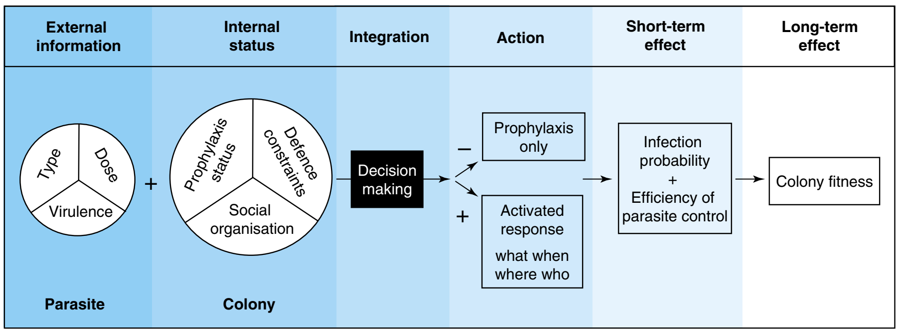


Figure 2.7: Regulation of the social immune response. Taken from [36].

virulence), and internal status of the colony (prophylaxis status, social organisation, defence constraints):

> *The colony member will not only have to decide which defence mechanism to employ, but whether or not to start a response at all, when and where to start it, and who should be responsible for the defence mechanism and who should be protected by it.*

It is important that an appropriate response is selected, as both under-reacting or over-reacting could be costly. Cremer et al. [36] also describe 'triage' behaviour, where the value of colony members is taken into account when deciding who to help first in the event of several infected individuals. This social immune response is illustrated in Figure 2.7, which shows collective decision-making at its core.

Unfortunately, how these collective decisions are made is not well-understood [36]. This is a shame, as there is a clear analogy between robot swarms and social insect colonies, and concepts from social immunity could potentially be applied in a swarm robotics context. In particular, reasoning about failed robots via collective decision-making would be a robust approach to distributed fault recovery.

Taking inspiration from biology

Stepney et al. [37] propose a conceptual framework that offers an approach to understanding natural systems, such that inspiration can be taken from them in a principled manner. The framework takes an interdisciplinary approach, which begins by probing the biological system to be used as inspiration, through observations and experiments, as shown in Figure 2.8. This allows a partial view of the system to be obtained, which is then used to build abstract mathematical or computational models of the biology. Through the execution and

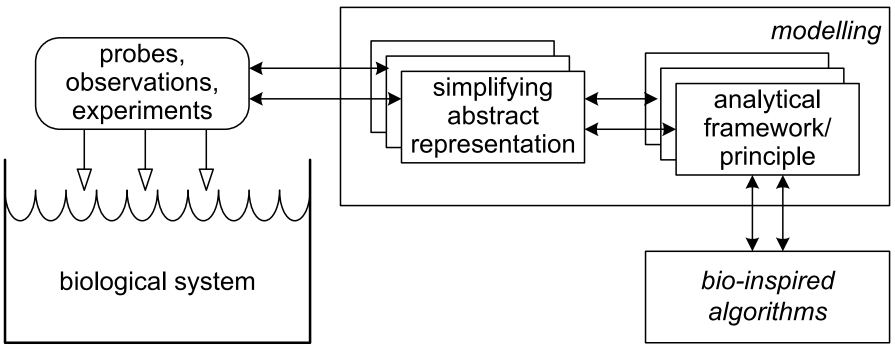

modelling
probes, observations, experiments
simplifying abstract representation
analytical framework/ principle
biological system
bio-inspired algorithms

Figure 2.8: Conceptual framework that allows a principled approach to be taken when designing bio-inspired algorithms. Taken from [37].

validation of these models, insight may be gained into the underlying biological processes, which can then be used in the construction of bio-inspired algorithms [38].

It is important to note that applying the conceptual framework proposed by Stepney et al. [37] in a swarm robotics context does not necessarily mean that the bio-inspired system produced would fall into the category of Nature-inspired Minimalist Swarm Robotics proposed by Sharkey [26]. It is entirely possible to be inspired by biology without being constrained by it — the conceptual framework simply ensures that the inspiration is sound, and is grounded in a proper understanding of the underlying biological system.

This conceptual framework could be used to take inspiration from natural systems that exhibit individual and/or collective immunity, in a principled manner, to engineer fault tolerant swarm robotic systems. For example, it may be possible to model social insect colonies and extract general principles about their behaviour during a parasitic infection, which could then be used as inspiration when developing an artificial analogue of social immunity for robot swarms.

However, Trianni [39] argues that there may be cases where it is not possible to take inspiration from natural systems. This is because the physical embodiment of individuals, or the type of possible interactions between them, may be so different from the target swarm robotic system that the required individual behaviours cannot be implemented in an artificial context [39]. For example, in social insect colonies, odour likely plays an important role in the detection of parasitic infections [40], for which there is no direct artificial analogue.

2.2.2 *Artificial Immune Systems*

Artificial Immune Systems are computational techniques that attempt to bridge the gap between immunology and engineering [38], defined by de Castro and Timmis [32] as:

> “*adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving.*”

In the past few years, AIS have found applications in various areas, including pattern recognition [41], optimisation [42], computer security [43], and intrusion detection [44]. A recent survey by Bayar et al. [45] also shows that AIS have been used extensively for fault diagnosis, detection, and recovery in the context of manufacturing systems.

Timmis et al. [46] suggest that AIS could be applied to solving problems associated with long-term autonomy in robot swarms, using the SYMBRION/REPLICATOR projects [47] as a case study. In particular, the grand challenge proposed by Kernbach et al. [48], in which a collective of 100 robots, would be left unattended in a room containing various power sources, for 100 days, with the aim of surviving for as long as possible. Specifically, Timmis et al. [46] argue that AIS techniques could be used to detect faults in individual robots, and even predict errors before they occur, in order to improve swarm longevity.

Indeed, as part of the SYMBRION project, Mokhtar et al. [49] take an AIS approach to the endogenous detection of faulty sensors, using a modified version of the Dendritic Cell Algorithm [50], and show that it is able to provide on-line error detection in robotic systems.

Ismail et al. [51] have also developed a collective behaviour inspired by granuloma formation in the innate immune system, which affords a swarm robotic system the ability to contain and repair partially failed robots. This algorithm is reviewed in detail in the following sections, as it provides a direct solution to the anchoring problem highlighted by Winfield and Nembrini [6] and Bjerknes [7].

Granuloma formation

Granulomas are structures that form in response to the pathogenic infection of individual cells, particularly infections by tuberculosis and Leishmaniasis. Immune cells called T-cells attempt to contain the infected cell, and prevent the pathogen spreading and infecting other

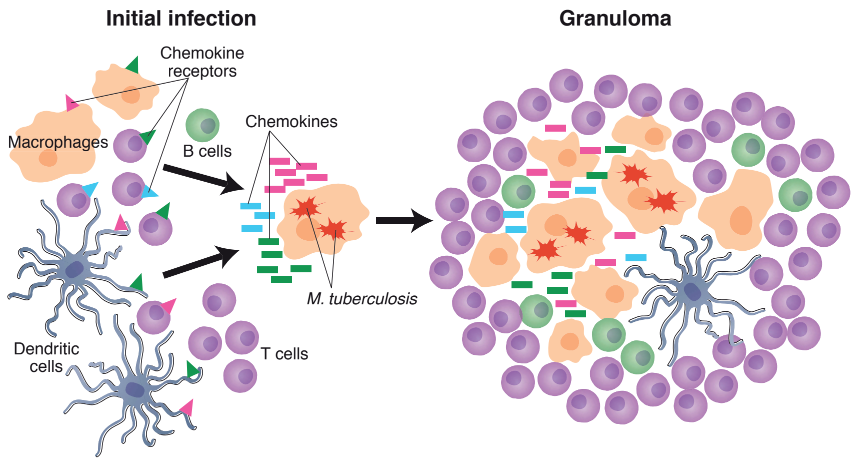


Figure 2.9: Illustration of granuloma formation initiated by the innate immune system in response to a tuberculosis infection. Taken from [53].

cells [52]. There are three types of cells primarily involved in granuloma formation: macrophages, T-cells and cytokines. Ismail et al. [51] describe the three main stages of granuloma formation as:

“

1. T-cells are primed by antigen presenting cells.
2. Cytokines and chemokines are released by macrophages, activated T-cells and dendritic cells. The released cytokines and chemokines attract and retain specific cell populations.
3. The stable and dynamic accumulation of cells and the formation of the organised structure of the granuloma.

”

As shown in Figure 2.9, macrophages engulf the pathogen and 'eat' it in a process referred to as phagocytosis, in an attempt to prevent the spread of the disease. However, the macrophage may become infected by the pathogen, and it will duplicate within the cell. This in turn leads to cell lysis — where the structure of the cell breaks down — allowing the pathogen to spread to other cells.

When a macrophage becomes infected, it will release a signal that attracts other macrophages and T-cells to the site of infection. These form a wall around the infected macrophages, encasing them and isolating them from uninfected cells, leading to the formation of the granuloma. The cells inside the granuloma will eventually die, resulting in the removal of the pathogenic material. The isolation of infected cells in this way allows the robustness of the immune system to be maintained [51].

Immuno-engineering

Ismail et al. [51] argue that there exists a natural analogy between the early stages of granuloma formation, for the removal of pathogenic material, and the potential repair of failed robots in a swarm. This is explored in the context of *immuno-engineering*, a new discipline defined by Timmis et al. [54] as:

“ *the abstraction of immuno-ecological and immuno-informatics principles, and their adaptation and application to engineered artefacts (comprising hardware and software), so as to provide these artefacts with properties analogous to those provided to organisms by their natural immune systems.* ”

This offers a principled approach to understanding the problem domain and the immune processes under study, adopting the conceptual framework proposed by Stepney et al. [37] (discussed in Section 2.2.1). Following this process, Ismail et al. [51] used Agent Based Modelling to simulate the process of granuloma formation, so that the core principles could be formalised and applied in an engineering context. The four principles that were extracted are:

“
1. Communication between agents in the system is indirect, consisting of a number of signals to facilitate coordination of agents
2. Agents in the system react to defined failure modes in a self-organising manner
3. Agents must be able to learn and adapt by changing their role dynamically
4. Agents can initiate a self-healing process dependent on their ability and location

”

Granuloma formation algorithm

Using these four principles, Ismail et al. [51] develop the *granuloma formation algorithm* (illustrated in Figure 2.10) in an attempt to create a ‘self-healing’ swarm robotic system. Under the assumption that robots are able to proprioceptively detect faults in themselves, and communicate their distress to other robots, Ismail et al. [51] focus on collective self-repair mechanisms that address the anchoring issue highlighted by Winfield and Nembrini [6] and Bjerknes [7].

Following on from the work of Bjerknes [7], Ismail et al. [51] simulate a swarm of robots performing phototaxis using the ω-algorithm (described in Section 2.1.1), considering a specific type of failure in a robot's power unit that causes a sudden loss of stored energy. The amount of remaining energy is sufficient for communication, but insufficient to power the robot's motors, resulting in a loss of mobility.

This failure mode allows the anchoring issues due to motor failure observed by Bjerknes [7] to be reproduced, whilst also allowing the robot to be 'repaired' by recharging its battery. In order for a faulty robot to be repaired, non-faulty robots must dock with it at predefined points around its body and donate energy from their own batteries, until it has enough energy to resume normal operation.

Once a robot has detected that it has suffered a power failure, it sends a distress signal to neighbouring robots. Those that receive this distress signal will negotiate with the faulty robot, to decide which of them should donate their energy. The amount of energy required to repair a failed robot is predefined, and functional robots will only donate energy if they have more than a minimum threshold level, so it may be necessary to recruit multiple donors to share the burden.

The donors will then be attracted to the faulty robot, analogously to the way T-cells are attracted to an infected macrophage emitting cytokines during granuloma formation, and surround it to 'isolate' it from the rest of the swarm. After repairing the failed robot, the donors resume phototaxis and continue towards the beacon.

Ismail et al. [51] show that this granuloma formation algorithm allows the swarm to reach the beacon despite the simultaneous partial failure of up to five robots. The algorithm provides the swarm with an active 'self-healing' ability in addition to the inherent 'self-repair' property of the ω-algorithm demonstrated by Bjerknes [7]. As discussed in Section 2.1.1, leaving failed robots behind is not a scalable strategy, and partially failed robots that adversely affect swarm behaviour must be dealt with explicitly. This immune-inspired recovery mechanism facilitates the repair of partially failed robots, allowing them to contribute to the emergent behaviour of phototaxis, therefore improving the overall reliability of the swarm robotic system.

2.2.3 *Summary*

The granuloma formation algorithm developed by Ismail et al. [51] represents a useful application of AIS to improving the fault tolerance of robot swarms, and provides an effective solution to the anchoring problem highlighted by Bjerknes [7]. Unfortunately, the generalisability of the algorithm is limited as it deals with a very specific failure

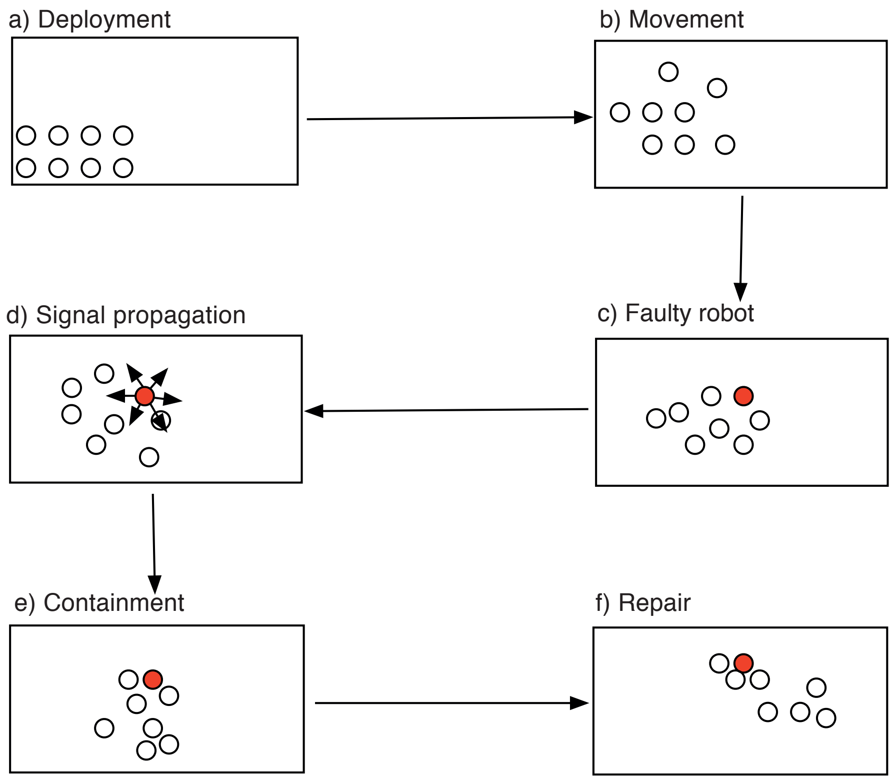


Figure 2.10: Illustration of a swarm robotic system executing the granuloma formation algorithm to repair a faulty robot. Taken from [55].

mode, and is restricted to implementation on robotic platforms that are able to share their stored energy with other robots in the swarm.

The algorithm could perhaps be used as part of an 'artificial immune repertoire' that a robot swarm may call upon once a fault has been detected. With an entire suite of self-repair mechanisms at their disposal, robot swarms may be afforded artificial 'immunity' at the collective level, by instantiating the appropriate 'immune response' depending upon the type of failure detected. The granuloma formation algorithm constitutes a suitable immune response to power failures, but some other recovery mechanism would be required for the repair of sensor, or motor failures. Some of the general concepts from the algorithm may still be applicable though, such as the recruitment strategy, when recovering from other types of failure.

Before any fault recovery mechanisms can even be initiated, it is first necessary to detect the presence of a fault and diagnose the cause of the failure. In a first step towards developing some form of distributed swarm immunity, this thesis proposes a solution to the problem of exogenous fault detection in robot swarms — the output of which would feed into diagnosis and recovery mechanisms.

AIS have recently been applied to the problem of fault detection in swarm robotic systems, taking both endogenous [56] and exogenous [12] approaches (reviewed in detail in Section 2.3). However, the research presented in this thesis does not take an immune-inspired approach. Nevertheless, in principle, it could be combined with diagnosis/recovery mechanisms based on AIS techniques (such as the granuloma formation algorithm) once a fault has been detected.

2.3 FAULT DETECTION

Fault detection is the process of determining whether or not a fault has occurred in a system. This is typically achieved by detecting deviations from normal behaviour, which are assumed to be due to the presence of a fault [57]. Christensen [58] broadly categorises fault detection in the context of collective robotic systems into *endogenous* and *exogenous* approaches (discussed in Sections 2.3.1 and 2.3.2). Endogenous fault detection refers to the ability of robots to proprioceptively detect faults in themselves, whereas exogenous fault detection allows robots to detect the presence of faults in each other.

Regardless of whether an endogenous or exogenous approach is taken, some method of distinguishing between normal and abnormal system behaviour is required in order to detect the presence of faults. This can be achieved using anomaly detection techniques [59], which detect observations of behaviour that do not conform to an expected pattern. These techniques take three different forms that assume varying levels of prior knowledge:

1. Supervised: Examples of both normal and abnormal behaviour are available
2. Semi-supervised: Examples of normal behaviour are available
3. Unsupervised: No examples of normal or abnormal behaviour are available

Supervised approaches essentially solve a binary classification problem, whereas semi-supervised and unsupervised approaches perform outlier detection [60].

While supervised approaches may initially seem attractive, the number of potential faults in robotic systems is very large [24], which makes it difficult to obtain a comprehensive set of examples of abnormal behaviour. This is exacerbated by compound effects of multiple simultaneous faults on a system's behaviour, so accounting for every possible permutation of failure scenarios is often infeasible. Semi-supervised or unsupervised approaches are therefore more commonly used, as they only require examples of normal system behaviour.

Examples of normal and/or abnormal system behaviour may be obtained using either *model-based* (analytical) or *data-driven* (model-free) methods [58]. In model-based approaches, a model of the system's expected behaviour is constructed, which the system's actual behaviour during operation can then be compared against. If a significant discrepancy between the expected and observed behaviour is detected, this may indicate the presence of a fault.

In contrast, data-driven approaches do not require the explicit construction of a model representing the system's expected behaviour. Instead, a system's normal behaviour is learnt using data collected during its operation, which can then be used to detect faults [58].

Whichever method is used, the misclassification of a system's behaviour can be costly. For example, if a non-faulty robot is erroneously classified as faulty this may result in the instigation of unnecessary collective recovery mechanisms, thus wasting time and energy. Conversely, partially failed robots left undetected may have a detrimental effect on swarm behaviour, as discussed in Section 2.1. Therefore, it is important that any fault detection system is designed such that the desired trade-off between these two cases can be tuned based on their relative importance.

2.3.1 *Endogenous fault detection approaches*

Endogenous fault detection is often self-contained to an individual robot, therefore many examples can be found for single-robot systems. A common approach is to use special-purpose hardware to detect component faults, such as rotary encoders for determining whether a robot's wheels are turning in response to control signals. However, as argued by Christensen [58], adding special-purpose fault detection hardware increases the cost, weight, and complexity of a robotic platform, which is undesirable.

This is especially problematic in the context of swarm robotic systems, as each individual robot must be cheap to manufacture, otherwise the production of large swarms will be prohibitively expensive. Furthermore, an increase in the size, weight, or power consumption of each individual robot may limit the flexibility of the swarm.

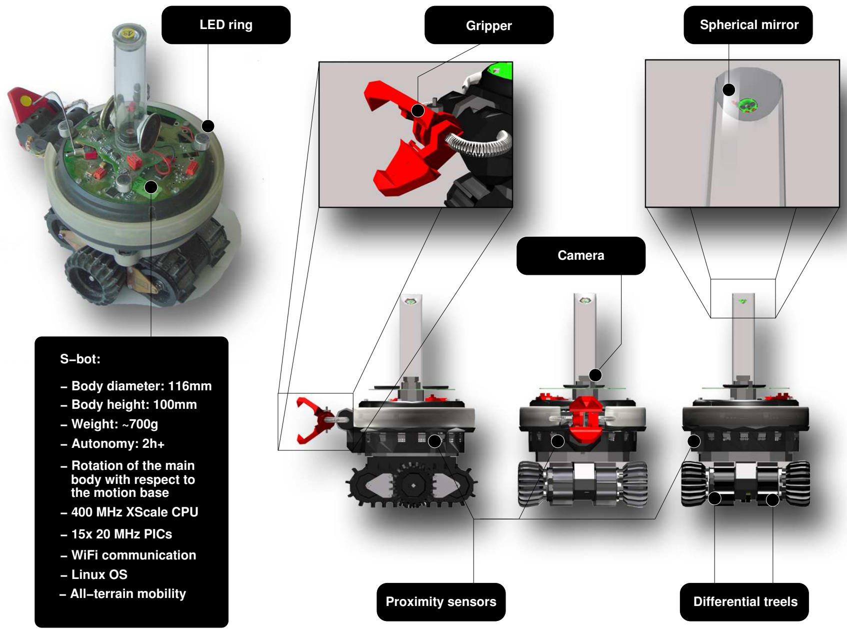


Figure 2.11: The *s-bot* robot platform. Taken from Christensen [58].

In addition, the fault detection hardware itself would be subject to faults [58]. Instead of using additional hardware to aid the detection of faults, there are a number of software-based approaches that can be used, a couple of which are reviewed in detail in this section.

Automatic synthesis of fault detection modules

Christensen et al. [61] use Time-Delay Neural Networks (TDNNs) [62] to automatically synthesise task-dependent fault detection modules for *s-bot* robots [63] (shown in Figure 2.11). This robot platform has been used in a significant body of swarm robotics research, so the work in [61] provides an example of endogenous fault detection that can be performed in real-time on swarm robotic hardware.

The fault detector is a separate software module that passively monitors information that flows in and out of the robot controller, and detects whether the robot is performing the pre-specified task correctly, or if a fault is affecting its behaviour. The robot controller is treated as a black box, and the TDDN learns to approximate the function that maps current and past sensory inputs, and control signals to the actuators, to a faulty or non-faulty classification.

The TDNN is trained using a supervised approach, based on data collected from experimental runs while the robots are operating normally, and after faults have been injected. Sensor readings, control signals, and whether or not the robot is faulty, are recorded at each control cycle. The sensor readings and control signals are used as input to the TDNN, which outputs a continuous value in the interval (0, 1). Output values above a certain threshold result in a faulty classification, otherwise the robot is classified as non-faulty. Back-propagation is used to train the TDNN such that it minimises the error between the predicted and true classification of the robot.

Three different robot tasks are considered, to test the performance of endogenous fault detection:

FIND PERIMETER: An *s-bot* must use its ground sensors to find the perimeter of a dark patch on the arena floor, and then follow this perimeter. A light source is placed in the centre of the floor patch, which the robot can sense with its on-board light sensors, providing the robot with a frame of reference.

FOLLOW THE LEADER: One *s-bot* (the leader) performs a random walk while another *s-bot* (the follower) follows at a short distance. If the follower falls behind, the leader will wait for it to catch up. The *s-bots* perceive each other using their omnidirectional cameras, and their IR sensors are used for obstacle avoidance. Faults are only injected into the follower.

CONNECT TO S-BOT: One *s-bot* must connect to another *s-bot* that is stationary, using its gripper. After making a successful connection, the connecting *s-bot* must wait 10 seconds before disconnecting, reversing, and attempting to connect to the stationary *s-bot* again. The connecting *s-bot* uses its camera to determine the location of the other *s-bot*. Only the connecting *s-bot* is injected with faults.

Faults are injected (via software) into the *s-bot*'s treels (combined tracks and wheels). There is a 50% chance that both treels will be affected, instead of just one of them. Two types of faults are considered: *stuck-at-zero* and *stuck-at-constant*, and have an equal probability of being injected. A *stuck-at-zero* fault causes the motor driving an affected treel to stop working. For *stuck-at-constant* faults, the motor speed of an affected treel is set to a random value, and remains fixed regardless of any control signals received. The TDNN learns to recognise sensor readings and actuator control signals that correspond to these kinds of faults, and will output a faulty classification if they are detected at run-time.

Christensen et al. [61] show that their TDNN fault detector is able to reliably detect the injected faults across all three tasks, with varying levels of latency (in the order of seconds) and false positives. However, *stuck-at-zero* faults cannot be detected during the waiting phase of the *connect to s-bot* task, because no signals are sent to the motors by the control program, so there is no way to tell that the treels have stopped working. In general, this approach is unable to detect faults that have no effect on a robot's behaviour.

Christensen et al. [64] extend this work to consider sensor faults in the *find perimeter* task. Faults are either injected into the *s-bot*'s front ground sensor, or two of the robot's front light sensors. Christensen et al. [64] demonstrate that TDNN fault detectors can be synthesised to reliably detect these sensor failures. It is also shown that a single TDNN fault detector can be trained to recognise faults in both the sensors and actuators. In addition, Christensen et al. [64] trained a single fault detector on three variations of the *connect to s-bot* task — the original task, plus one where the *s-bot* being connected to is moving, and another where three *s-bot*s are already connected in a line. It is shown that the TDNN is still able to reliably detect the faults, and Christensen et al. [64] conclude that it is possible to train fault detectors that are robust to variations in the task.

The main issue with this endogenous fault detection approach, is that the TDNN learns to discriminate between faulty and non-faulty information flow using supervised learning — both normal and faulty behaviour of the *s-bot* robots must be recorded and used to train the fault detector prior to deployment. The synthesised fault detectors are therefore entirely task-dependent, and any new tasks would have to be trained for offline. This precludes the possibility of the robots adapting their behaviour online, as any behaviour not learned during the training phase may be detected as a fault. Every fault to be detected must also be known *a priori*, so that the TDNN can be trained to recognise the faults at run-time. Given that the TDNN is solving a binary classification problem rather than performing anomaly detection, any faults not trained for may be confused with normal behaviour.

Another problem is that unexpected sensor readings and control signals may be caused by other robots in the environment. For example, in the *follow the leader* task, the follower *s-bot* may endogenously misclassify itself as faulty if the leader *s-bot* develops a fault. Similarly, encountering a third robot during this task, the occurrence of which has not been trained for, may also result in erroneous endogenous fault detection. Therefore, its seem that this approach may not work

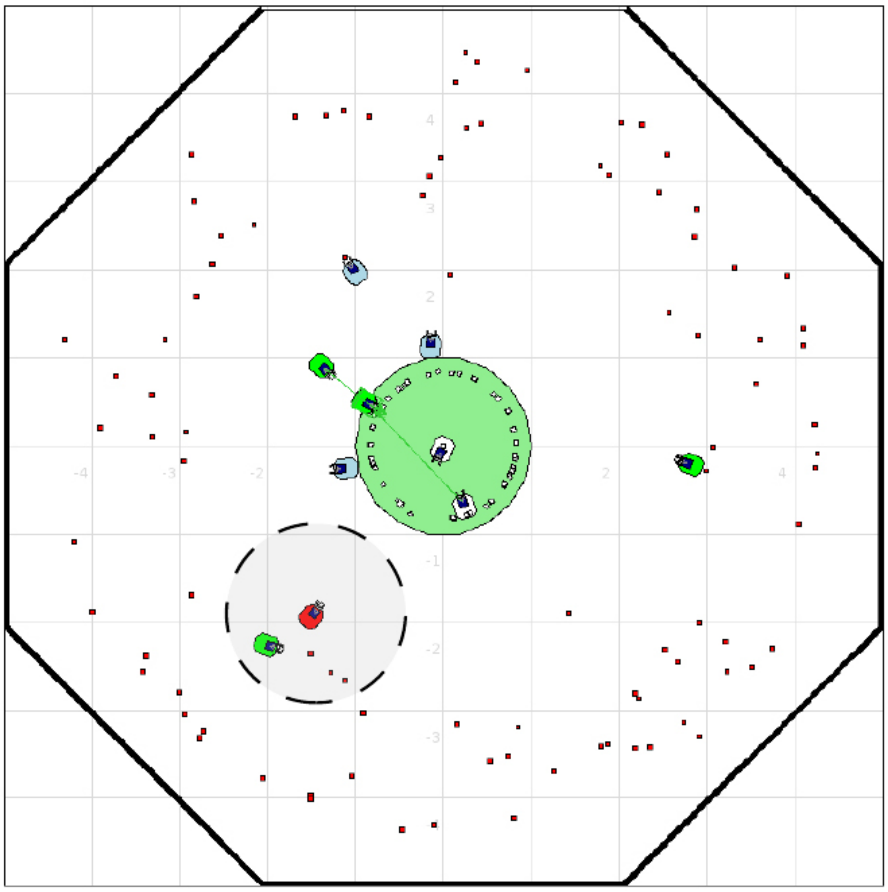


Figure 2.12: Screenshot of the foraging simulation, containing a swarm of 10 robots. Robots at the base are white, those searching for objects (represented by squares) are light blue, those with an object in their grippers are green, and the faulty robot is red. Taken from Lau [65].

well in a swarm context, as it would be infeasible to train for every possible scenario that a robot may find itself in, especially if it is to be deployed in an environment that is unknown ahead of time.

Adaptivity to dynamic environments

Most endogenous fault detection approaches only make use of the data gathered from a single individual — the robot attempting to detect the presence faults in itself. Lau [65] argues that this is an under-utilisation of the data available in a swarm robotic system, and that data from other robots within the local neighbourhood could be used to make endogenous fault detection more efficient.

Using a foraging task as a case study (shown in Figure 2.12), Lau et al. [56] define metrics that can be used to measure the performance of a robot with respect to the task: objects collected, energy used, and distance travelled. An individual robot could attempt to detect anomalies in its own performance based on these metrics, which may indicate the presence of a fault. However, the spatial distribution of the objects, and their availability, is influenced by the performance of the other robots in the swarm. Therefore, differences in performance due to faults in the individual, and those due to changes in the environment must be distinguished [56]. For example, a robot may collect fewer objects than usual either because it is faulty, or because the other robots in the swarm have already collected most of the available objects in the local area.

Lau et al. [56] develop a data-driven approach that allows each individual to cross-reference its performance against that of neighbouring robots in the swarm, to determine whether it is operating correctly. Under the assumption that the swarm is homogeneous, the performance of any given robot should be similar to that of other robots in the same local area of the environment. Therefore, if a robot detects that its own performance is significantly different to that of its neighbours, then it may have developed a fault.

To perform endogenous fault detection, an individual robot receives foraging performance data from its immediate neighbours, and models the distribution of values using the Receptor Density Algorithm (RDA) [66] — an anomaly detection algorithm inspired by the T-cell signalling mechanism in the immune system. Based on the assumption that the majority of neighbouring robots are non-faulty, the robot will classify itself as faulty if its own performance does not fit the distribution of 'normal' performance.

Lau et al. [56] considered three different types of motor failure, and demonstrated that their approach allows partially failed robots to endogenously detect the presence of faults in themselves. Lau et al. [67] later extended this work to improve performance when faced with multiple simultaneously faulty robots in the swarm, by only considering data from control cycles where performance data for at least two neighbouring robots was available, thus increasing the probability that the majority of neighbouring robots will be non-faulty. Performance was also shown to improve with swarm size, as more non-faulty robots are available for social comparison [67].

Despite the assumption that data from neighbouring robots will represent non-faulty performance, this approach is essentially unsupervised because no labelled examples of normal behaviour are available to the classifier. Although this allows the model of 'normal' behaviour to continuously adapt to dynamic environments, if the majority of an individual's neighbours are faulty, then it will misclassify itself as non-faulty based on the similarity of foraging performance.

Problems with endogenous fault detection

It may seem as though the problems associated with partial failures discussed in Section 2.1 could be solved by the incorporation of endogenous fault detection mechanisms. If individual robots can proprioceptively detect when they have developed a fault, they could then alert neighbouring robots so that recovery mechanisms can be initiated. However, in some cases, it may not even be possible for the robot to identify failure in itself. Christensen [58] provides examples of faults that cannot be detected by the robot in which they occur:

“
- a dead battery
- a short-circuit on the main board
- a bug that causes the on-board software to hang
”

Even if a robot is able to detect that it has developed a fault, it may be unable to signal this to the rest of the swarm if there is a fault in its communications hardware. Consequently, endogenous methods alone are not sufficient to ensure that all faults are reliably detected.

2.3.2 *Exogenous fault detection approaches*

Christensen et al. [10] argues that the robustness of swarm robotic systems can be improved through the use of exogenous fault detection, as it does not rely on potentially faulty robots being able to detect faults in themselves and alerting nearby robots. Exogenous fault detection in a swarm robotics context is particularly attractive, as it leverages the swarm's ability to perform many independent classifications in parallel, which could then be combined to arrive at a collective consensus about which robots are truly faulty.

In comparison to endogenous fault detection, exogenous fault detection is a much harder engineering problem, because the fault itself may not be outwardly observable — only its effect on a robot's behaviour may be detectable. It may only be possible to detect fault in a robot's sensors once the robot has been observed to collide with an obstacle, for instance. This is exacerbated by the fact for some faults there may be a delay between the occurrence of the fault and observable symptoms [61]. For example, a motor fault that causes a robot's

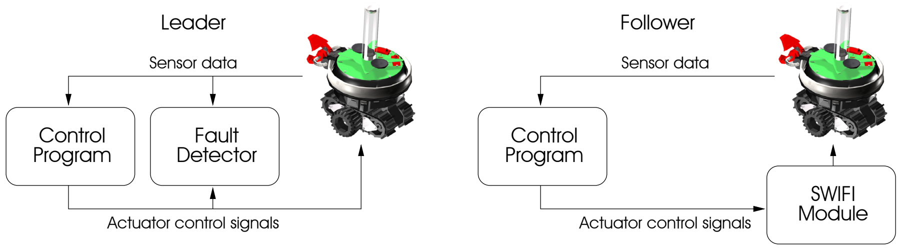


Figure 2.13: Exogenous fault detection architecture for the *follow the leader* task. The Software Implemented Fault Injection (SWIFI) module is used to inject faults in the follower's treels. Taken from Christensen et al. [9].

wheels to stop can only be detected if enough time has passed for an absence of movement to be detected. Furthermore, the method of observing another robot's behaviour is likely to be prone to sensor noise, which can be difficult to distinguish from erroneous behaviour.

Another major problem is that a robot's 'normal' behaviour is an emergent product of its controller code and interactions with other robots and the environment in which it is situated. Even if a robot is programmed to perform a simple task such as obstacle avoidance, its behaviour will be quite different in an empty arena compared to one cluttered with obstacles. Similarly, a particular fault will have a different effect on a robot's behaviour depending on the context. This makes it very difficult to provide examples of normal and abnormal behaviour *a priori*, as the definition of each changes at run-time.

Despite these challenges, solving the problem of exogenous fault detection in swarm robotic systems is a worthy pursuit, as it has great potential to improve their fault tolerance and reliability. Surprisingly, this it has not received much attention from swarm robotics researchers, perhaps due to the difficulty of the problem. This section reviews what little research exists on exogenous fault detection in a swarm robotics context, which represents the current state-of-the-art.

Exogenous fault detection in a collective robotic task

Christensen et al. [9] build on their previous TDNN-based endogenous fault detection approach [61, 64] (described in Section 2.3.1) and extend it to automatically synthesise exogenous fault detection modules for the *follow the leader* task (shown in Figure 2.13). This approach only considers collective robotic systems comprising two robots, however their work is still quite relevant as *s-bot* robots are used, which are designed for use in swarm robotic systems.

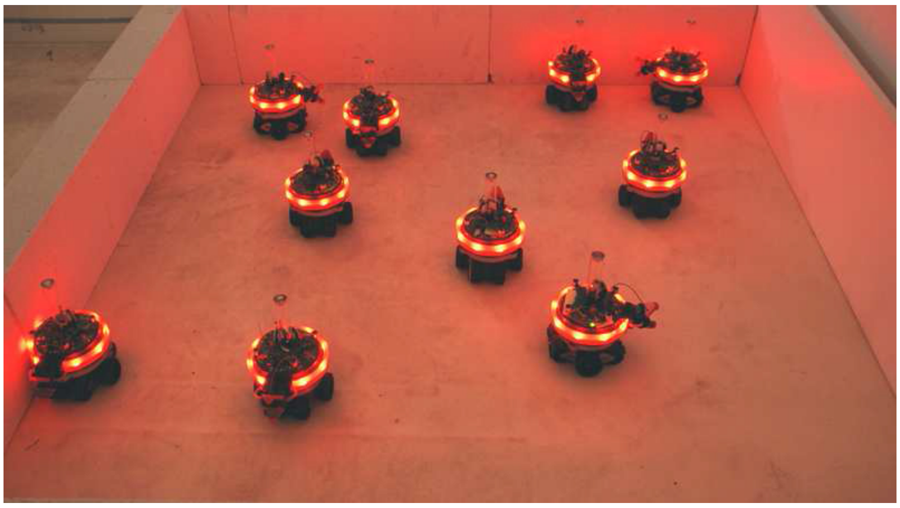


Figure 2.14: A swarm of 10 *s-bot* robots flashing their LEDs in synchrony, using the firefly-inspired exogenous fault detection algorithm. Taken from Christensen [58].

Again, *stuck-at-zero* and *stuck-at-constant* motor faults are considered, and are only injected into the follower. The leader *s-bot* uses a TDNN for exogenous fault detection in the same way as the follower uses one for endogenous fault detection, except that the sensor readings and control signals of the leader are correlated with the fault state of the follower. This allows the leader *s-bot* to recognise particular inputs and outputs to its controller program that correspond to faults in the follower *s-bot*.

An attractive feature of this exogenous fault detection approach, is that it does not require any explicit communication between the leader and follower *s-bots*. This is especially important in the context of exogenous fault detection, as relying on information from a potentially faulty robot may affect the robustness of fault detection.

Although faults are detected successfully in the scenario considered, this approach suffers from the same problems due to supervised learning as the endogenous approach based on the same architecture [61] (discussed in Section 2.3.1). Unfortunately, this means that this method of exogenous fault detection a poor candidate for use in swarm robotic systems.

From fireflies to fault-tolerant swarms of robots

As discussed in Section 2.2, Winfield and Nembrini [6] envisage the ability of robots to identify failures in neighbouring robots, so that they may be 'isolated' from the rest of the swarm. In a step towards the realisation of this vision, Christensen et al. [10] have developed a method of exogenous fault detection for swarm robotic systems, taking inspiration from the synchronised flashing behaviour seen in some species of fireflies.

Each *s-bot* flashes its on-board LEDs periodically, which can be sensed by other *s-bots* with their own on-board omnidirectional cameras. Christensen et al. [10] show that a group of *s-bots* is able to synchronise themselves through only local interactions, such that they flash their LEDs in unison, as shown in Figure 2.14. Once synchronisation has been achieved, the absence of flashes allows faults in other *s-bots* to be detected, as failed robots will cease to flash. These periodic flashes function as a 'heart-beat' mechanism [10], removing the need for a failed *s-bot* to explicitly signal to others in the swarm that it requires assistance.

When non-faulty *s-bots* detect failure in another robot, they move over to it and physically attach themselves via their gripper, simulating 'repair'. After 15 seconds the faulty robot detects that it has been 'repaired', and resumes normal operation. Christensen et al. [10] show that this approach is robust to multiple faults, and that a self-repairing swarm of robots is able to survive a relatively high failure rate. Bjerknes [7] likens this to an immune response, which affords a level of fault tolerance above that of simple redundancy.

However, Christensen et al. [10] only consider the case of completely failed robots, the effect of which on collective behaviour has been shown to be relatively benign by Winfield and Nembrini [6] and Bjerknes [7]. The occurrence of partial failures, such as motor failure during collective locomotion, is of far greater concern. For instance, if this fault detection mechanism were to be implemented on a swarm of robots performing phototaxis using the ω-algorithm, the anchoring issues would still manifest. A motor failure will not affect the robot's ability to synchronise the flashing of its on-board LEDs with other robot, thus preventing the fault from being detected exogenously.

Although Christensen et al. [10] demonstrate their approach to be successful, its usefulness is somewhat limited by the fact that it is unable to detect the partial failure of individuals. There is clearly scope here to develop a more robust method of exogenous fault detection, which would allow individuals to detect when other robots in the swarm are deviating from their expected behaviour, rather than only when complete failure occurs.

Model-based exogenous fault detection

Khadidos et al. [11] take a model-based approach to exogenous fault detection in swarm robotic systems, which allows faults in a robot's controller to be detected. Given a copy of the robot controller, along with a particular set of input values (sensor readings), the expected outputs (motor speeds) can be determined. Each robot in the swarm broadcasts its sensor readings and motor speeds to neighbouring robots. Receiving robots can then use a copy of the robot controller to check whether the reported motor speeds are consistent with the corresponding sensory input. If there is a significant discrepancy between the expected and reported output, then it may be inferred that a fault has occurred in the sending robot's controller.

This model-based approach is shown to be able to detect a single type of fault — noise added to a robot's navigation system. This causes the robot controller to produce unexpected motor outputs, given a particular sensory input. The main disadvantage of this approach, is that it only allows the internal workings of the robot controller to be validated. In the event of sensor failure, the faulty robot's controller would still produce the expected outputs given the reported sensory inputs, so the fault would not be detected.

Immune-inspired abnormality detection

Tarapore et al. [12] propose an exogenous fault detection approach based on a model of the adaptive immune system, which represents the current state-of-the-art in the literature. The adaptive immune system must be able to discriminate between self and non-self, so that the body's own cells and tissues are tolerated and allowed to function normally, whilst also detecting and attacking abnormal cells. This tolerance is achieved without any hard-coded notion of what constitutes normal cells — instead it is believed to be the result of the dynamics of regulatory and effector T-cell populations [12].

Their exogenous fault detection system is based on a simplified version of the crossregulation model (CRM) developed by Leon et al. [68]. This leverages the immune system's ability to tolerate self cells, which are abundant, and attack pathogens, which are not abundant [69]. The CRM works by modelling the population dynamics and interactions of cells present in the adaptive immune system (antigen presenting cells, effector cells, and regulatory cells), to capture the principles of immune tolerance.

Each robot in the swarm executes its own internal instance of the CRM to determine whether or not the observed behaviour of nearby robots should be tolerated. Distinct robot behaviours are encoded as antigens within the CRM, which the T-cell population is able to discriminate between based on their abundance [12]. This allows abnormal behaviours (those exhibited by the minority of neighbouring robots) to be detected, whilst tolerating normal robot behaviours.

Tarapore et al. [12] test their fault detection system on four different case study swarm behaviours: aggregation, dispersion, flocking, and homing. This is performed in a custom simulation with a toroidal environment, which allows robots to continue to interact with each other, even if the swarm becomes disaggregated. The following *fault-simulating* behaviours are tested:

1. Move in a straight line
2. Random walk
3. Circle around a fixed point
4. Stop completely

These behaviours are designed to mimic bugs in the robot controller code, as well as sensor, motor, and battery faults.

Each neighbouring robot is observed over a sliding time window of 45 seconds, and its behaviour is encoded as a set of boolean values that are concatenated to form a binary feature vector. Six features are defined that capture three different aspects of a robot's behaviour: (i) the robot's proximity to other robots, (ii) the robot's actions, and (iii) the robot's behavioural response to neighbouring robots:

1. Has there been at least one neighbour in the range [0, 30 cm], for the majority of the past time window?
2. Has there been at least one neighbour in the range [30, 60 cm], for the majority of the past time window?
3. Have they exceeded 5% of the maximum distance they can travel in a time window?
4. Have they exceeded 5% of their maximum speed in the past time window?
5. Has their angular acceleration exceeded 3% of its maximum, while there has been at least one neighbour in the range [0, 60 cm], at least once in the past time window?
6. Has their angular acceleration exceeded 3% of its maximum, while there have been no neighbours in the range [0, 60 cm], at least once in the past time window?

An observing robot calculates how many of its ten nearest neighbours are assigned to each feature vector, and generates a proportional number of antigen presenting cells for each within its internal CRM instance. The cell dynamics of the CRM are configured such that the observer will tolerate a particular behaviour if it is observed in more than one neighbouring robot. A particular robot's behaviour will also be tolerated if it differs by less then 1/3 of its features from any behaviours being expressed by two or more neighbouring robots, allowing for a degree of fuzzy matching. In any other case, the observed robot's behaviour will be classified as faulty.

This unsupervised data-driven approach allows the characterisation of normal robot behaviour to change online, based on the behaviour of nearby robots, and is therefore task-independent. No prior knowledge of possible failure modes is required — rather, *abnormal* behaviour (that which is different from the majority of nearby robots) is detected, under the assumption that deviations from normal behaviour is due to the presence of a fault.

Tarapore et al. [12] show that their exogenous fault detection approach exhibits high tolerance to robots behaving normally, and is able to reliably detect the fault-simulating behaviours, provided that they do not appear too similar (in terms of feature vector comparison) to the normal swarm behaviour (as illustrated in Figure 2.15). Furthermore, due to the fluid definition of normal behaviour, non-faulty robots will still be tolerated if the entire swarm switches to a different behaviour (for example, from aggregation to flocking). Tarapore et al. [12] also demonstrate that their approach is scalable, and performs well with swarms of up to 100 simulated robots.

Tarapore et al. [69] later extended their approach to work with swarms of robots exhibiting heterogeneous behaviours, which would previously have been detected as abnormal. In this extended approach, if an abnormal behaviour has been classified as normal in the past, or has not been encountered before, then it will only be classified as *suspicious*. When the level of suspicion exceeds some threshold, the behaviour will be classified as abnormal. Tarapore et al. [69] show that this incorporation of memory into the system allows heterogeneous behaviours to be tolerated, at the expense of detection latency.

The main weakness of this immune-inspired approach to exogenous fault detection is that, like the work of Lau et al. [56], 'normal' behaviour is defined by the majority behaviour of neighbouring robots. Therefore, if the number of faulty robots in the local neigh-

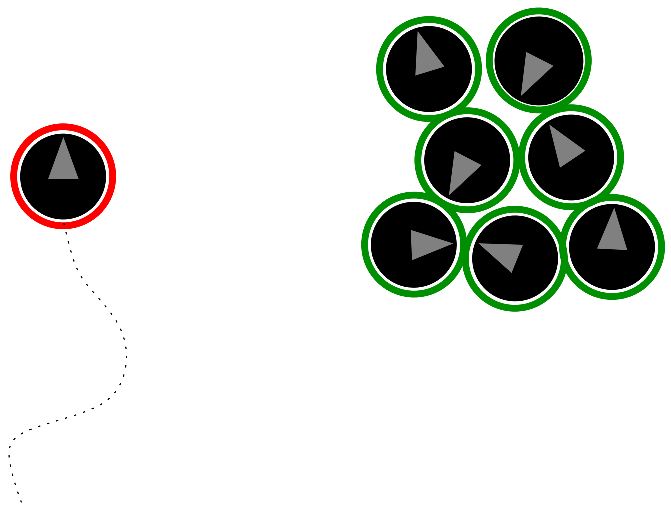


Figure 2.15: Illustration of immune-inspired abnormality detection. The green robots on the right are behaving in a similar manner, so they are tolerated, whereas the faulty red robot on the left is detected as abnormal, due to its inconsistent behaviour. Taken from Tarapore et al. [69].

bourhood outweighs the number of non-faulty robots, the model of normal behaviour will become distorted, resulting in misclassifications. Unfortunately, this means that the approach will not cope well when the swarm contains a large proportion of faulty robots.

Another problem is that certain failure modes cannot be detected if the faulty robot's behaviour is similar to that of the rest of the swarm. Taking the *circle* fault-simulating behaviour as an example: the feature vectors essentially encode *"is the robot close to other robots?"*, *"is the robot moving?"*, and *"is the robot turning in response to neighbouring robots?"*. If the circling robot is surrounded by other robots performing aggregation, then these features are insufficient to distinguish between faulty and non-faulty robot behaviour, so the fault will remain undetected. This is problematic, as partially failed robots can have a detrimental effect on swarm behaviour (see Section 2.1).

Despite these limitations, the immune-inspired approach presented by Tarapore et al. [12], and its extension to cope with heterogeneous behaviours [69], seems to be a promising data-driven method of exogenous fault detection, and should enhance the fault tolerance and reliability of the swarm robotic system when combined with fault diagnosis and recovery mechanisms.

2.3.3 *Summary*

This section has reviewed the latest advances in both endogenous and exogenous fault detection, in the context of swarm robotic systems. These studies represent the first step in an explicit process of fault detection, diagnosis, and recovery, which can be used improve the fault tolerance of individual robots as well as the entire swarm. Although endogenous approaches are useful and can increase overall swarm reliability, they cannot be solely relied upon to guarantee fault tolerance. Instead, exogenous fault detection methods must also be used to ensure long-term autonomy of swarm robotic systems.

2.4 SUMMARY

This chapter has reviewed the fault tolerance and reliability of swarm robotic systems, and highlighted the problems that can be caused by partially failed robots, motivating the need for an explicit approach to fault tolerance. Potential solutions to this problem were then reviewed in the context of natural/artificial immunity, with a focus on immune-inspired fault recovery mechanisms. Finally, both endogenous and exogenous fault detection approaches were examined, in the context of swarm robotic systems.

The swarm robotic exogenous fault detection methods reviewed in this chapter represent the extent of existing literature on the subject, of which there is very little. Clearly, there is scope for the development of other approaches, particularly those which may be able to cope with a large proportion of partially failed robots, as this feature is lacking in existing work. Without a robust and generalisable solution to the problem of exogenous fault detection, research into fault diagnosis and recovery mechanisms is unlikely to progress beyond task/environment/platform-specific solutions.

Part II

FAULT DETECTION VIA PREDICTION OF FUTURE BEHAVIOUR

3 PREDICTING FUTURE BEHAVIOUR

As discussed in Section 2.3.2, a robot's 'normal' behaviour is an emergent product of its controller code, and interactions with other robots and the environment in which it is situated. This makes it difficult to provide examples of normal and abnormal behaviour *a priori*, as the definition of each changes depending on the context.

These issues make unsupervised data-driven approaches to fault detection more attractive, as they do not require examples of normal or abnormal behaviour to be encoded into the system before deployment. The fault detection systems proposed by Lau et al. [56] and Tarapore et al. [12] overcome the problem of context-sensitive behaviour by using neighbouring robots as a model of normal behaviour. However, this assumes that the majority of neighbouring robots will be non-faulty, which may not always be the case.

The exogenous fault detection approach proposed in this thesis aims to overcome the problem of context-sensitive behaviour without relying on the assumption that most neighbouring robots will be non-faulty. This is achieved by providing each robot with an internal model that can generate examples of normal behaviour at run-time, based on the current context. The true behaviour of neighbouring robots can be compared against these examples of normal behaviour, to perform fault detection in a semi-supervised manner.

This chapter provides an overview of the proposed fault detection system at an abstract level, and reviews related work concerning the use of internal models. The experimental infrastructure required to implement this approach is also summarised, part of which is covered in greater detail in Appendix A. The remaining chapter in this part of the thesis (Chapter 4) presents the results of initial experimental work with a single robot, as an intermediate step towards implementing the proposed exogenous fault detection system in a swarm context.

3.1 EXOGENOUS FAULT DETECTION SYSTEM

It is proposed that each robot is provided with an embedded simulator, which can be used to make predictions about the behaviour of other robots based on a model of their expected behaviour. It is assumed each robot possesses a copy of neighbouring robots' controller code, which they can instantiate within their own internal simulation. The model of expected behaviour therefore comprises a copy of another robot's controller, and a simulator that is able to run the controller code in a simulated environment. Given that individuals are able to internally simulate the behaviour of other robots in the swarm, this can be used to compare expected behaviours against observed behaviours.

The motivation for using an internal simulation to predict robot behaviour is that the controller code provides an executable model of the robot's expected behaviour. If the real world scenario can be reproduced in simulation, then assuming that the simulation is able to mimic the real world controller's sensory inputs and actuator outputs, the simulated robot should behave like the real robot, thus allowing its behaviour to be predicted. This means that the proposed method would be independent of robot controller architecture, and could therefore even be used in heterogeneous swarms. However, the simplifying assumption is made initially that the swarm is homogeneous, so each robot has an identical controller. It is also assumed that the controllers will be static, in the sense that the robots do not learn new behaviours online.

An observing robot would initialise its internal simulation such that the relative positions and orientations of the other robots in reality are reproduced within the simulation. The simulation can then be executed for a short period of time, and the final position of each robot recorded. Real robot sensors and actuators are afflicted with noise, which should be modelled in the internal simulation. Therefore, the behaviour of each simulated robot will be stochastic. This means that from any particular initialisation, the predicted endpoint of a robot will differ between repeat runs of the simulation. The internal simulation must therefore be executed multiple times, to sample from the underlying probability distribution of possible endpoints. These endpoint distributions may then be compared against actual observed positions of the robots in reality. If there is a significant discrepancy between the predicted and observed behaviour for a particular robot, then this may indicate that it has developed a fault. An example scenario is illustrated in Figure 3.1.

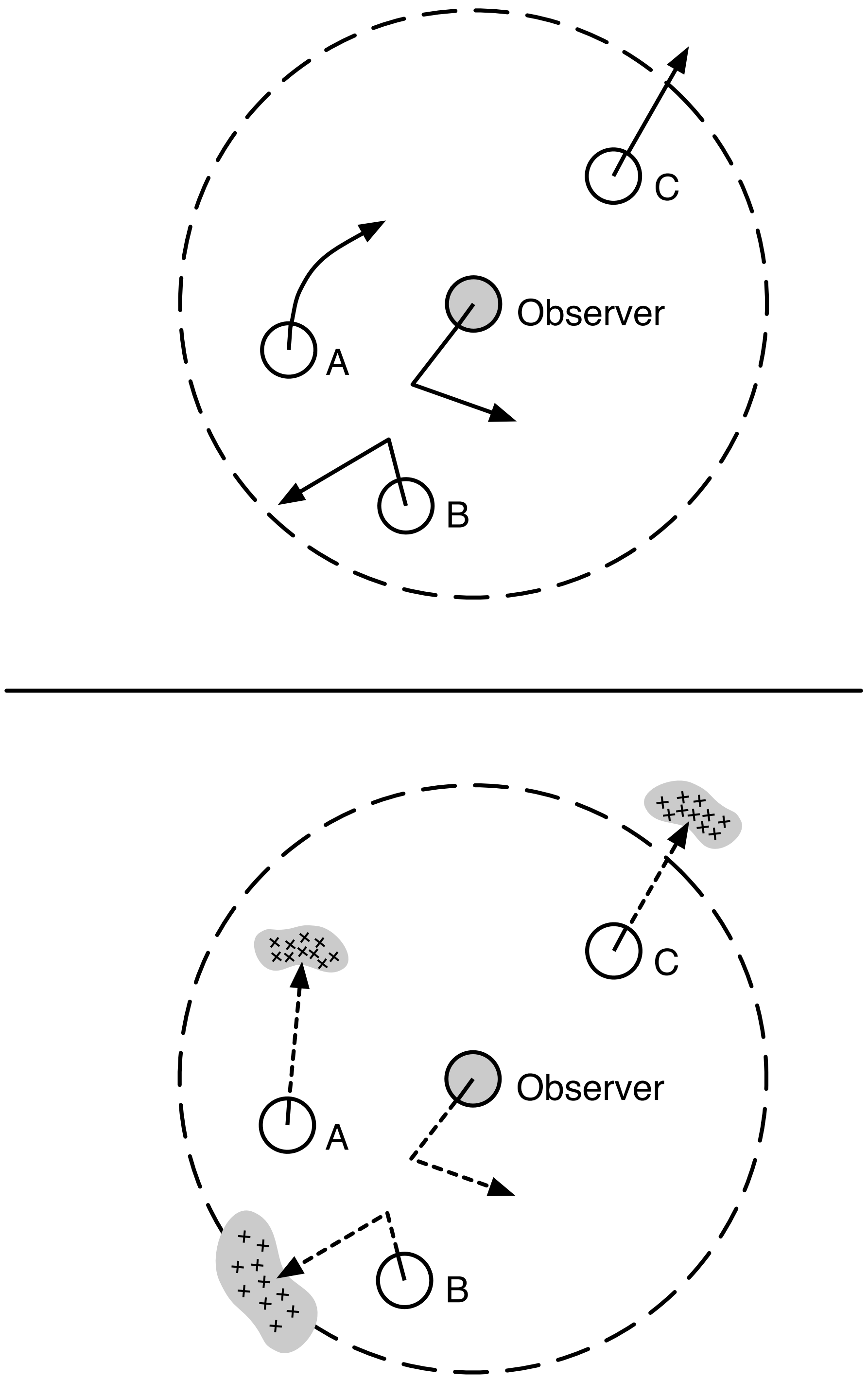


Figure 3.1: Illustration to show the discrepancy between observed and expected behaviours. *Top:* Situation in reality. Robot *A* has a fault in its right wheel motor, causing it to veer to the right. *Bottom:* Observer's internal simulation of its neighbours. The crosses denote the predicted endpoints of the robots. The predicted regions of non-faulty endpoints are shown in grey. Robots *B* and *C* are non-faulty, so behave as expected and will be classified as non-faulty. There is a significant discrepancy between the expected and observed behaviour of robot *A*, so it will be classified as faulty.

A major benefit of this approach is that it *preserves context*. If a robot's programming and the context of its embodiment are known, then it is possible to determine whether its behaviour is to be expected given the current scenario. By contrast, data-driven approaches reduce a robot's behaviour to summary statistics, causing this context to be lost, along with valuable information that can be utilised to perform fault detection.

This method of exogenous fault detection should also be robust, as each robot will independently monitor the behaviour of every other robot within its own range of perception, and will draw their own conclusions about which robots might be faulty. It is therefore completely distributed, allowing it to scale with increasing swarm size.

3.2 ROBOTS WITH INTERNAL MODELS

Related work that concerns the use of simulation-based internal models (as opposed to abstract mathematical models) for predicting robot behaviour exists in contexts other than fault detection. These studies are briefly reviewed here, as they are particularly relevant to the research presented in this part of the thesis.

3.2.1 *Embodied evolution*

O'Dowd et al. [71, 72, 73] use an embedded simulator to develop an online, on-board, distributed evolutionary algorithm for a swarm of robots, which allows the robots to adapt their behaviour in an open-ended manner. Each robot instantiates multiple model robots executing the same evolved controller within its internal simulator. It then monitors the simulated robots interacting with each other and their environment, so that the fitness of the evolved behaviour can be evaluated. The internal simulator executes faster than real-time, which allows several generations of evolution to occur before the fittest solution is selected to replace the real robot's current controller. This accelerates the speed of evolution, as each candidate solution does not need to be executed on a real robot to assess its fitness.

Although their use of an internal simulator is atypical of swarm robotics research, O'Dowd et al. [71] argue that their approach is justified because the cognitive ability of individual robots in a swarm need not be constrained, and that their solution is still scalable due to the use of local communication and decentralised control. Their work

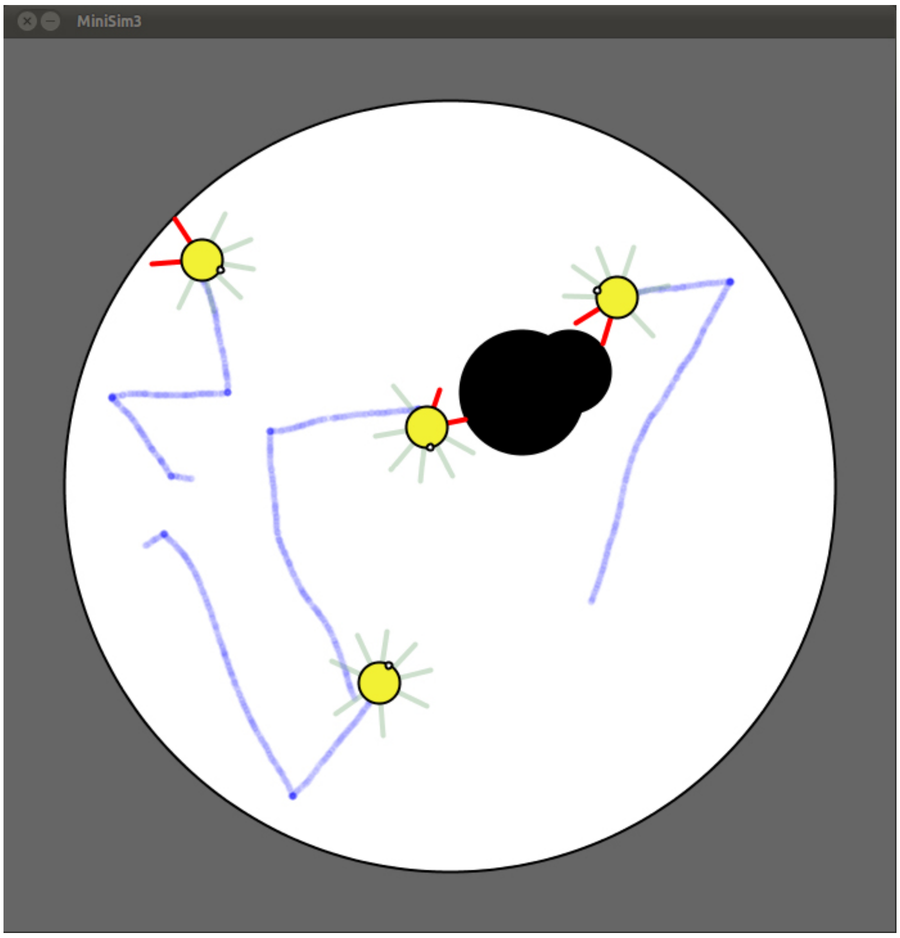


Figure 3.2: Visualisation for the minimal e-puck simulator developed by O'Dowd [70]. Four simulated e-puck robots are shown, along with their past trajectories and IR sensor ranges. The black circles represent fixed obstacles that the robots can detect with their simulated IR sensors, in addition to the circular arena wall. Taken from [70].

can therefore be categorised as a Scalable Swarm Robotics approach, according to Sharkey's taxonomy [26].

O'Dowd et al. [71] demonstrate the use of the embedded simulator on real e-puck robots, each augmented with a Linux extension board [74] that improves processing and memory resources. The simulator was written in C and is deliberately minimal, because it was designed to run on the Linux extension board as fast as possible. Every object in the simulator is represented as a point on a 2D plane with a fixed radius, and the e-puck movement is simulated using simple two-wheel differential kinematics. The robots' IR sensors are modelled as range-limited cones, and distances are converted to sensor readings using a look-up table of raw sensor data collected from a real robot, with uniform noise added at run-time. Efficient simulation is made possible by assuming that physical properties such as mass,

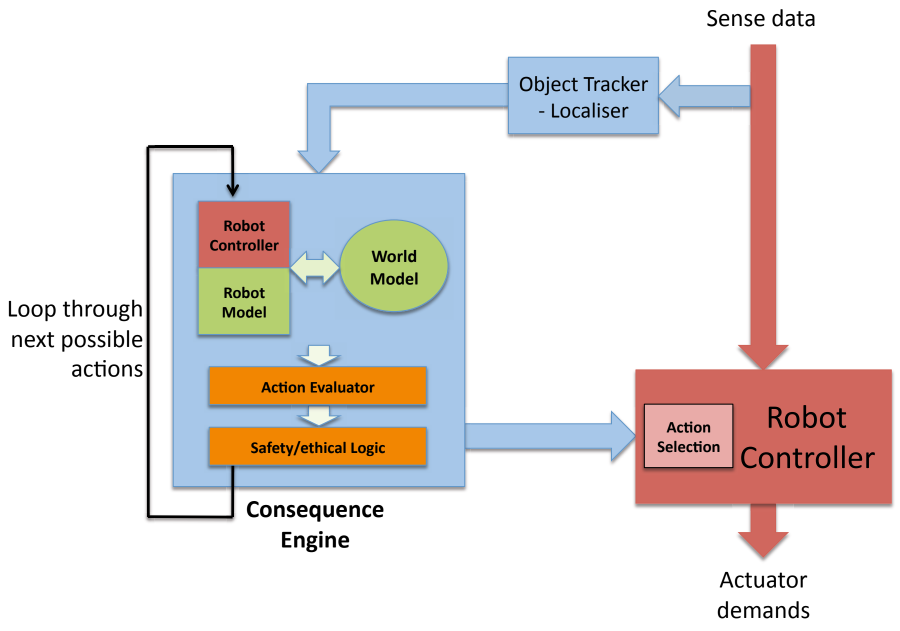


Figure 3.3: Consequence Engine architecture. Internal model data flow is shown in blue. Robot control data flow is shown in red. Taken from [75].

inertia, and momentum need not be modelled, as the e-puck robots have high motor torque relative to their weight, so are able to stop almost immediately. This allows for fine-grained simulation, with each simulation step representing 40 ms of real-time, which corresponds to 25 discrete iterations for every simulated second. A visualisation of the embedded simulator is shown in Figure 3.2.

Although O'Dowd et al. [71] do not use the internal simulator to predict the future behaviour of individual robots from specific starting positions and orientations, there is no reason why it could not be used for this purpose. When used in an evolutionary swarm robotics context, a high level of noise is added to the simulation to prevent the evolutionary process from exploiting modelling inaccuracies [71]. However, if precise prediction of future behaviour is required, this noise can be omitted to improve simulation accuracy.

3.2.2 *Consequence Engine*

Recently, Winfield [76] proposed an architecture based on internal models, dubbed the *Consequence Engine* (shown in Figure 3.3), which allows a robot to predict the consequences of its own actions. The robot may then make an informed decision about which future action to take, based on whether each action is considered safe and/or ethical. The internal model is a simulator that comprises a model of

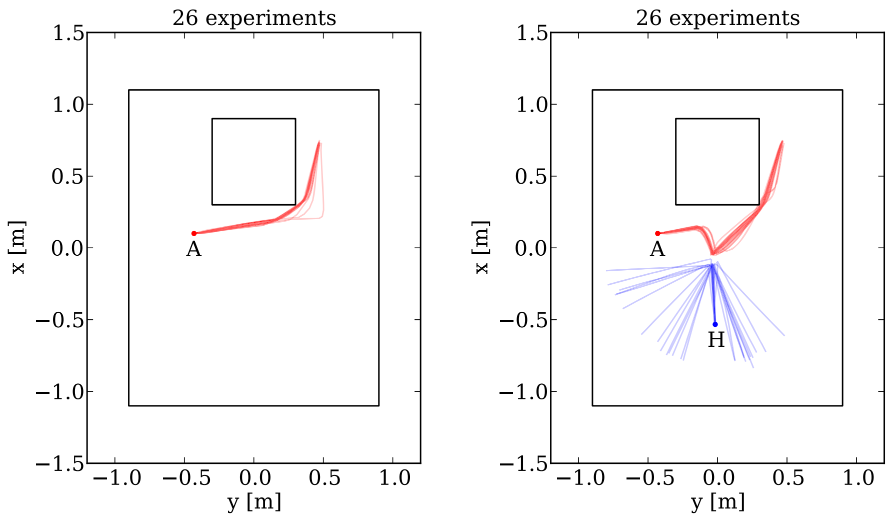


Figure 3.4: Results of robot ethics experiment conducted by Winfield et al. [75]. Robot *A* starts on the left of the arena and moves towards its goal in the upper right. Robot *H* starts towards the bottom of the arena, and moves towards the virtual hole. Robot *A* intercepts robot *H*, so that it performs obstacle avoidance and its trajectory is deflected to safety. Taken from [75].

the world, and a model of the robot situated within it. The world model represents the robot's real-world environment, which may include static objects (environmental obstacles) as well as dynamic objects, such as other agents. The robot model represents the real robot's morphology, sensors, and actuators. Its behaviour within the simulated world is dictated by a copy of the real robot's controller code.

This Consequence Engine's internal model is initialised using the real robot's sensory data such that the simulated world, and the location and state of the robot model within it, is consistent with the current real-world scenario. The internal model is then executed to predict the consequence of a given action, for some time period into the future. The outcome is recorded, then the internal model is reset to the same initial state, so that it can be executed again to predict the consequence of another possible action. By predicting the outcome of various possible actions, the Consequence Engine is able to determine the set of actions that is considered safe and/or ethical. This architecture continuously operates alongside the real robot controller, acting as an advisor that suggests which actions to take next. The result is a robot with some degree of self-awareness, that can make better decisions by 'imagining' their outcome.

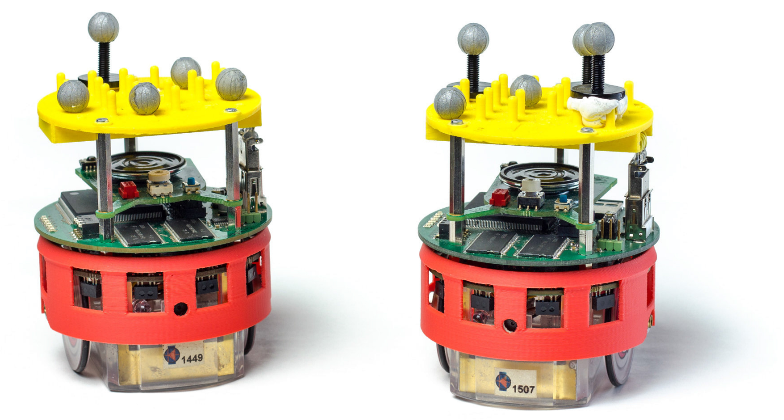


Figure 3.5: Two e-puck robots fitted with yellow 3D-printed hats that provide a matrix of pins for reflective markers, which are used by a Vicon tracking system to uniquely identify the robots. Each robot has a Linux extension board connected to the motherboard, and a USB Wi-Fi adapter that slots underneath the 3D-printed hat. Taken from [77].

Ethical robots

Winfield et al. [75] use the Consequence Engine architecture to demonstrate 'ethical' behaviour of real robots, in the toy example scenario shown in Figure 3.4. Robots A and H are placed in an arena with a virtual hole in the ground, which can only be sensed by robot A. Robot H does not have a Consequence Engine, and simply executes an obstacle avoidance controller, so will 'fall' into the virtual hole if left to its own devices. Robot A internally models the real world, containing itself and robot H. Using the Consequence Engine, robot A is able to predict the future trajectory of robot H, and determines that it will fall into the hole unless action is taken to intercept its trajectory. Robot A therefore changes its planned course of action, such that robot H is forced to avoid it and move to safety. It is important to note that robot A has *a priori* knowledge of robot H's controller code (in addition to its own). This is necessary to predict the behaviour of robot H in the internal model. The robot controllers are also *stateless*, which makes initialisation of the internal model much easier, as only the position and orientation of each robot must be known.

The infrastructure required to implement this simple experiment is quite extensive, as the physical e-puck robots used by Winfield et al. [75] (shown in Figure 3.5) have neither the sensing capability nor the computational resources required to implement an embedded Consequence Engine architecture. Object localisation and tracking is

Figure 3.6: Vicon tracking system at the Bristol Robotics Laboratory, used by Winfield et al. [75] to implement their experimental work. Taken from [77].

achieved via a Vicon[1] tracking system (shown in Figure 3.6) and a network infrastructure. This virtual sensing capability is mainly used for the initialisation of the internal model — the robots' on-board IR sensors are still used for short-range obstacle avoidance. The internal model is implemented using the Stage [78] robot simulator, which runs on a separate server in a service-oriented architecture, accepting requests from robots over a network connection. The e-puck robots are able to communicate with these networked components via the use of a Linux extension board and a USB Wi-Fi adapter. Conceptually, the Consequence Engine runs on-board one of the robots, despite the use of remote processing and virtual sensing.

The Consequence Engine architecture is only useful in practice if a robot is able to predict the consequences of its actions in real-time, so that it has time to act accordingly. The experimental framework developed by Winfield et al. [75] allows the Consequence Engine to update at a speed of 2 Hz, providing 0.5 seconds of real-time to predict and analyse the consequence of future actions before its next invocation. The Stage simulator can simulate the simple example scenario shown in Figure 3.4 600 times faster than real-time, allowing a budget of 300 seconds to perform the necessary predictions. Winfield et al. [75] chose to predict the next 10 seconds of future behaviour, thus

1 http://www.vicon.com

allowing around 30 possible actions to be simulated before the Consequence Engine next updates. The e-puck robots move at 10 cm/s, so this corresponds to 1 metre of distance travelled for each action.

Safer robots

Blum [77] presents another experiment with the Consequence Engine that focuses on safety, rather than ethics. A single 'intelligent' robot (that implements the Consequence Engine architecture) must navigate from one end of arena to the other, whilst avoiding five other robots performing simple obstacle avoidance. The intelligent robot achieves this by predicting the trajectories of the other robots, to determine whether they will come within some unsafe radius around its location, and takes action to prevent this from occurring in reality. Figure 3.7 shows an example experimental run, which demonstrates that the intelligent robot is able to safely reach its goal without risking collisions with the other robots.

In the ethical robot experiment devised by Winfield et al. [75], the intelligent robot only simulated the outcome of its own interactions with another robot. In Blum's robot safety experiment [77], the intelligent robot internally simulates the outcome of other robots not only interacting with the environment (in the form of wall avoidance), but also interacting with each other. The virtual sensor range of the intelligent robot is also restricted, so it can only predict the trajectories of other robots that are currently visible (as shown in Figure 3.7). This experiment is therefore typical of a swarm scenario, where robots frequently interact with each other and their environment, and are limited to local sensing.

3.3 EXPERIMENTAL INFRASTRUCTURE

The Consequence Engine architecture is very similar to that of the exogenous fault detection system proposed in Section 3.1, just applied in a different context — particularly with respect to the internal model, which predicts the behaviour of robots situated in a simulated model of the world. This internal model runs alongside the robot controller, and must be initialised in the same way, such that it reproduces the real-world scenario. The recent work of Blum [77] is particularly relevant, as it provides an example of predicting future trajectories of robots in a swarm context. In order to implement the

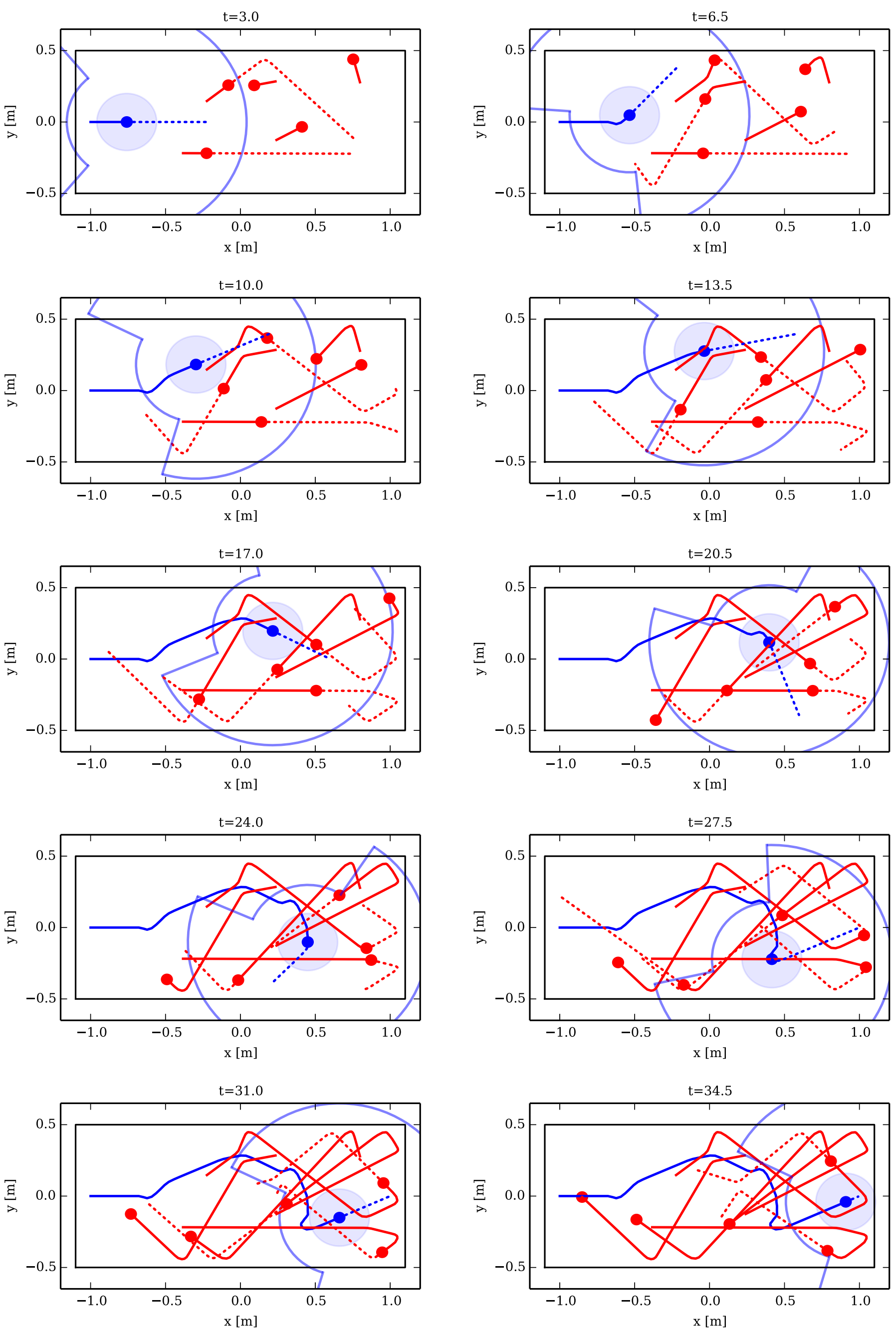


Figure 3.7: Time-lapse of an example run from the robot safety experiment conducted by Blum [77]. The intelligent robot is shown in blue, and the other robots are shown in red. Solid lines represent true trajectories, while dotted lines represent predicted trajectories. Virtual sensor range and safety zone around the intelligent robot are also shown in blue. Taken from [77].

exogenous fault detection system, a similar experimental infrastructure was therefore required. The development of this experimental infrastructure is discussed in the following sections.

3.3.1 *The e-puck robot platform*

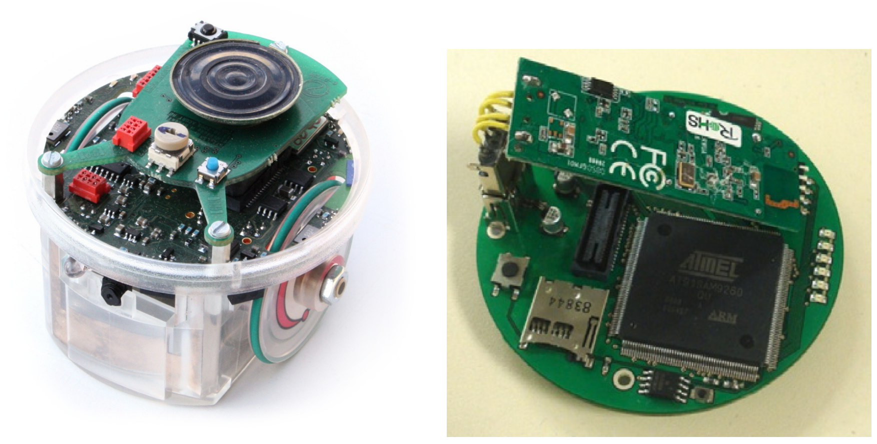
Like O'Dowd et al. [71] and Winfield et al. [75], the e-puck robot platform (shown in Figure 3.8a) was used for the experimental work presented in this thesis. The e-puck uses differential drive stepper motors to move around, which afford the robot precise movement with virtually no inertia, making its behaviour easier to predict in simulation. Eight active IR transceivers are distributed around the e-puck's body, which allow it to sense its proximity to obstacles and communicate with nearby robots. The robot also has a colour camera that can be used to implement basic vision.

The e-puck robot has very limited hardware resources, which limits its usefulness as a research platform. In order to enhance its utility, it is augmented with a Linux extension board [74] (shown in Figure 3.8b). This improves the computation, memory, and networking performance of the e-puck, and is comparable to the Gumstix Overo COM turret[2], which offers similar functionality. The board features an ARM processor that runs in parallel with the dsPIC microcontroller on the e-puck motherboard, and communication between the two is achieved via an SPI bus. The dsPIC handles low-level motor control, data processing and sensor interfacing, while the extension board may be used for high-level control algorithms, wireless communication, and computationally expensive operations such as image processing [74]. The Linux board also provides the robot with networked communication via the use of a USB Wi-Fi adapter.

3.3.2 *Internal simulator*

The eventual goal of this research was to embed the proposed exogenous fault detection system on the Linux extension board, so the simulator developed by O'Dowd [70] was chosen to implement the internal model. This was chosen over more general purpose robot simulators (such as Stage) due to its minimal nature, which facilitates a fully-embedded a solution that does not require remote processing. However, the simulator was originally designed for the embedded on-

2 http://www.gctronic.com/doc/index.php/Overo_Extension



(a) Basic e-puck robot. Taken from [79]. **(b)** Linux extension board with USB Wi-Fi adapter. Taken from [74].

Figure 3.8: Extended e-puck robot platform.

line evolution of robot controllers, not for accurately predicting robot behaviour. A significant problem with predicting behaviour is that there will always be some discrepancy between simulation and the real world, referred to as the *reality gap* [80]. Closing this reality gap is important when attempting to accurately predict robot behaviour for the purpose of exogenous fault detection, as the future position of each robot must be precisely predicted to minimise false positives.

In the field of swarm robotics, simulators are typically only used to develop robot controllers offline for later deployment on real robots. This application is not specifically geared towards accurately predicting the behaviour of a real robot so a crude model of robot behaviour, and therefore a significant reality gap, is often sufficient. Similarly, the Consequence Engine architecture employed by Winfield et al. [75] and Blum [77] only requires predictions of future behaviour that are accurate enough to determine the most favourable course of action. In an ideal world, the reality gap would be entirely eliminated. However, as argued by Jakobi [81], it is impossible to create an exact model of a robot and its environment — the simulation will inevitably remain an approximation of reality. To explain why, O'Dowd et al. [72] decompose the reality gap into three categories of correspondence between simulation and reality:

“ ROBOT-ROBOT: differences in physical robot aspects, such as their morphology.

ROBOT-ENVIRONMENT: differences in the dynamic interactions between robots and their environment.

ENVIRONMENT-ENVIRONMENT: representation of salient features of the environment. ”

In order to accurately predict robot behaviour, all three categories must be modelled with sufficient fidelity. There is a trade-off between the fidelity of the simulation and the speed at which it can be executed. A high fidelity simulation may model the robot's behaviour very well, but will run slowly. Conversely, a low fidelity simulation may run very quickly, but provide poor predictions of robot behaviour. The simulation must run faster than real-time for it to be useful for predicting robot behaviour, while also providing predictions accurate enough to achieve good fault detection performance. After making some modifications for the research in this thesis, to ensure that each category of correspondence was modelled with sufficient fidelity, the minimal e-puck simulator developed by O'Dowd [70] was found to produce adequate predictions of robot behaviour.

Robot-robot correspondence is satisfied quite easily, as the e-puck robots can simply be modelled as a circles with the same circumference as a real e-puck. Their positions are updated using two-wheel differential drive kinematics, with wheel speeds calculated from measurements of a real robot's movement. In O'Dowd's original implementation, each step of the simulation represented 40 ms of real-time. For this research, this was reduced to 10 ms, to increase the fidelity of simulating the robots' movement to 100 updates per second. Robot-environment correspondence is harder to achieve, as it relies upon the use of an accurate IR sensor model. It has been shown that the response of active IR sensors depends not only on the distance from an obstacle, but also the angle, and the proportion of the beam that is reflected [82]. However, in this minimal simulator the IR sensor readings depend only on the distance from an obstacle, and are emulated using raw data obtained from the real robot's sensors, with the addition of uniform noise [72]. It was decided that this sensor model should remain simplistic, to investigate the effect of imprecise simulation on fault detection performance. Environment-environment correspondence was side-stepped somewhat by limiting scope to include only empty environments free of obstacles, for the sake of simplicity. With only arena walls to model with simple 2D geometry, this form of correspondence is easily satisfied.

For the experimental work in this thesis, the parameters of the simulation were calibrated manually to produce a sufficiently faithful reproduction of real robot behaviour. The focus was not upon obtaining perfect predictions of the real robot's future behaviour, rather predictions accurate enough that reliable fault detection could be achieved.

3.3.3 *Functionally equivalent robot controllers*

In order to facilitate meaningful comparisons of expected and observed behaviours, the real and simulated robots must both be programmed with functionally equivalent controller code. This should ensure that any deviation between the expected and observed behaviour of a non-faulty robot is due solely to the reality gap and sensing inaccuracies.

Player/Stage

The initial intention was to develop the robot controllers using software freely available from the Player/Stage project [83]. This open source project provides two main pieces of software: the Player robot device server [84], and the Stage multi-robot simulator [85]. These tools are widely used by robotics researchers, and allow for the implementation of a robot controller on real robots and simulated counterparts. The Player server provides a network interface to a variety of robot sensors and actuators via TCP sockets, allowing control programs to be written in any language that supports sockets [83]. Stage may be used as a plug-in to Player, allowing client programs to control simulated versions of real robots through a common interface.

The Player server can be executed on the Linux extension board [74], providing a network interface to the e-puck's sensors and actuators. Robot controller code also running on the Linux board is able to control the robot via this interface, by communicating with the local Player server over TCP sockets. Actuator commands are sent to, and sensor data received from, the dsPIC on the e-puck motherboard via the SPI bus. There should be no perceivable difference between real or simulated robots from the perspective of a client program written for use with Player/Stage. This is attractive from the perspective of developing functionally equivalent controller code, as it enables the development of identical code for robot controllers that are to be deployed on both simulated and real robots, giving confidence that their control logic is equivalent.

Unfortunately, despite these benefits, Stage is too heavy-weight to be executed on the Linux extension board. It is for this reason that Winfield et al. [75] resorted to running Stage remotely on a separate server for their experiments with the Consequence Engine. As discussed in Section 3.3.2, a minimal e-puck simulator was chosen for the research presented in this thesis instead, with the aim of developing a fully embedded solution. This necessitated an alternative method of developing functionally equivalent controller code.

Minimal e-puck simulator

The main challenge with the chosen hardware platform, is that the robot's control logic may be split across the ARM processor on the Linux extension board and the dsPIC microcontroller on the e-puck motherboard. To avoid simulating the interaction of these two components, a higher level of abstraction is required. A simple solution is to enforce a master-slave relationship, such that the dsPIC only performs low-level interfacing with sensors and actuators. This ensures that all of the control logic is implemented on the Linux extension board, thus removing the need to separately simulate code running on the dsPIC.

This is especially important given that the dsPIC is programmed using low-level C code, while the control logic running on the ARM processor may be written in any language that can interface with the Linux device that represents the SPI bus. Attempting to achieve functional equivalence when simulating a robot controller that is split across two different programming languages would be particularly challenging, especially if the robot simulator is written in yet another programming language. Fortunately, the minimal e-puck simulator developed by O'Dowd [70] is written in C++, so it was possible to write controller code in the same language for both the Linux extension board and the internal simulator.

The remaining challenge was that of developing an API that provided a common interface to the robot's sensors and actuators, allowing identical controller code to be used for both the real and simulated robots. For the physical hardware, the back-end of this API allows controller code running on the Linux extension board to retrieve sensor readings and send actuator commands to the dsPIC via the SPI bus. For a simulated robot, the API back-end translates wheel speeds into differential kinematics, and returns emulated IR sensor values based on distances to simulated obstacles.

3.3.4 *Observing neighbouring robots*

In order for an observing robot to make comparisons between the expected and observed behaviour of its neighbours, the e-puck robots must have a method of observing each other's behaviour. For this research, it is sufficient to define the observed behaviour of a real robot as its position and orientation over time. Thus, only (x, y) position coordinates and the angle representing the robot's orientation need to be recorded. Ideally, this data would be obtained using on-board sensor hardware. However, in order to simplify the problem initially each robot is provided with local information about the position/orientation of other robots, collected using a tracking infrastructure (like that used by Winfield et al. [75]) that observes the swarm from a bird's-eye view. This is described in more detail in Appendix A.

In terms of collecting equivalent observation data using on-board sensors, there are a number of options available. Winfield and Erbas [86] demonstrate that e-puck robots are able to track the relative direction of movement, and distance, of another robot by tracking its position and size in its camera field of view. However, the e-puck robot would only be able to observe neighbours directly in front of it. In order to observe the behaviour of every neighbour, the e-puck omnidirectional vision turret may be more useful [29], although this depends on the frame rate of the camera.

An alternative approach might be to use the e-puck range and bearing board [87]. Each board comprises 12 IR emitters and receivers spaced 30° apart, which allow e-puck robots to determine the relative range and bearing to their neighbours, at a distance of up to 80 cm. Range and bearing data can be used to calculate the relative position coordinates of another robot using simple trigonometry, however another method would be required to sense the orientation of a robot. For example, robots could proprioceptively sense their own orientation via an on-board compass, and broadcast this information to neighbouring robots. Unfortunately, no e-puck range and bearing boards were available for this research, but similar virtual sensing capabilities can be implemented using a tracking system.

3.4 SUMMARY

This chapter has outlined the proposed exogenous fault detection system, which is based on the comparison of expected and observed robot behaviours. Related work on robots with internal models has been reviewed, including embedded simulations for online evolution, and predicting consequences of future behaviour for engineering safe and/or ethical robots. Ideas from these publications have contributed to the development of an experimental infrastructure for embedded predictions of future robot behaviour, which integrates the minimal e-puck simulator developed by O'Dowd [70] and virtual sensing via a tracking infrastructure like that used by Winfield et al. [75]. Appendix A describes this tracking infrastructure in more detail, which was used for the purpose of detecting faulty robot behaviour during the experimental work presented in the next chapter.

4 | SINGLE ROBOT FAULT DETECTION

As an intermediate step towards implementing the proposed exogenous fault detection system in a swarm context, experimental work was carried out to investigate the viability of fault detection based on simulated predictions of a single robot's behaviour. Although this is a simpler task than predicting the behaviour of a robot in a swarm, it is still non-trivial. This chapter presents the results of this initial experimental work, which shows that simulation can be used to successfully predict real robot behaviour. However drift between simulation and reality occurs over time due to the reality gap, thus necessitating periodic reinitialisation of the simulation to reduce false positives. Using a simple obstacle avoidance controller afflicted with partial motor failure, it is shown that selecting the length of this reinitialisation time period is non-trivial, and that there exists a trade-off between minimising drift and the ability to detect the presence of faults. Following this is a discussion of open problems with this fault detection approach, and proposed solutions.

4.1 FAULT DETECTION

As discussed in Section 3.1, assuming that a simulation can provide sufficiently accurate predictions of real robot behaviour, it should be possible to use it for the purpose of fault detection. The robot controller can be instantiated within the simulation, embodied in a simulated model of the real robot, and used to generate predictions of non-faulty behaviour. A significant discrepancy between these simulated predictions and the real robot's observed behaviour may indicate the presence of a fault.

Bjerknes and Winfield [8] demonstrated that motor failure had the most detrimental affect on a swarm's ability to carry out its task. For this reason, motor failure is used here as a case study. However, instead of testing complete motor failure, which would be very easy to detect due to rapid divergence of non-faulty and faulty behaviour, partial motor failure is investigated. The particular fault considered here is a permanently slow left wheel. Over time, this fault will cause

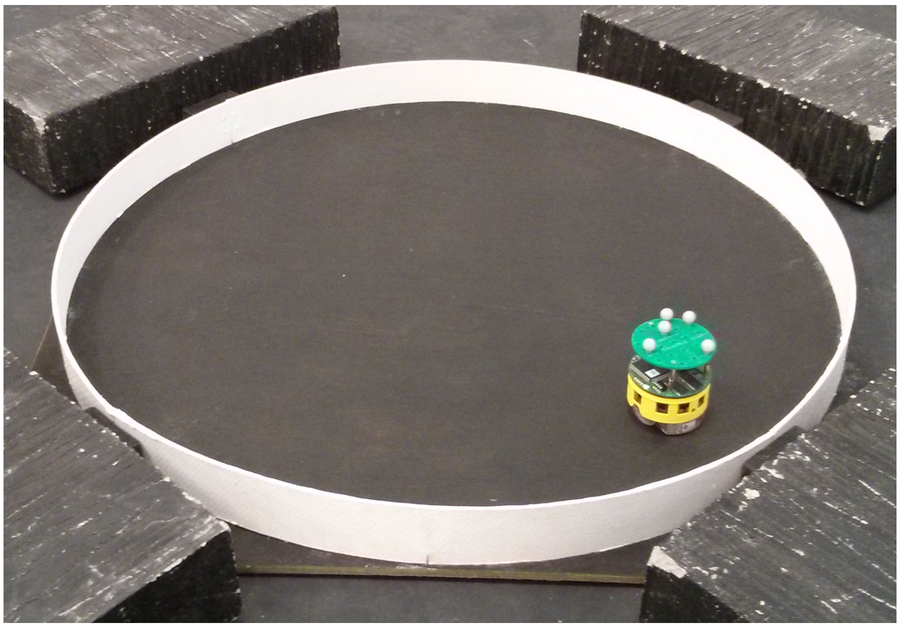


Figure 4.1: The e-puck robot with Linux Extension Board and tracking hat, inside an enclosed circular arena with a diameter of 80 cm.

the robot to veer gently to the left. This is a minor fault, and therefore quite difficult to detect.

The focus of this experimental work was not upon finding an optimal solution for the case study considered here, rather to investigate the fundamental issues inherent in the proposed fault detection approach, primarily as an indication of whether similar issues might exist in other scenarios.

4.2 EXPERIMENTAL SETUP

The task of obstacle avoidance was chosen as a case study, because it is well-understood, and relatively simple to model in simulation. The e-puck robot performs obstacle avoidance in an enclosed 80 cm diameter circular arena free of obstructions, as shown in Figure 4.1. This particular arena shape was chosen because it can be easily modelled in the minimal simulator as a simple radius from the world origin.

4.2.1 *Robot controller*

The robot controller implements a simple obstacle avoidance behaviour that sets the wheel speeds based on IR sensor readings. The IR sensors are the only input to the robot controller, and the readings ob-

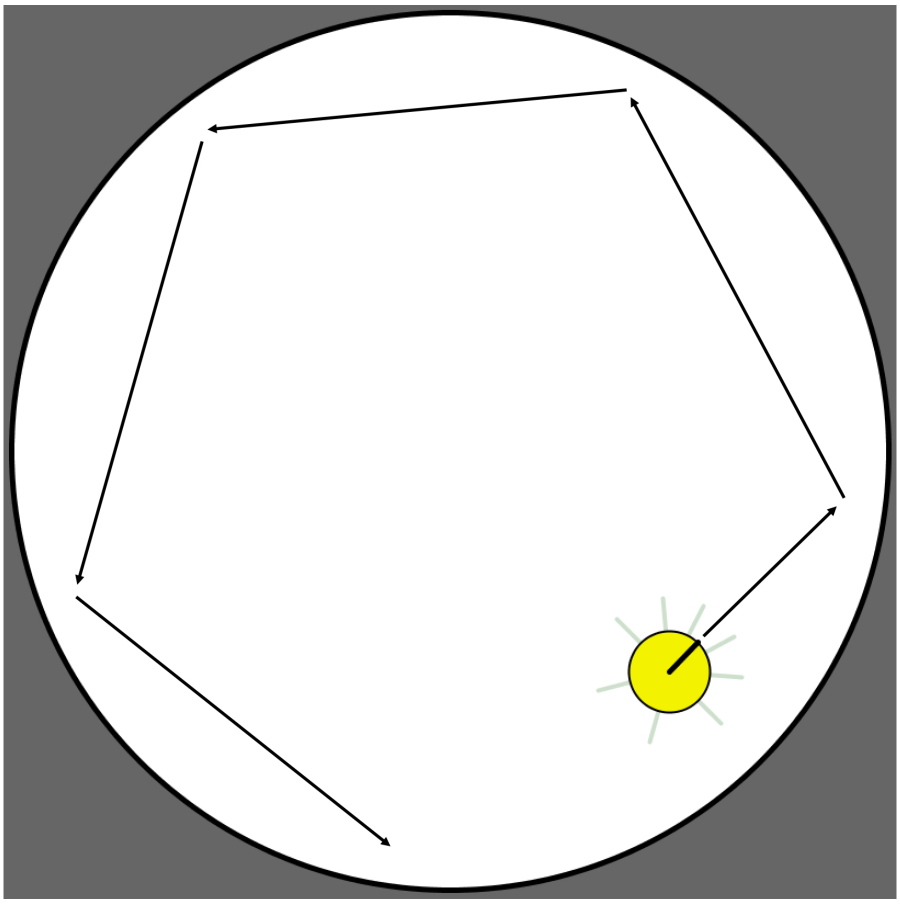


Figure 4.2: Simulation of the e-puck in the circular arena shown in Figure 4.1. The lines protruding from the robot's body represent the range of each IR sensor. The arrows represent a typical trajectory resulting from the obstacle avoidance controller.

tained from them are directly translated into left and right motor speeds using a vector of weights. This tight coupling of the sensors and actuators effectively results in a Braitenberg vehicle [88]. IR sensor values below a certain threshold are ignored, so the robot's behaviour is insensitive to IR interference. Unless any of the IR sensor values are above this threshold, the robot will move in a straight line at 2.6 cm/s.

As with the work of Winfield et al. [75], the robot controller is deliberately stateless. This is because from the perspective of an outside observer, given only a snapshot of the system at a particular instant in time, it would not be possible to determine the internal state of the robot controller. It may be possible to infer the internal state given a history of the robot's behaviour, however this was beyond the scope of this initial work.

4.2.2 *External observer*

The minimal e-puck simulator was executed on a separate machine that acts as an external observer. As with the work of Winfield et al. [75], a model of the arena is known *a priori*, and a global view of the world is available to the observer. Figure 4.2 shows the simulated e-puck robot situated within a model of the arena.

In order to perform the comparison of expected and observed behaviour, a method of observing the real robot's behaviour was required. The motion capture system described in Appendix A was used to monitor the position and orientation (or *pose*) of the robot over time, by tracking the pattern of retro-reflective markers placed on the 'hat' that the robot wears (see Figure 4.1). The tracking data was also used for post-experiment analysis, and transmitted to the robot over Wi-Fi so that it could be instructed to drive to a desired initial pose at the start of each experimental run.

4.3 PROBLEM ANALYSIS

This section presents the results of initial experiments that were carried out to investigate the issues inherent in using simulated predictions of robot behaviour for fault detection.

4.3.1 *Real robot behaviour*

For any particular initial pose, the robot's endpoint after a certain time period may be recorded. However, even with deterministic controller code, a real robot's behaviour is stochastic. This is because the sensory inputs to the controller are often prone to noise, particularly in the case of active IR sensors. There may also be a small discrepancy between the wheel speeds output by the controller and the actual speed of the robot, due to variations in the surface that the robot is driving on. Therefore, for any particular initial pose, there will in fact be a probability distribution of possible endpoints that the robot may reach after a given time period.

To demonstrate this, the tracking system was used to instruct the robot to drive to the coordinates (200, -200), and turn to an angle of $45°$, such that it is facing the arena wall as shown in Figure 4.2. Once in position, the robot executes the obstacle avoidance controller for

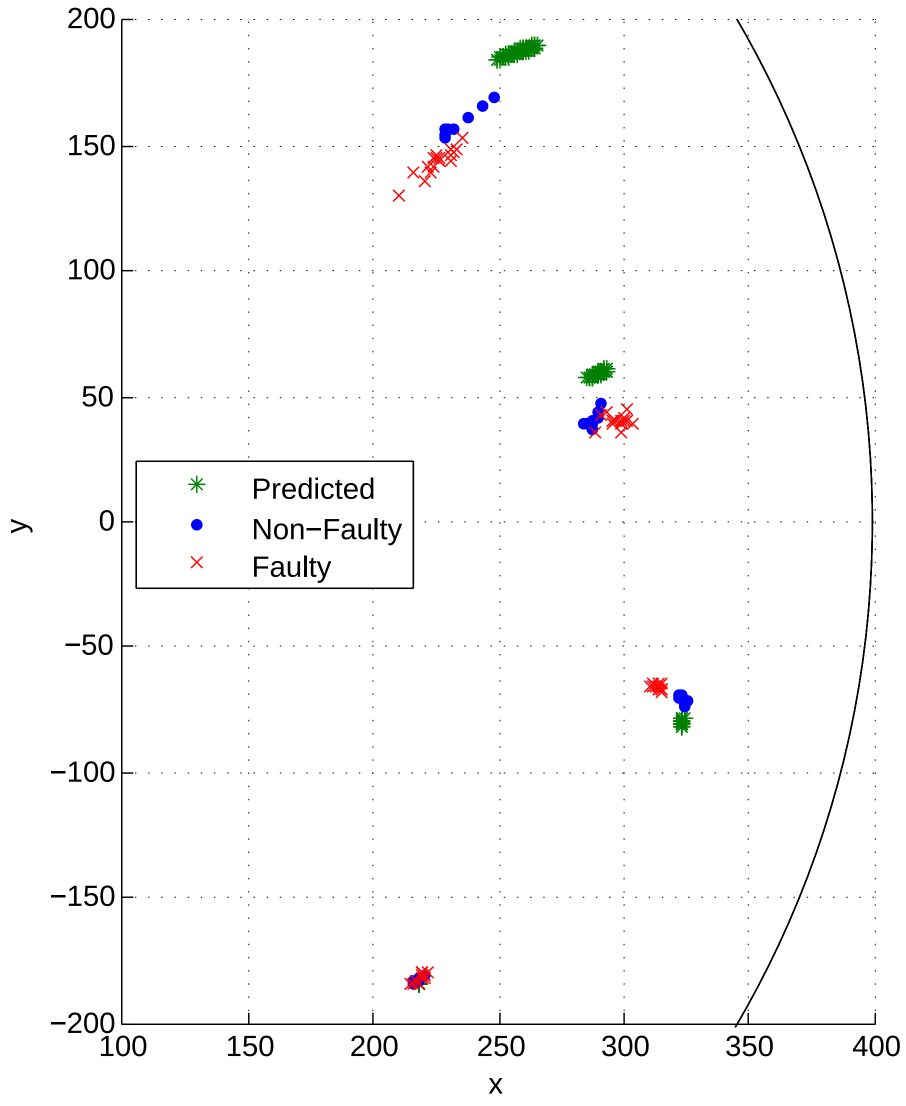


Figure 4.3: Real robot non-faulty and faulty class distributions, and the predicted non-faulty class distribution over time. The robot begins at the coordinates (200, -200) facing the wall at 45°, as shown in Figure 4.2.

20 seconds. Throughout the duration of the run, the robot's position is recorded using the tracking system. The robot then drives back to the same initial pose, and repeats the run. The experiment was first carried out using a non-faulty robot, and then repeated with a permanently 'faulty' robot that used a modified controller which reduced the left wheel speed output. For each robot, 15 repeat runs were carried out. This sampled from the underlying probability distribution of possible endpoints for each class of behaviour.

Figure 4.3 shows the results from the experiment. Initially, there is little variation in the robot's behaviour. However, over time the spread of the probability distribution increases. This is because the robot's

True class	**Classification**	**Outcome**
Faulty	Faulty	True Positive (TP)
Faulty	Non-faulty	False Negative (FN)
Non-faulty	Faulty	False Positive (FP)
Non-faulty	Non-faulty	True Negative (TN)

Table 4.1: Possible classifications and their outcomes

trajectory after detecting the arena wall varies due to differences in the IR sensor readings between runs. It can be seen that the spread of the faulty robot's endpoint distribution similarly increases over time.

4.3.2 *Simulated prediction of future behaviour*

In order to predict the behaviour of the non-faulty robot, the simulation is initialised with the same initial pose. The simulated robot's behaviour is similarly stochastic, due to noise added to the simulated IR sensor readings, and a small amount of noise added to the motor speeds. The simulation executes significantly faster than reality, so the same run is repeated 100 times to sample from the distribution of endpoints.

It can be seen from Figure 4.3 that the simulated predictions of the real robot's non-faulty behaviour are not perfect, and the difference between the classes increases over time. This drift occurs due to the reality gap — specifically due to imperfect robot-environment correspondence resulting from the simplified model of the IR sensors.

4.3.3 *Behaviour classification*

Distinguishing between non-faulty and faulty real robot behaviour is essentially a classification problem. It would be possible to simulate multiple different classes of fault, and train a classifier to detect them. However, this would require *a priori* knowledge of all possible failure modes. Instead, it is assumed that only non-faulty behaviour is known from the robot controller, and that any significant deviation from this should indicate the presence of a fault. If a fault is so subtle that it is indistinguishable from the non-faulty class, then it is assumed to be benign and not worth detecting. This approach is an example of one-class anomaly detection [59].

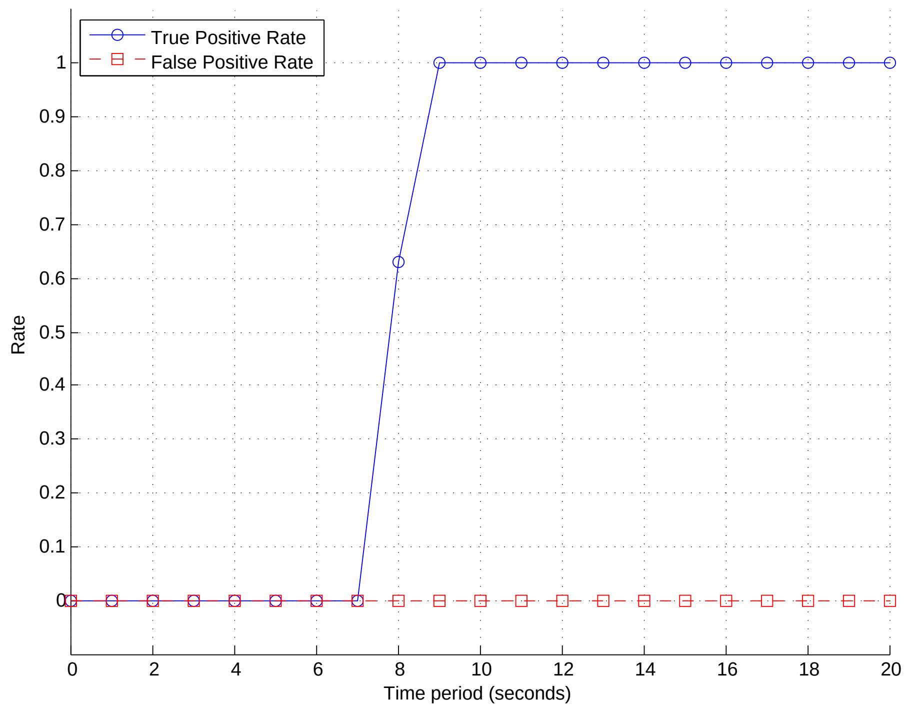


Figure 4.4: TPR and FPR vs time period in unbounded space.

The real robot's position is classified based on a simple uniform distance threshold from the mean of the predicted distribution. This implicitly defines a circular 2D spatial decision boundary. Any test point within this region will be classified as non-faulty, and any point outside it as faulty. From Figure 4.3 it can be seen that the classes could be modelled better using an ellipse, especially after a longer time period, but a circular boundary is sufficient for this initial work.

Table 4.1 enumerates the four possible outcomes of the classification. For any time period after initialisation, each real robot endpoint shown in Figure 4.3 can be classified based on its distance from the mean of the predicted distribution. The classification outcomes are aggregated to produce the total number of true positives, false negatives, false positives, and true negatives at a specific time. The True Positive Rate and False Positive Rate can then be calculated using the following equations.

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN} \tag{4.1}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \tag{4.2}$$

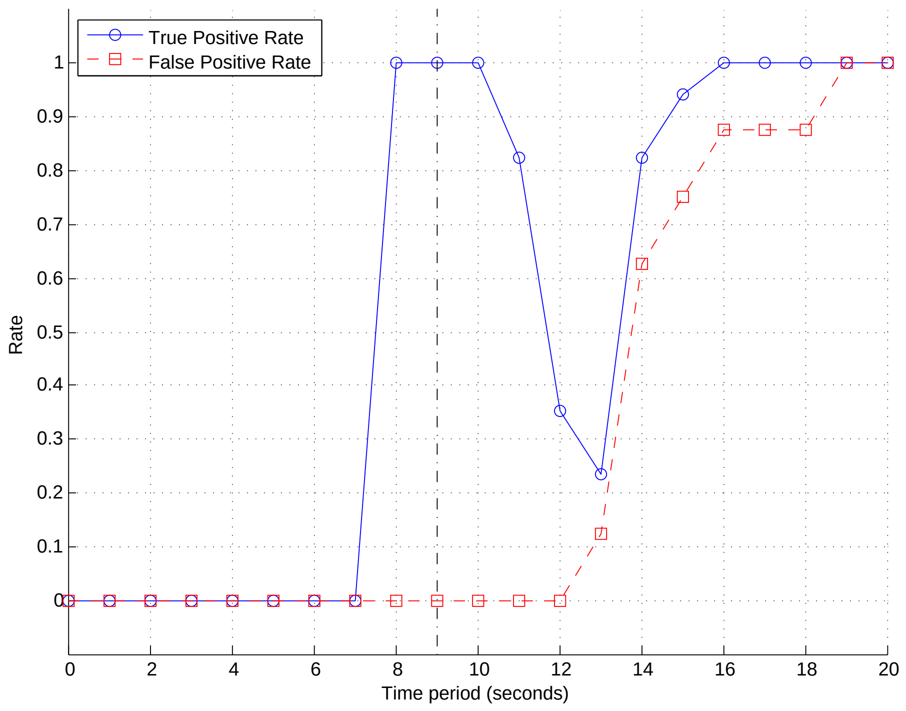


Figure 4.5: TPR and FPR vs time from one particular initial pose. The robot detects the arena wall 9 seconds after initialisation.

Figure 4.4 shows how the TPR and FPR vary over time when the robot traverses unbounded (and empty) space. The arena wall is removed, and the non-faulty robot simply moves in a straight line due to a lack of sensory input. The TPR begins at zero, and quickly increases to 1 between 7 and 9 seconds. This is due to the non-faulty and faulty classes becoming more easily separable as time passes. The TPR will remain at 1 forever, because the classes will simply separate further with time. The FPR remains at zero throughout. This is because the simulator is able to predict the straight line movement of the real robot accurately, so drift is minimal and the non-faulty class remains contained within the decision boundary.

Figure 4.5 shows how the TPR and FPR vary over time when the robot traverses bounded space, starting in the same initial pose as in Figure 4.3. Initially the robot behaves as if it is traversing unbounded space, because the arena walls are beyond the range of its IR sensors. Immediately after initialisation, faulty behaviour is indistinguishable from non-faulty behaviour so the TPR is zero, but quickly increases to 1 after 7 seconds as the classes separate and the faulty class moves outside the decision boundary.

The non-faulty robot starts to detect the arena wall after approximately 9 seconds, and it turns away. This causes its trajectory to intersect that of the faulty robot, and the once-separable classes begin to overlap again. This results in a temporary drop in the TPR due to an increase in the number of false negatives. This short drop in the TPR is relatively benign, as it only briefly reduces the likelihood of detecting a fault when the robot is near the arena wall. If the fault is persistent, then it will be detected once the robot starts moving away from the wall and the TPR recovers as the classes diverge again.

Of greater concern, is the effect of drift on the FPR over time. Figure 4.5 shows that shortly after the robot detects the wall, the FPR beings to increase. This is caused by increasing drift between the simulation and reality, which results in the non-faulty class moving outside the decision boundary that encircles the predicted class, and a rise in false positives. It is desirable to minimise false positives, as they may result in action being taken against a non-faulty robot mistakenly suspected of being faulty, which could be costly.

It is important to note that this increase in the FPR is not observed in unbounded space, and is due to poor robot-environment correspondence. The effect could be reduced by improving the simulated model of robot behaviour, but there will always be some amount of drift, because a simulation cannot hope to model the complexity of the real world in its entirety. Therefore, it is necessary to periodically reinitialise the simulation after a certain time period, to enforce an upper bound on the amount of drift. Without this, it is difficult to determine whether discrepancies between simulation and reality are caused by drift, or by a fault in the real robot.

4.3.4 *Reinitialisation time period*

Selecting an appropriate reinitialisation time period is non-trivial. A long time period allows the non-faulty and faulty classes to separate, making them easier to differentiate, and improves the ability to detect minor faults. Unfortunately, a long time period increases the likelihood of the robot encountering an obstacle before reinitialisation, and therefore runs the risk of increased drift and potentially false positives. A shorter time period is desirable because it minimises drift, and reduces the latency of fault detection, but may only allow major faults to be detected.

Clearly, selection of the reinitialisation time period must trade off multiple objectives. Furthermore, the optimal time period for the motor fault considered here may not be optimal for another class of fault.

The focus here is not on finding the optimal time period, but rather to illustrate that a trade-off exists.

It seems reasonable to suggest that the time period should be chosen under the assumption that the robot is always traversing unbounded space. From Figure 4.4 it can be seen that after a time period of 10 seconds it should be possible to reliably detect the fault. However, there is little benefit in using a time period much longer than this, as the classes are already be separable.

4.4 FAULT DETECTION AT RUNTIME

In the previous analysis, the TPR and FPR were calculated by classifying endpoints generated from repeated non-faulty and faulty real robot runs from the same initial pose. When performing fault detection at run-time, these distributions of data are unavailable. Instead, the simulation is initialised using tracking data and then predicts non-faulty behaviour over the specified time period. The real robot's observed position after the same time period (now a single test point, rather than a distribution) is then classified according to its distance from the predicted mean. Once the classification is complete, the process repeats, and the simulation is initialised with new tracking data. The correctness of the classification is highly dependent on the robot's initial pose relative to the arena wall. Figure 4.6 shows how the distance between the real robot's endpoint and the predicted mean varies with time, with a reinitialisation time period of 10 seconds. Initially, the robot is non-faulty, but a fault is injected after 60 seconds.

When the simulation is first initialised, the robot does not encounter the arena wall within the 10 second time period, so drift is minimal. As it approaches the wall, drift begins to increase because the robot interacts with the wall before the simulation is reinitialised. The level of drift peaks when the simulation is initialised at the point where the robot first detects the wall, as this maximises the amount of post-wall drift that can occur within the time period. As soon as the robot's initial pose advances past the interaction with the wall, drift immediately drops back to minimal levels, as the robot is effectively moving through unbounded space again.

After the fault is injected, the robot's baseline distance from the predicted mean increases. This appears more stable because the circular arena causes the faulty robot's curved trajectory to remain at more

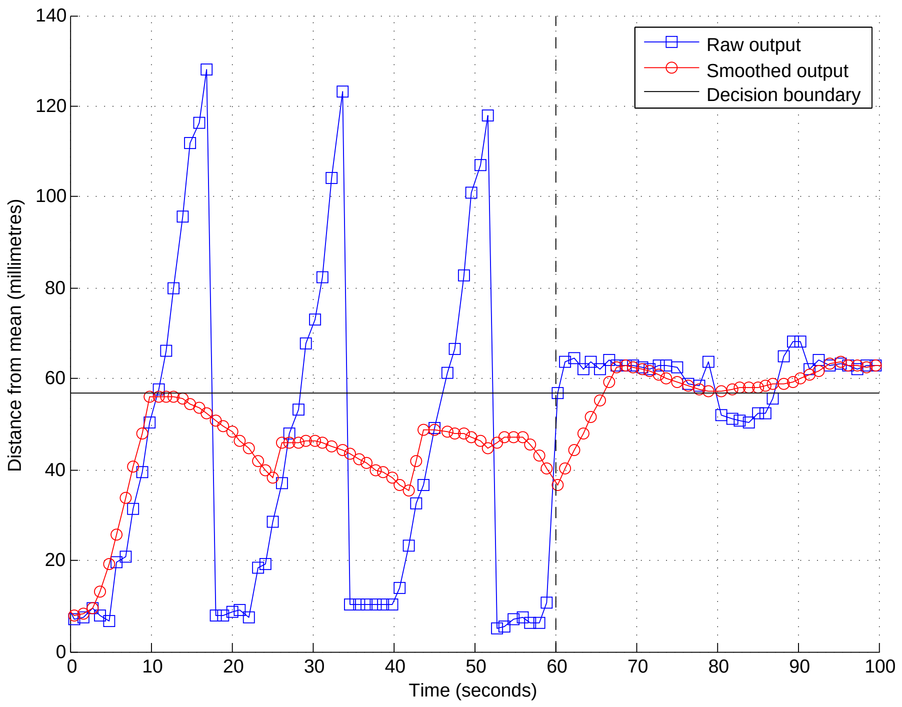


Figure 4.6: Raw and smoothed classifier output (distance between the real robot endpoint and the predicted mean) over time. The reinitialisation time period is 10 seconds. A fault is injected after 60 seconds.

consistent distance from the predicted mean. The brief drop after 80 seconds is caused by an overlap in the classes when the robot reaches the wall. If it were not for the spikes in the classifier output caused by drift when the robot is non-faulty, the non-faulty and faulty classes could be quite easily differentiated. For example, a decision boundary at 40 mm would maximise the TPR, but the spikes in the output would result in many false positives.

Christensen et al. [61] demonstrated that by thresholding a moving average of the output of a fault detector, the number of false positives could be reduced by filtering out spikes in the data. Here, the same technique has been applied to the raw classifier output to produce the smoothed output shown in Figure 4.6. This allows the fault detector to ignore brief anomalies in the data, so that it will only detect persistently faulty behaviour. The smoothing not only helps to prevent increases in the FPR, but also decreases in the TPR. This is because short drops in the classifier output caused by overlapping classes are smoothed out when the robot is persistently faulty. With smoothing applied, the classes can now be differentiated using the decision boundary shown in Figure 4.6. However, note that this ap-

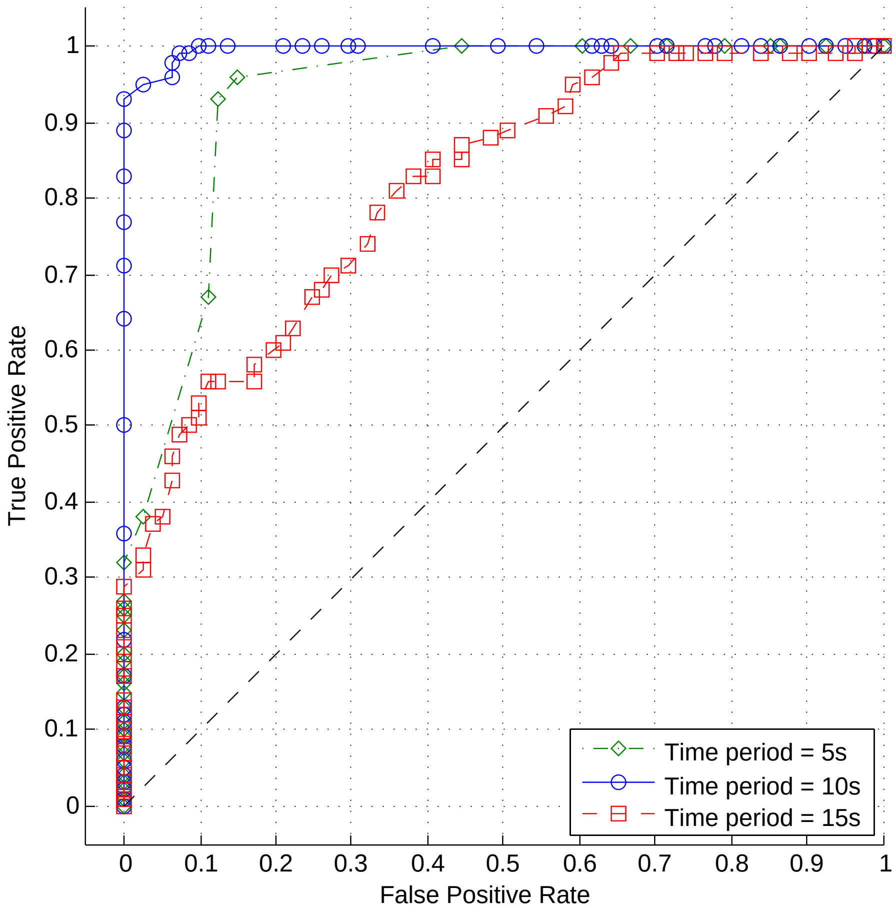


Figure 4.7: ROC curves for reinitialisation time periods of 5 seconds, 10 seconds, and 15 seconds. The analysis is performed on the smoothed output of the classifier.

proach increases the latency of fault detection, and may prevent some intermittent faults from being detected.

4.4.1 *ROC analysis*

Finally, Receiver Operating Characteristics (ROC) analysis was carried out to assess the performance of the classifier. This was achieved by calculating the TPR and FPR for a range of decision boundary sizes. ROC curves for three different time periods are shown in Figure 4.7. Each point on a curve represents a particular TPR/FPR tradeoff produced by some decision boundary size. The diagonal line represents the performance of a random classifier.

When a reinitialisation time period of 5 seconds is used, the classifier's performance is clearly much better than a random classifier. However, the time period is too short as it does not allow the non-faulty and faulty classes to fully separate before reinitialisation. Using a time period of 10 seconds produces almost perfect results for this particular case study, as shown by the larger area under the curve. The decision boundary that gives the highest TPR with no false positives is a threshold of 57 mm from the mean, as shown in Figure 4.6. A time period of 15 seconds causes the classifier to perform much worse, because the longer time period results in a large amount of drift, causing an increase in the FPR.

4.5 OPEN PROBLEMS

Despite the encouraging results from this initial experimental work with a single robot, there are a number of open problems with the proposed fault detection approach when applied in a swarm context.

4.5.1 *Modelling active IR sensors*

The robot's active IR sensors each comprise an emitter and receiver. In order to detect nearby objects, each sensor emits a beam of IR light, and measures the amount of light that is reflected. It is usually assumed that a high sensor reading therefore corresponds to the detection of an obstacle. However, as explained by Quinn et al. [82], referring to active IR sensors as proximity sensors is somewhat misleading, as their raw readings are inherently ambiguous. This is because the amount of reflected IR light is not only a function of the distance to an object, but also the angle at which the IR beam strikes the object, and the proportion of the beam that is reflected.

In a swarm scenario the problem is exacerbated by the fact that IR light emitted by each robot can be directly sensed by its neighbours. The amount of directly sensed IR light will similarly be a function of not only the distance between the robots, but also the angle of the beam and how much of it strikes the receiver. Unfortunately, an IR sensor cannot distinguish between reflected and directly received IR light, nor how many robots or obstacles contributed to the sensed value [82]. Ambient IR light will also be directly received by the sensors, and will affect the robot's behaviour. This effect can be mitigated somewhat by programming the robots to measure ambient IR light levels during initialisation, and then offsetting their sensor readings thereafter. However, this relies on ambient IR levels remaining constant after initialisation, which may not always be the case.

This inherent ambiguity in the IR sensor readings makes the sensors very difficult to model faithfully in simulation, which, in turn, makes accurate prediction of robot behaviour in a swarm context problematic. Robot simulators typically naively model IR sensors as simple proximity sensors, and just return the distance between the sensor and the nearest obstacle. If identical controller code is to be used for both real and simulated robots, then these distances must be translated into raw IR sensor values, which is typically achieved via the use of a look-up-table or mathematical function fitted to sensor readings from a real robot. Unfortunately, as argued by Quinn et al. [82], the assumption that the IR sensor values will only be affected by the distance to obstacles is too simplistic.

As mentioned in Appendix A, a tracking infrastructure could potentially be used to close this reality gap via automated calibration of the simulation. The position and orientation of each robot in a swarm can be recorded while they carry out a particular behaviour. By fusing this data with IR sensor data simultaneously recorded by each robot, the relationship between the sensor readings and the distance/angle to neighbouring robots and obstacles could be determined. However, automated calibration of simulation was not the focus of this research, so was not pursued further.

4.5.2 *Uncertainty of predictions*

A significant issue with predicting future behaviour, is that the behaviour of all neighbouring robots must be predicted simultaneously. Each time a robot avoids an obstacle, or another robot, there is uncertainty in the outcome, thus increasing the spread of predicted endpoints. This is not only a problem in simulation due to the reality gap — real robot behaviour also varies due to noisy sensors and actuators, as shown in Figure 4.3. The illustration in Figure 4.8 shows how the predicted region of endpoints would vary between robots, depending on whether they interacted with other robots.

In a swarm scenario, the number of interactions between other robots and the environment means that the cumulative uncertainty will quickly become unmanageable. This will make it difficult to predict behaviour with the degree of accuracy required to detect faulty behaviour, thus necessitating even more frequent reinitialisation of the internal simulation. As discussed in Section 4.3.4, if the reinitialisation time period is too short, then it will become difficult to detect subtle faults.

4.5.3 *Influence of robots beyond sensor range*

Perhaps even more problematic, is that due to a lack of global perspective, an observing robot cannot simulate the behaviour of robots beyond their sensor range. This is an issue because the future behaviour of their neighbours may depend on the behaviour of robots that are not simulated. For example, consider the scenario in Figure 4.8. In reality, the behaviour of robot *A* is affected by that of robot *D*, which was beyond sensor range of the observer, so was not internally simulated. This will result in the misclassification of non-faulty robots, as their predicted and observed behaviours will be inconsistent.

If the local sensing range were long enough, a potential solution might be to only test robots within some inner radius for faults. This is based on the idea that robots may physically 'insulate' each other from the influence of robots far away. It may then be possible to accurately predict the behaviour of robots close to the observer without requiring global knowledge of the swarm, provided that a sufficient outer ring of other robots are also simulated.

4.5.4 *Incorrect prediction due to faulty robots*

Another significant problem with fault detection via the prediction of future behaviour, is that faulty robots may influence the classification of non-faulty robots. This is because the observing robot can only predict the behaviour of faulty robots by simulating their non-faulty controller code. As an example, consider again the situation in Figure 4.8. If the observer's sensor range were extended to include robot *D*, and robot *D* were afflicted with complete motor failure, then the future behaviour of robot *A* would be predicted incorrectly. This is because robots *A* and *D* would be predicted to avoid each other as shown in the top panel of Figure 4.8, despite robot *D* remaining stationary in reality due to motor failure, allowing robot *A* to continue moving in a straight line. Therefore, although the proposed fault detection method may succeed in detecting faulty robots, non-faulty robots may be incorrectly classified if their behaviour is predicted incorrectly as the result of another robot suffering a fault.

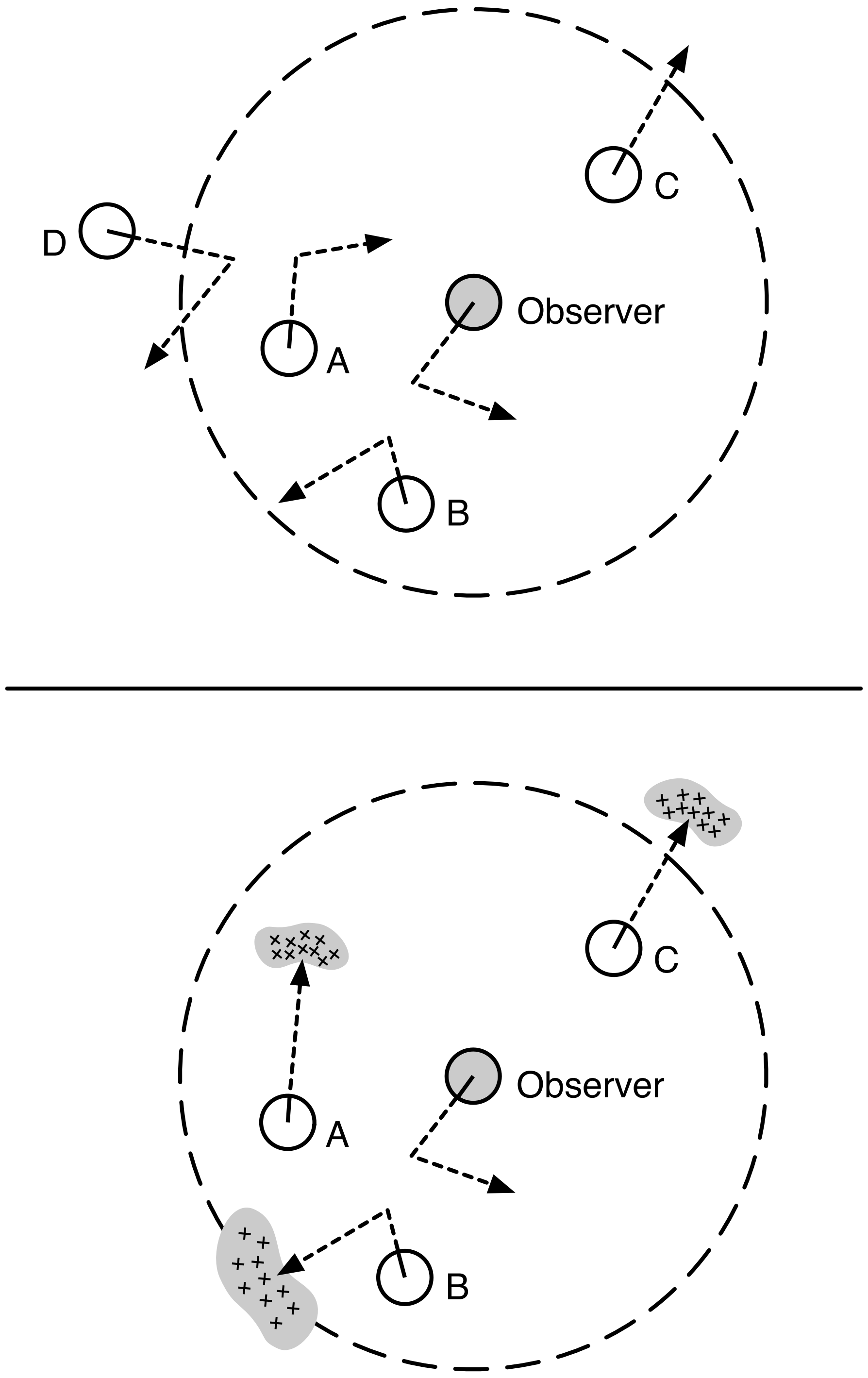


Figure 4.8: Illustration to show the influence of robots beyond sensor range. *Top:* Situation in reality. Robot *D* is beyond sensor range of observer, so cannot be internally simulated. *Bottom:* Observer's internal simulation of its neighbours. The crosses denote the predicted endpoints of the robots. The predicted regions of non-faulty endpoints are shown in grey.

4.6 SUMMARY

This chapter has presented work undertaken to test the proposed fault detection system on a single robot. It was shown that an important choice with such an approach is the selection of an appropriate reinitialisation time period for the internal simulation. This is a non-trivial consideration, as a trade-off exists between minimising drift caused by the reality gap, and detecting faulty behaviour. The experimental results show that there exists an optimal time period that provides the best trade-off characteristic between the TPR and FPR for the partial motor failure used as a case study. However, it is important to note that the optimal time period is likely to be different for each failure mode and robot task.

Unfortunately, predicting future behaviour in a swarm context is much more difficult, and there are a number of open problems with this approach. Although the work of Blum [77] (see Section 3.2.2) demonstrates that predicting future swarm behaviour is possible for the purpose of engineering safer robot controllers, higher precision predictions of behaviour are required to avoid false positives when performing fault detection. This level of precision is difficult to achieve due to the issues outlined in Section 4.5. Consequently, prediction of future behaviour was abandoned in favour of analysing past behaviour, which sidesteps many of these issues, as will be discussed in the next part of this thesis.

Part III

FAULT DETECTION VIA ANALYSIS OF PAST BEHAVIOUR

5 | ANALYSING PAST BEHAVIOUR

The exogenous fault detection system proposed in Chapter 3 was based on the assumption that it would be possible to accurately predict the future behaviour of individual robots in a swarm. Although the work of Blum [77] shows promise in this direction, behavioural predictions must be very accurate to avoid misclassifying non-faulty robots. The required level of accuracy is difficult to achieve given only a single snapshot of the current real-world scenario, due to the inherent uncertainty in the behaviour of each robot. In an attempt to reduce uncertainty, this chapter proposes a variation on the fault detection system presented in the previous chapter, which is instead based on the analysis of past behaviour. This allows concrete observations of neighbouring robots to be collected over a past time window before attempting to discriminate between normal and abnormal behaviour. This chapter also details the experimental infrastructure implemented to carry out fault detection via the analysis of past behaviour. The performance of the fault detector is later assessed in Chapter 6.

5.1 REVISED FAULT DETECTION SYSTEM

Predicting future behaviour is certainly important in the context of the Consequence Engine architecture [76], as robots must be able to evaluate the consequences of their possible future actions so that appropriate choices can be made to avoid catastrophe. However, in the context of fault detection, predicting future behaviour is unnecessary. As shown in Chapter 4, a certain period of real-time must pass before the real robot's true behaviour can be compared to its predicted behaviour. In principle, it does not matter whether the internal simulation is used to predict behaviour at the start or end of this time period. If prediction is performed at the start of the time period, as proposed in Chapter 3, only an initial snapshot of the real-world scenario is available. The advantage of performing prediction at the end, is that the true behaviour of other robots during the time period can also be taken into account to reduce uncertainty in the predictions.

Figure 5.1 illustrates how the analysis of past behaviour can be used to perform exogenous fault detection. In contrast to fault detection via the prediction of future behaviour (described in Chapter 3), where a single internal simulation is used to predict the behaviour of every neighbouring robot simultaneously, a separate internal simulation is used to classify each neighbouring robot. The scenario shown in Figure 5.1 only considers the observer's internal simulation of robot *A*. Once the internal simulation has been initialised, the behaviour of every robot other than robot *A* is 'replayed' based on observations recorded over a past time window. The behaviour of robot *A* is predicted as usual by executing its simulated robot controller, but now the behaviour of the other robots will be identical for each repeat execution of the internal simulator. By predicting the behaviour of robot *A* in isolation, the uncertainty of multiple robots' behaviour is no longer compounded, resulting in more accurate predictions.

An important requirement of this new approach is that each robot shares observation data with its neighbours. As shown in Figure 5.1, the observer's local sensor range only includes robot *A* and two other robots. This is insufficient information to reliably reproduce the behaviour of robot *A*, which will also be influenced by the two robots beyond the observer's sensor range. However, if robot *A* broadcasts its own observations, this allows the observing robot to internally simulate every robot that may have influenced the behaviour of robot *A* during the past time window. Note that although observation data is shared, the approach still remains local and decentralised, because each observer only uses data from its immediate neighbours.

This revised approach overcomes many of the problems associated with predicting future behaviour that were discussed in Section 4.5. By classifying each robot separately using known past observations of the rest of the robots, uncertainty in the predicted behaviour is greatly reduced, albeit at increased computational expense. In addition, robots beyond the shared sensor range are no longer a problem, because any influence they may have had during the past time window has already been observed, therefore they will not affect the predicted behaviour. Similarly, faulty robots will not cause the behaviour of non-faulty robots to be incorrectly predicted, as their past behaviour is known and can be internally simulated. The main drawback of this approach is that separate internal simulations must be executed for each robot to be classified, thus consuming more time and computational resources.

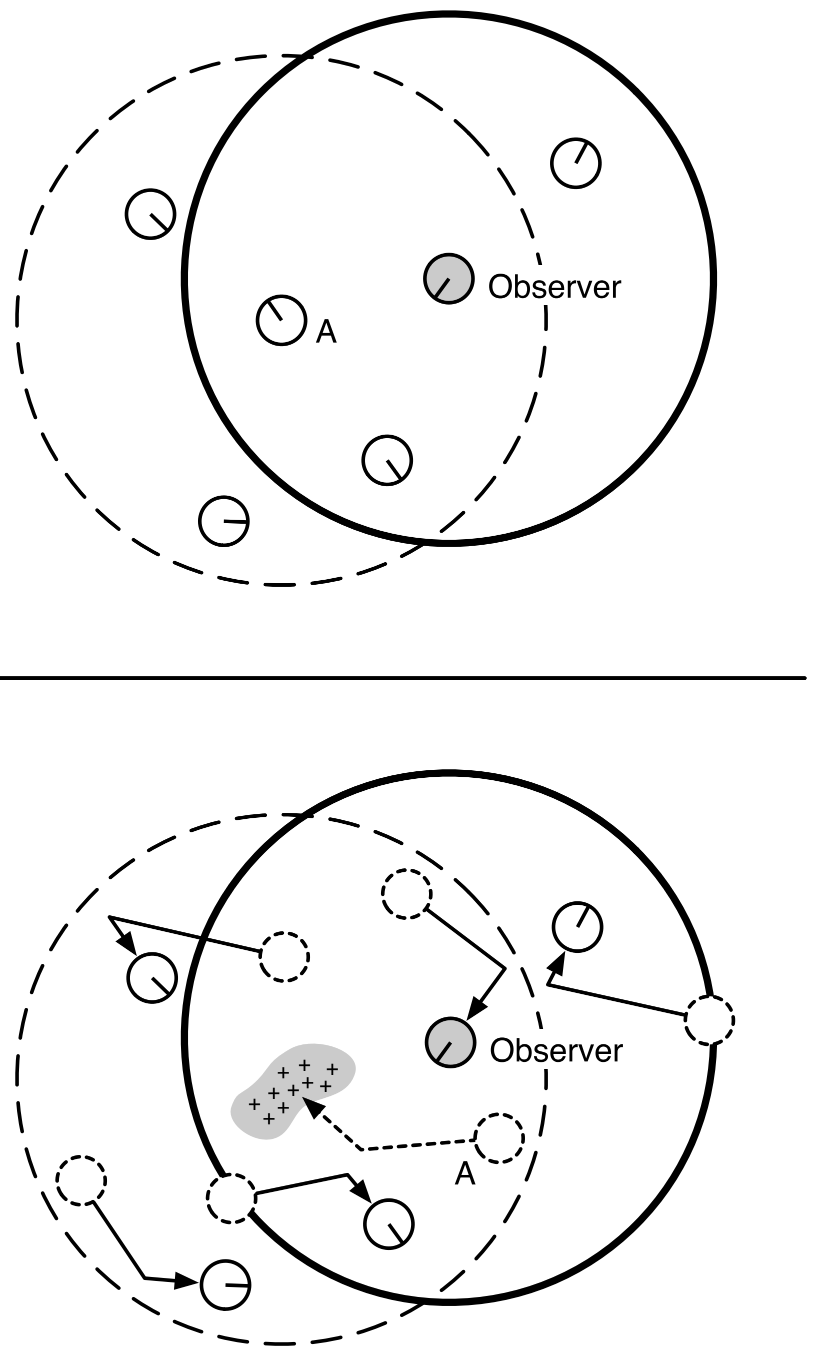


Figure 5.1: Illustration to show how the internal simulation is used to classify robot behaviour. *Top:* External simulation when the fault detection cycle is initiated, showing the true endpoint of robot *A*. *Bottom:* Observer's internal simulation of robot *A* (the focal robot). The replayed behaviour of the non-focal robots over the course of the past time window is indicated by the arrows. The crosses denote the predicted endpoints of robot *A* from 10 repeat runs. The predicted region of non-faulty endpoints is shown in grey.

5.2 ARGOS SIMULATOR

Instead of continuing to work with physical hardware, the decision was made to carry out the remaining experimental work entirely in simulation. This sidestepped the difficulties of accurately modelling real robots in simulation (as discussed in Section 4.5), and removed the constraints of working with dated robot hardware. Implementing the proposed exogenous fault detection system on real robots will ultimately be necessary to demonstrate its value. However, it must first be shown to work in principle, and simulation was a useful tool for rapidly prototyping the ideas presented in this thesis.

The revised fault detection system was implemented in a well-established open-source multi-robot simulator written in C++, called Autonomous Robots Go Swarming (ARGoS) [89, 90], which is specifically designed for the simulation of robot swarms. ARGoS was developed as part of the Swarmanoid project [91], and is still actively maintained by its creator Carlo Pinciroli. It has gained traction in the swarm robotics community in recent years and is currently on version 3.0.0 (used here), which is freely available online[1].

So that the revised exogenous fault detection system could be tested on robot swarms, it was first necessary to select a swarm behaviour as a case study. An aggregation behaviour was chosen, as it provided ample opportunity for the robots to observe and classify each other, allowing fault detection performance to be thoroughly tested. In particular, the ω-algorithm developed by Bjerknes [7] (described in Section 2.1.1) was reimplemented in ARGoS to produce an aggregation behaviour. Bjerknes' original implementation of the ω-algorithm [7] included a symmetry breaking mechanism that allowed the swarm to perform emergent phototaxis towards a beacon placed in the environment. This symmetry breaking mechanism was not required for the aggregation behaviour, therefore it was omitted for this research.

5.2.1 *The e-puck robot model*

In addition to the Swarmanoid robot models (foot-bot, hand-bot, eye-bot), ARGoS has built-in support for the e-puck robot platform. Given that the ω-algorithm was originally implemented on real e-puck robots, the same robot platform was used simulation.

1 `http://www.argos-sim.info`

Although an experimental infrastructure was built with the intention of embedding the exogenous fault detection system on real e-puck robots (see Chapter 3 and Appendix A), the primary aim of this research was not to develop a fault detector that could run on e-puck hardware. Despite the popularity of the e-puck robot platform in swarm robotics research, it was first produced in 2004 and is now over 11 years old, so is no longer representative of the state-of-the-art in physical hardware.

Instead, the aim of this research was to develop a fault detection system that would adhere to the Scalable Swarm Robotics principles of decentralised control and local sensing [26], without being constrained to a particular hardware platform. The cognitive and sensing capabilities of the robots simulated in this research are beyond what can currently be achieved with a real e-puck robot, so the ARGoS model merely serves as a platform to test the fault detector with embodied local sensing and decentralised control.

5.2.2 *External simulation*

In order to gather observation data to test the fault detector with, an artificial analogue of reality was required. This is referred to as the *external* simulation, to distinguish it from the internal simulation each robot uses for the purpose of fault detection. For both the external and internal simulations, each simulation step represents 100 ms of real time. This is the ARGoS default value, and provides enough granularity without incurring unnecessary computational expense.

The external simulation is initialised by randomly placing ten simulated e-puck robots within a 0.8×0.8 m region in the centre of a 3×3 m walled arena, with random initial orientations, as shown in Figure 5.2. Their initial positions and orientations are drawn from uniform distributions. The robots in the external simulation simply perform ω-algorithm aggregation, and carry out fault detection in the background. The exogenous fault detection system is passive, so it has no effect on their behaviour.

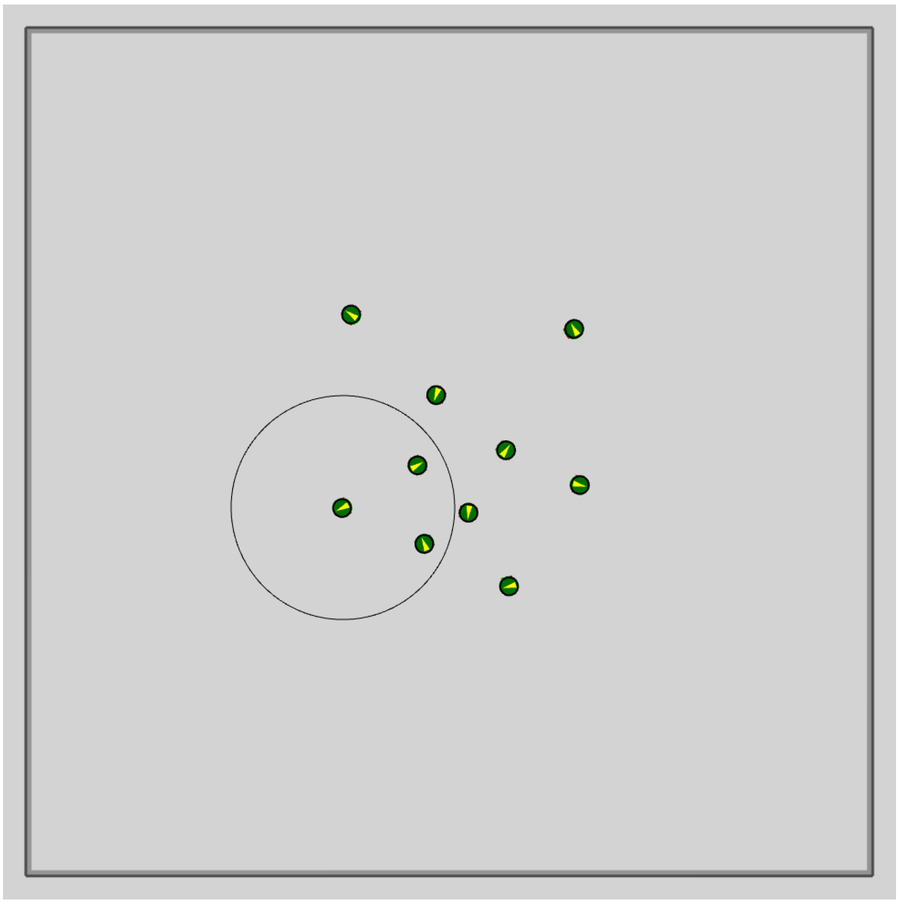


Figure 5.2: Screenshot of the experimental setup in ARGoS for ω-algorithm aggregation. Ten simulated e-puck robots are randomly placed within a 0.8×0.8 m region in the centre of the 3×3 m arena, with random initial orientations. The 40 cm (radius) sensor range of one of the robots is shown.

5.2.3 *Observation data*

As mentioned in Section 4.2.1, the internal state of a robot cannot be determined by an outside observer, given only a snapshot of the system at a particular instant in time. Consequently a stateless controller was used for the experimental work in Chapter 4. However, the ω-algorithm controller uses a finite state machine, so the internal state of a robot must be known in order to initialise the internal simulator. Although it may be possible to infer this from recent observations, it is assumed that each robot broadcasts the full internal state of its controller. For the ω-algorithm aggregation controller, this includes:

– Current state (forward/avoidance/coherence)
– Aggregation timer value (in seconds)
– Desired heading (in degrees)
– Distance turned (in centimetres)

Knowing that a robot is in the forward, avoidance, or coherence state is insufficient to initialise the internally simulated robot controller. A robot executing the ω-algorithm will attempt to perform a coherence manoeuvre when the aggregation timer exceeds its threshold, thus the timer value must be broadcast so that an observer knows when the robot will next attempt to aggregate with its neighbours. Similarly, the observed robot may be in the process of turning on-the-spot towards a desired heading when it is observed. It must therefore also broadcast the heading it is attempting to turn to, and how far it has already turned, so that the turn can be internally simulated by an observer.

At every simulation tick, each robot in the external simulation records data about any neighbouring robots that it observed during that tick. Each observation includes the following:

- Robot ID
- Position
- Orientation
- Controller state

In addition to observations of neighbouring robots, the observing robot records the same details about its own behaviour at each tick. This is necessary for it to instantiate a model of itself in its internal simulation. Without this, its model of the world would be incomplete. It is assumed that the observer proprioceptively senses its own position and orientation (via odometry and an on-board compass), which is subject to the same noise that applies to sensing the position and orientation of other robots (see Sections 5.2.7 and 5.2.8).

This observation data is stored in a circular buffer, the capacity of which is limited to the size of the fault detection time window. The use of a circular buffer means that stale observation data is discarded as new data is collected, ensuring that the memory resources required by each observer is bounded. The amount of memory required for observation data within each slot of the circular buffer (representing a single point in time) is determined by the sensor range of the robot, and the dynamics of the swarm behaviour. The observing robot uses this observation data itself for detecting faults in other robots, but it also shares its observations with neighbouring robots.

5.2.4 *Range and bearing sensor/actuator*

As discussed in Chapter 3, the original intention was to use a tracking infrastructure to implement a virtual sensor for each real e-puck robot, which would allow the robot to observe the behaviour of its neighbours. Given that the experimental work presented in the remainder of this thesis is carried out entirely in simulation, there is no longer any need for a tracking infrastructure, as robot pose data can be recorded directly from ARGoS. However, to make the robot sensor model slightly more realistic, a generic range and bearing sensor/actuator pair was used instead. This allowed the use of features built-in to ARGoS, such as simulating line-of-sight occlusion, and packet loss.

Garattoni et al. [92] recently published a more detailed e-puck model than the one built into ARGoS, which includes a model of the e-puck range and bearing hardware designed by Gutiérrez et al. [87]. The simulation model developed by Garattoni et al. [92] was not used for this research project, as the amount of noise introduced into the simulated sensor readings is very high, rendering the sensor data unusable for the proposed fault detection method. Figure 5.3 shows a comparison of the sensed and actual distance between two simulated robots using the ARGoS model developed by Garattoni et al. [92]. The error at each distance is much higher than originally reported by Gutiérrez et al. [87] for the real hardware — an average of 2.39 cm, and 6.87 cm in the worst case. It remains unclear why there is such a discrepancy between the results, especially given that the simulated noise model developed by Garattoni et al. [92] is based on sensor readings collected from real e-puck robots.

Given the range and bearing of another robot, an observer can calculate that robot's coordinates using simple trigonometry. However, this will be in a coordinate system relative to the observer. Whenever the observing robot moves, its relative coordinate system changes. Thus, in order to track the trajectory of a neighbouring robot over time, it would be necessary to transform each observed pose into the same coordinate system. This transformation would have to account for the observer's own movement, in addition to any movement of the observed robot. For the sake of convenience, it is instead assumed that the robots share a global coordinate system, removing the need to transform observation data between relative coordinate systems. This is especially useful when robots share observation data with each other, which would otherwise need to be transformed.

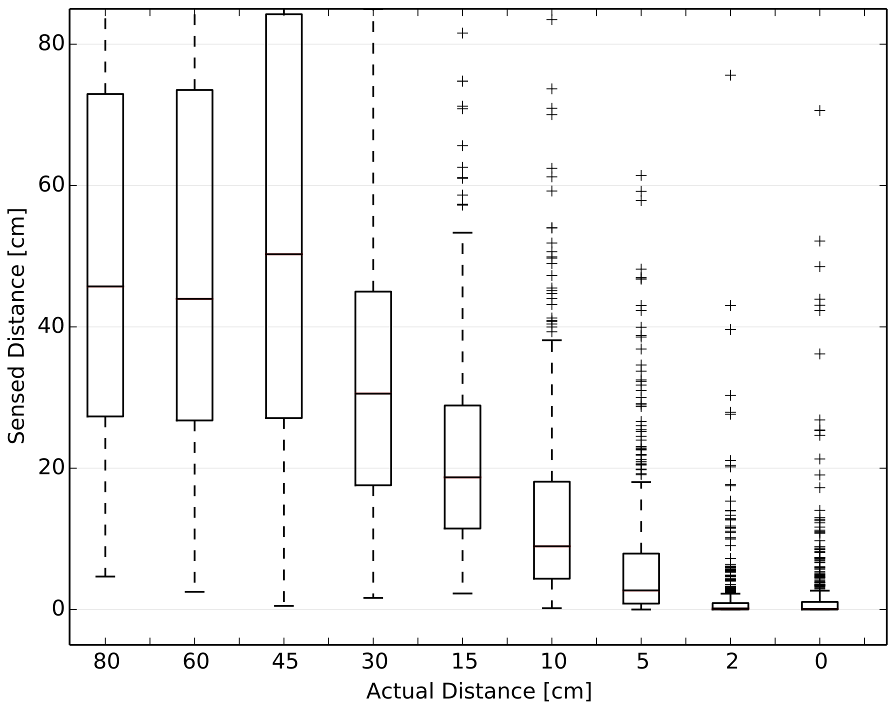


Figure 5.3: Comparison of sensed and actual distance between two simulated robots using the ARGoS model of the e-puck range and bearing board developed by Garattoni et al. [92]. Taken from [92].

Consequently, relative range and bearing data is not used to calculate an observed robot's pose. Instead, the observer senses the absolute coordinates and orientations of neighbouring robots in the global coordinate system. This is similar to the proposed tracking system virtual sensor approach, except that the observations are subject to local sensing restrictions such as line-of-sight and packet loss, as the pose data is only made available if a range and bearing packet is received.

Although the maximum range of the real e-puck range and bearing board is 80 cm [87], the simulated range and bearing sensors/actuators were limited to a range of 40 cm, as shown in Figure 5.2. This range was chosen because it was found to produce stable aggregation with, whilst also ensuring that sensing remained local.

5.2.5 *Range and bearing packets*

The range and bearing sensor/actuator is not only used for localisation — each packet also contains a message payload that can be used to send data between robots. This is a particularly useful feature for the proposed fault detection approach, as it allows robots to broadcast their own ID in each range and bearing packet. An observing robot can then uniquely identify the source of each packet, and use this information to associate each observation with the correct robot across multiple instants in time.

Although not explicitly implemented, it is assumed that the robots broadcast their own orientation (sensed via an on-board compass), as it is not possible to sense the orientation of a robot from range and bearing values alone. If an observer successfully receives a packet from another robot, that robot's absolute orientation in the global coordinate system is made available to the observer.

As mentioned in Section 5.2.3, each robot broadcasts its internal controller state. It also sends its own observations of neighbouring robots in the range and bearing packet. This is necessary for sharing local perspectives, as described in Section 5.1.

5.2.6 *Packet drop probability*

Every time a robot should have received a packet, there is some chance that the packet will be dropped. This may occur in reality due to IR interference, or if multiple robots send data at the same time, resulting in collisions. If a packet is dropped, then the sending robot essentially becomes invisible until another packet that it sends is successfully received, resulting in discontinuous observations. The effect of discontinuous observations on the fault detector is discussed in detail in Section 5.4.3.

The default value chosen for this parameter is a packet drop probability of 1%. This is relatively low, as the fault detector requires mostly continuous observations of neighbouring robots to work effectively. The effect of varying the packet drop probability is analysed in greater detail in Section 6.7.

5.2.7 *Position noise*

As explained in Section 5.2.4, the positions of neighbouring robots in the global coordinate system are made available to each observer (subject to local sensing restrictions). In order to simulate imperfect sensing, noise is added to the sensed position of each robot. This is achieved by generating a noise vector with a length that is randomly sampled from a normal distribution with zero mean, and a standard deviation of $\sigma_{position}$. The angle of the noise vector is randomly from a uniform distribution over the interval $[0°, 360°]$. This vector is added to the true position of a robot, and the resulting coordinates are made available to the observer.

This is equivalent to generating a noisy position by sampling from a bivariate normal distribution that is centred on the robot's true position. This noise model is identical to that built into the generic range and bearing sensor ARGoS model, but is somewhat unrealistic. In reality, the error in the range data would differ from the error in the bearing data, as shown by Gutiérrez et al. [87]. The error in the range data will depend on the reliability of mapping IR signal strength to distances, while the error in the bearing data would depend on the number of IR transceivers used on the range and bearing board. This would result in an 'arc' of noise, rather than noise that is uniformly distributed around the robot's true position. The amount of noise in the sensor readings would also increase with distance from the observer, but this is not modelled in the default ARGoS implementation — the level of noise is constant irrespective of the distance between observer and observee.

Although modelling the error in the range readings and bearing readings separately would theoretically produce a more realistic noise model, any attempt to do so would be arbitrary if it is not based on real sensor data. Unfortunately, as explained in Section 5.2.4 the e-puck range and bearing sensor model developed by Garattoni et al. [92] could not be used for this research project, because the amount of noise they introduce into the sensor readings is too high. In terms of the research question this thesis attempts to answer, the focus is not upon creating an accurate noise model for the simulated sensors. Rather, it is to investigate how well the fault detector copes with slightly unreliable sensor data, and therefore internally simulated robot poses that are inconsistent with 'reality'.

The default value chosen for $\sigma_{position}$ is 1 mm, so 95% of sensed positions will be within a 2 mm radius of a robot's true position (2 standard deviations from the mean). This is a somewhat unrealistically low level of noise when compared to existing range and bearing sensor hardware, however the proposed fault detection approach requires relatively accurate observations of robot behaviour. The effect of varying $\sigma_{position}$ is analysed in more detail in Section 6.7.

Parameter	**Value**
Experiment duration	400 s (6 m 40 s)
External/internal simulation step	100 ms
Range and bearing sensor range	40 cm
Range and bearing packet drop probability	1%
Position noise standard deviation ($\sigma_{position}$)	1 mm
Orientation noise standard deviation ($\sigma_{orientation}$)	1°
Motor noise standard deviation (σ_{motor})	0.1

Table 5.1: ARGoS simulation default parameter values.

5.2.8 *Orientation noise*

As discussed in Section 5.2.5, it is assumed that each robot broadcasts its current orientation, which would be sensed via an on-board compass. In reality, this orientation data would be subject to error, so a simple noise model is used to simulate the error. The noise is generated by sampling a single value from a normal distribution with zero mean, and a standard deviation of $\sigma_{orientation}$. This noise value, which may be positive or negative, is added to the true orientation before it is broadcast by the robot.

The default value chosen for $\sigma_{orientation}$ is 1°, so 95% of sensed orientations will be within $\pm 2^\circ$ of a robot's true orientation (2 standard deviations from the mean). Again, perhaps this is an unrealistically low level of noise, but the fault detector requires accurate observation data to perform well. The effect of varying $\sigma_{orientation}$ is analysed in more detail in Section 6.7.

5.2.9 *Motor noise*

The default ARGoS noise model is used to generate noise for the simulated e-puck's differential drive motors (separately for each wheel). The noise is generated by sampling a single value from a normal distribution with zero mean and a standard deviation of σ_{motor}, and multiplying it by the desired wheel velocity. This noise value, which may be positive or negative, is added to the desired wheel velocity. This model causes the level of noise to increase as the robot's speed increases.

The default value chosen for σ_{motor} is 0.1. Given that the maximum straight-line speed allowed by the ω-algorithm controller is 3.22 cm/s, in the worst case, 95% of wheel velocities will be within ±0.64 cm/s of a robot's desired wheel velocities (2 standard deviations from the mean). The amount of noise applied to the wheel velocities should have no effect on the fault detector, assuming that noise is modelled accurately in the internal simulator.

5.3 REIMPLEMENTING THE ω-ALGORITHM

As mentioned in Section 5.2, the ω-algorithm was implemented in ARGoS to serve as a case study swarm behaviour to test the fault detector on. The goal was not to exactly reproduce the ω-algorithm in simulation, so long as the ARGoS implementation exhibited a reliable emergent aggregation behaviour. Consequently, there are some differences between Bjerknes' original implementation on real e-pucks [7] and the ARGoS implementation, which are detailed in this section.

5.3.1 *Departures from the original implementation*

In Bjerknes' original implementation [7], the real e-puck robots emitted IR light to signal their presence to neighbouring robots. This allowed a robot to infer the location of its neighbours based on which of its IR sensors could detect the emitted IR light. This served a dual purpose — when performing a coherence manoeuvre a robot could estimate the centroid of its neighbours; and, if another robot was detected close by, it would trigger an avoidance manoeuvre.

Unfortunately, in ARGoS it is difficult to simulate this particular use of the IR transceivers. ARGoS assumes that they will simply be used as active proximity sensors, whereby the IR receivers only detect reflected IR light that has been emitted by the sensing robot. IR light directly received from other emitting robots is not simulated by ARGoS. It is notoriously difficult to model active IR sensors in simulation, particularly when they are used for something other than a standard proximity sensor. Rather than attempting to create an accurate model of Bjerknes' IR-based communication in ARGoS, range and bearing sensor data was used to implement coherence and avoidance behaviours instead. Given that this data is required for the proposed fault detection method, it was also used for the ω-algorithm aggregation swarm behaviour. Relative range and bearing data was used by the ω-algorithm controller, as it does not require observations to be correlated over time.

Parameter	Value
Aggregation timer threshold ω	3 s
Avoidance radius	20 cm
Coherence sensor range	40 cm
Maximum motor speed	3.22 cm/s

Table 5.2: Parameter values used for the reimplementation of the ω-algorithm aggregation in ARGoS.

Despite the different source of sensor data, the ARGoS reimplementation of the ω-algorithm is conceptually the same as the original. When a robot performs a coherence manoeuvre, it uses the range and bearing of other robots within sensor range to calculate a vector to the centroid of its neighbours, then turns to the angle of this vector. Similarly, when performing an avoidance manoeuvre, a robot turns such that it faces 180° away from the centroid vector angle. As is consistent with the original ω-algorithm implementation, the simulated e-pucks do not perform obstacle avoidance using proximity sensors. Thus, they will not react to environmental obstacles. The size of the simulated arena is large enough that the swarm aggregates within effectively unbounded space, so no robot will ever reach the walls.

It is important to note that position/orientation noise and packet loss only applies to range and bearing data used by the fault detector. Although this is somewhat artificial, it is critical that the aggregation behaviour is unaffected by this noise, otherwise the effect of noise on the fault detector cannot be isolated during sensitivity analysis.

5.3.2 *ω-algorithm parameters*

The parameter values used for the ARGoS reimplementation of the ω-algorithm are slightly different to those originally used by Bjerknes [7]. This is because Bjerknes tuned the parameter values to produce stable phototaxis behaviour, which requires the swarm to be more tightly aggregated in order for the symmetry breaking mechanism to work properly. Since only aggregation was required for this research project, a set of parameter values were chosen that allowed the robots more space to move around, whilst ensuring stable aggregation. The parameter values that were used are shown in Table 5.2.

5.4 FAULT DETECTOR

Fault detection is performed in *cycles*. In each fault detection cycle, the observing robot uses observation data (direct and secondary) from the past fault detection time window to internally simulate neighbouring robots, and classify their behaviour as either faulty or non-faulty. The fault detector has no memory of past classifications — each fault detection cycle is entirely independent of any preceding cycles and their classification results.

Only robots directly observed at both the very start and end of the time window will be classified. This ensures that the chosen fault detection time window length is adhered to. If this is found to be too restrictive (robots not being classified often enough), then some tolerance could be built in to allow more robots to be classified. Provided that the time difference between the earliest and latest observations of a particular robot is above a certain threshold, classification could be allowed. However, this threshold would need to be set relatively high (e.g. 90% of the time window) to ensure that the chosen time window length is respected.

5.4.1 *Internal simulations*

For each neighbouring robot that will be classified, a separate internal simulation is executed to detect faults in its behaviour. In this internal simulation, the robot being classified is referred to as the *focal* robot. The other robots in the internal simulation are referred to as *non-focal* robots. Although these internal simulations are executed by the same observer, the robots present in each simulation will differ, depending on secondary observation data available from the focal robot.

Figure 5.4 illustrates the differences between these internal simulations. In the observer's internal simulation of robot *A*, the observer uses direct observations of itself, and robots *A*, *B*, and *C*. Secondary observations obtained from robot *A* are used to simulate robots *D* and *E*. Similarly, in the observer's internal simulation of robot *B*, the observer uses direct observations of itself, and robots *A*, *B*, and *C*. Secondary observations obtained from robot *B* are used to simulate robots *E*, *F*, and *G*. Robots *H* and *I* are beyond the sensor range of the observer and robots *A* and *B*, so they are not included in any of the observer's internal simulations.

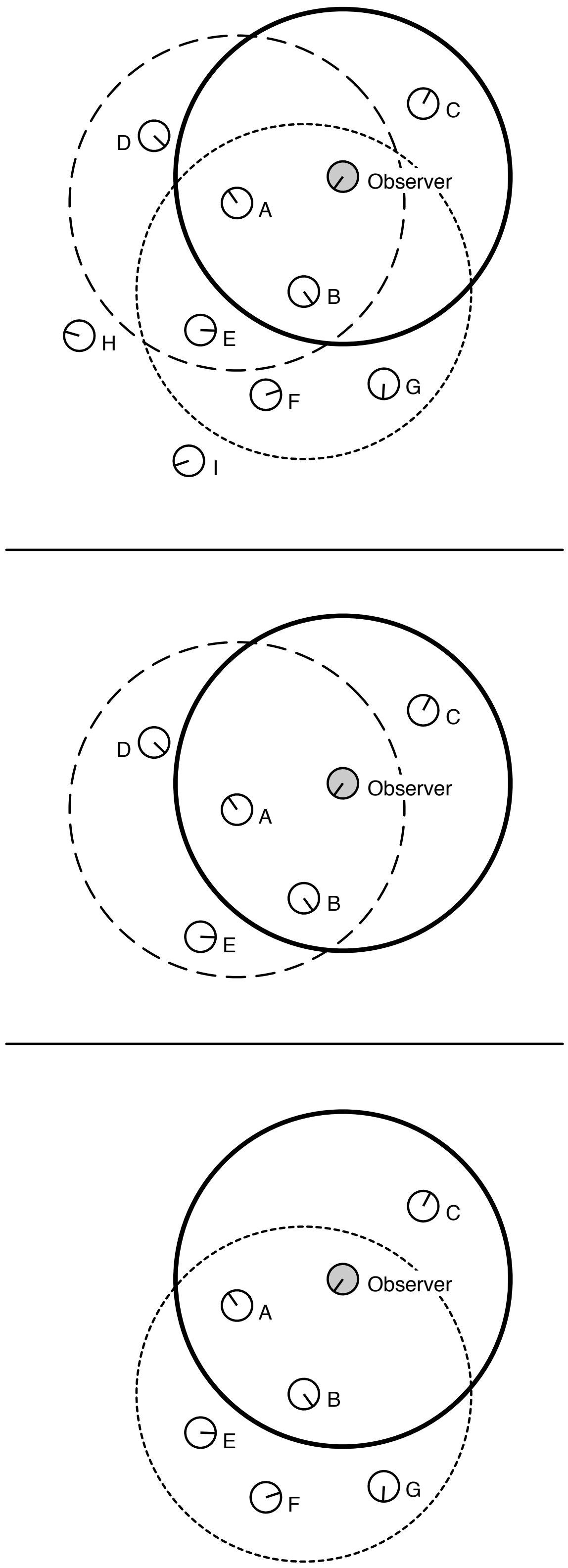


Figure 5.4: Illustration to show the differences between internal simulations. The sensor ranges of the observer, and robots *A* and *B* are shown (solid, dashed, and dotted, respectively). *Top:* External simulation. *Middle:* Final state of observer's internal simulation of robot *A*. *Bottom:* Final state of observer's internal simulation of robot *B*.

No longer constrained by the aim of embedding the internal simulator on real robot hardware, an instance of ARGoS is also used for each observer's internal simulator. This sidesteps issues with developing functionally equivalent code (discussed in Section 3.3.3), as the robot controller code for the external and internal simulations is identical. Note that no attempt is made to create an artificial 'reality gap' between the external and internal simulations, as any discrepancies introduced would be arbitrary. Separate parallel instances of ARGoS can be used for each robot's internal simulation, or a single instance can be shared between all robots sequentially, depending on the computational resources available. Interprocess communication between multiple instances of ARGoS is achieved via shared memory.

5.4.2 *Predicting non-faulty robot behaviour*

The internal simulation is used to predict the bounded region where the focal robot should have ended up, under the assumption that it is non-faulty. If the true endpoint of the focal robot is within this bounded region, then it will be classified as non-faulty. Otherwise, it will be classified as faulty. Figure 5.1 shows how the internal simulation is used to predict non-faulty robot behaviour. First, the position and orientation of each robot, and the internal state of the focal robot's controller, is initialised using observation data from the start of the time window. There is no need to initialise the internal state of the non-focal robots, as their behaviour is predetermined.

At each step of the internal simulation, the position and orientation of each non-focal robot is updated according to the observation data, such that their past behaviour over the past time window is essentially 'replayed'. If direct observation data from the observer is not available for a particular non-focal robot at any simulation tick, then secondary observation data from the focal robot will be used. If no observation data is available from either the observer or the focal robot, the non-focal robot will be removed from the internal simulation. However, it may reappear later in the simulation if more observations of it become available.

The focal robot's non-faulty behaviour is predicted by executing its ω-algorithm controller code, and allowing it to be influenced by the predetermined behaviour of the non-focal robots. Range and bearing packets that would have been received by the focal robot during the time window are also simulated. This ensures that any input that the focal robot's controller would have received in 'reality' is reproduced, so that its non-faulty behaviour can be accurately predicted.

The observation data used to initialise the internal simulation (and to replay the behaviour of non-focal robots) is a combination of direct observations from the observer, and secondary observations received from the focal robot. The internal simulation will therefore include robots that may not have been directly observable by the observer. The observer's observations of itself are included in this observation data. Therefore, the observer's behaviour will also be replayed, in the same way as other non-focal robots. Note that observation data for the focal robot other than its start and end point is ignored — its behaviour is purely determined by running the ω-algorithm controller, and only its true endpoint is required to perform the classification. Any intermediate observations would only be useful if trajectory data were to be taken into account during classification.

5.4.3 *Discontinuous observation data*

In order to reliably replay the behaviour of non-focal robots in the internal simulation, continuous observation data for those robots must be available. However, there are a number of reasons why observation data may be discontinuous. Firstly, line-of-sight is required for robots to observe each other via the range and bearing sensor, so robots may occlude each other. Secondly, even if line-of-sight is established, range and bearing sensing is subject to packet loss, essentially making the observed robot temporarily invisible. Finally, robots on the boundary of sensor range may move in/out of sight multiple times over the course of a fault detection time window.

In the event of discontinuous observations, no interpolation between available data points is performed. Instead, robots simply disappear from the internal simulation when they cannot be observed, and reappear if observed again within the fault detection time window. Discontinuous observation data makes it difficult to reproduce the focal robot's behaviour accurately, and will therefore impact fault detection performance.

5.4.4 *Sampling multiple endpoints*

Due to the stochastic nature of robot behaviour, the internal simulation must be run multiple times to sample from the underlying probability distribution of focal robot's possible endpoints. For each repeat run, the focal robot's behaviour is predicted until the time window of observation data comes to an end. The endpoint of the focal robot is then recorded, and the internal simulation is reinitialised with a different random seed. The resulting sample of endpoints is used to estimate the probability distribution of possible endpoints.

The default number of repeat runs was chosen to be 30. This was found empirically to produce satisfactory results without incurring unnecessary computational expense. A higher number of repeat runs would be necessary if there were greater uncertainty in the predicted behaviour, due to more sensor noise, for example. The effect of this parameter value on the fault detector's performance is analysed in detail in Section 6.7.

5.4.5 *Classification*

In the experimental work presented in Chapter 4, the robot's true position was classified using a simple uniform distance threshold from the mean of the sampled endpoint distribution. This is quite naive, as it implicitly defines a circular 2D spatial decision boundary, which does not take into account the shape of the endpoint distribution. Now that fault detection is being performed in a swarm context, the focal robot's predicted endpoints may be split into multiple disjoint regions, due to interactions with other robots. This necessitates a more sophisticated approach for modelling the (potentially disjoint) region of expected non-faulty robot behaviour.

Given the set of predicted non-faulty endpoints for the focal robot, the aim is to classify the focal robot's true endpoint as faulty or non-faulty. Kernel density estimation (KDE) [93] is a non-parametric method for estimating a probability density function. It is used here to model the shape of the underlying probability distribution of possible non-faulty endpoints, because the shape of the distribution cannot be assumed to be parametric. This is achieved by estimating the probability density function $\hat{p}(\mathbf{x})$ from the sampled endpoints $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ as follows:

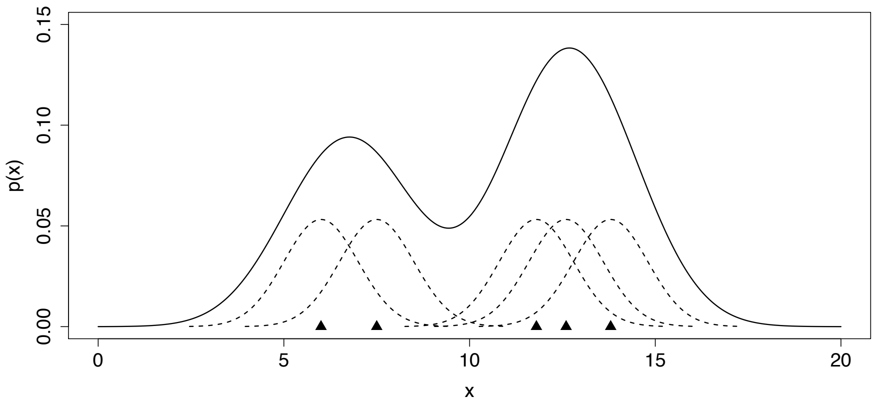


Figure 5.5: Univariate KDE example for 5 data points. The dashed lines represent the Gaussian kernels (described by Equation 5.2) placed over each data point, and the solid line represents the estimated probability density function created by summing over the kernels (described by Equation 5.1).

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{e}_i); \qquad K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-1/2}\, K(\mathbf{H}^{-1/2}\,\mathbf{x}) \tag{5.1}$$

where $\mathbf{x} = (x,\ y)^T$ and $\mathbf{e}_i = (x_i,\ y_i)^T$ are vectors that represent points in 2D space, $\mathbf{H}$ is a 2×2 bandwidth matrix used to apply smoothing, and K is the bivariate Gaussian kernel function [94]:

$$K(\mathbf{x}) = \frac{1}{2\pi}\exp\left(-\frac{\mathbf{x}^T\mathbf{x}}{2}\right) \tag{5.2}$$

These equations essentially describe the process of modelling the underlying probability distribution by summing over Gaussian kernels placed over each of the predicted endpoints. Figure 5.5 presents a visual example for univariate data, which generalises to 2 dimensions when a bivariate kernel function is used.

It is very important to select an appropriate bandwidth matrix, in order to correctly model the underlying distribution. An amount of smoothing must be applied that strikes a compromise between the model being too jagged, and being over-smoothed [94]. The bandwidth matrix is therefore calculated automatically from the endpoint data using a plug-in bandwidth selector.

Defining a decision boundary

Kernel density estimation allows the underlying probability distribution of non-faulty endpoints to be modelled non-parametrically, but this alone does not provide any way of discriminating between faulty

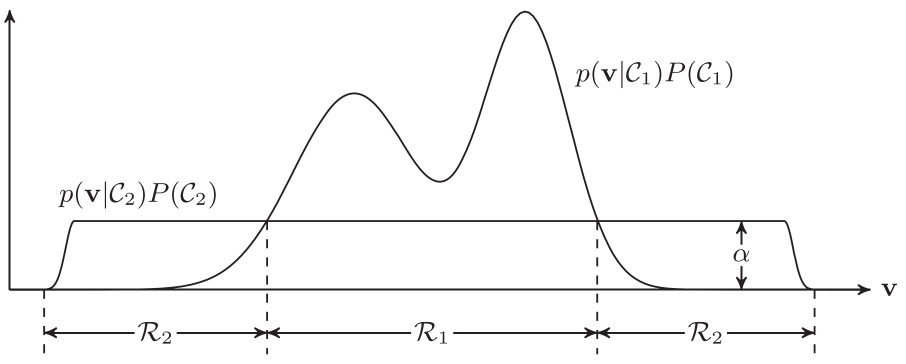

$p(\mathbf{v}|\mathcal{C}_1)P(\mathcal{C}_1)$
$p(\mathbf{v}|\mathcal{C}_2)P(\mathcal{C}_2)$
α
$\mathbf{v}$
$\mathcal{R}_2$
$\mathcal{R}_1$
$\mathcal{R}_2$

Figure 5.6: Bayesian formalism for determining whether the focal robot's true endpoint $\mathbf{v}$ is anomalous. The input space is divided into regions $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathbf{v}$ is classified as an anomaly if it falls within region $\mathcal{R}_2$. The threshold α defines the anomalous distribution. Taken from [66].

and non-faulty behaviour. In order to use this estimated probability density function for binary classification, a boundary must be set to produce a region within which the focal robot is expected to end up if it is non-faulty. Ideally, the area under the probability density function would be integrated to define a boundary within which a certain percentage of the area (and therefore probability of being non-faulty) is contained. Unfortunately, this is a computationally expensive operation, and is difficult to implement for multi-modal distributions. Owens et al. [66] propose the following simple method of applying a threshold to the estimated probability density function, to create a decision boundary for the purpose of anomaly detection.

Given the predicted non-faulty endpoints $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$, the focal robot's true endpoint $\mathbf{v}$ should be classified as non-faulty (class $\mathcal{C}_1$) if it is thought to come from the same distribution as the sampled endpoints. Otherwise, it should be classified as faulty (class $\mathcal{C}_2$) [66]. Every sampled endpoint represents non-faulty behaviour, so belongs to class $\mathcal{C}_1$. The focal robot's true endpoint $\mathbf{v}$ belongs to class $\mathcal{C}_1$ with probability $P(\mathcal{C}_1)$ and class $\mathcal{C}_2$ with probability $P(\mathcal{C}_2)$, and $P(\mathcal{C}_1) + P(\mathcal{C}_2) = 1$. To minimise the probability of misclassification, the true endpoint $\mathbf{v}$ is assigned to the class with the largest posterior probability, so $\mathbf{v}$ is assigned to $\mathcal{C}_1$ if $P(\mathcal{C}_1|\mathbf{v}) > P(\mathcal{C}_2|\mathbf{v})$ [66]. It is stated by Bayes' theorem [95] that:

$$P(\mathcal{C}_i|\mathbf{v}) = \frac{p(\mathbf{v}|\mathcal{C}_i)P(\mathcal{C}_i)}{p(\mathbf{v})} \tag{5.3}$$

Thus, $\mathbf{v}$ is assigned to class $\mathcal{C}_1$ when:

$$p(\mathbf{v}|\mathcal{C}_1)P(\mathcal{C}_1) > p(\mathbf{v}|\mathcal{C}_2)P(\mathcal{C}_2) \tag{5.4}$$

The quantities involving class $\mathcal{C}_1$ are modelled using kernel density estimation on the predicted endpoints. Since only predicted non-faulty endpoints are available, the distribution of faulty endpoints is unknown, so it is simply assumed to be a uniform distribution with probability density α. Given this assumption, Equation 5.4 is equivalent to applying a threshold to the estimated probability density of the predicted non-faulty endpoints [66]. This is illustrated with an orthographic projection in Figure 5.6. Therefore, the classification decision for the focal robot's true endpoint $\mathbf{v}$ is as follows:

$$\text{Classification}(\mathbf{v}) = \begin{cases} \text{Faulty} & \text{if } \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}\left(\mathbf{v} - \mathbf{e}_i\right) < \alpha \\ \text{Non-faulty} & \text{otherwise} \end{cases} \tag{5.5}$$

Setting the α threshold

The α threshold cannot simply be set to a fixed probability density, as the height of the probability density function estimated via KDE will vary greatly depending on the distribution of sampled endpoints. Instead, the density estimate at the coordinates of each endpoint is calculated, then the α threshold is set to the minimum of these densities. This ensures that all of the predicted endpoints will be contained within the thresholded region. The result is a non-linear 2D decision boundary, as shown in Figure 5.7.

The fault detector is parametrised with an α threshold scale factor, to allow a buffer region to be added around the endpoints, to prevent overfitting. A value less than 1 will result in a lower α threshold, and therefore a larger region of predicted non-faulty behaviour. By default, this is set to 1 (no scaling). The effect of varying this parameter value is investigated in Section 6.7.

To perform the classification, the probability density estimate at the focal robot's true endpoint (obtained from the final observation in the time window) is calculated. If this is less than α, then the true endpoint is outside the predicted region of non-faulty behaviour, so the focal robot is classified as faulty. Otherwise, the true endpoint is within the thresholded region, so the robot is classified as non-faulty. This KDE-based classification method was implemented using the `ks` R package [96], which was integrated into the ARGoS controller code using the RInside library [97], allowing direct execution of R [98] code from C++.

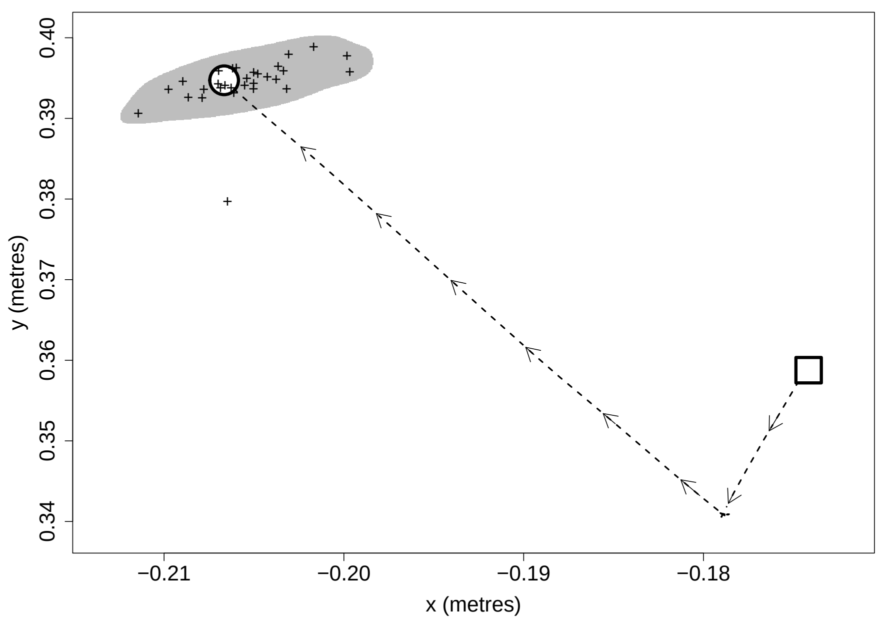


Figure 5.7: Example of classification using a non-linear decision boundary, created by thresholding the kernel density estimate of the predicted endpoint distribution. The crosses represent sampled endpoints, and the grey region delineates the area within which a non-faulty robot is expected to end up. The true trajectory of the focal robot is shown, starting at the square and ending at the circle. In this instance the robot was non-faulty, and was classified correctly (true negative).

5.4.6 *Fault detection time window*

The length of the fault detection time window is an important consideration, as it affects the performance of the fault detector. The time window length is a similar concept to the reinitialisation time period used in the work presented in Chapter 4, except that it applies to the analysis of past behaviour, rather than the prediction of future behaviour. It was shown in Section 4.4.1 that a reinitialisation time period of 10 seconds resulted in the best fault detection performance, however the case study task is now quite different, so this time window length may no longer be the best choice — especially given that fault detection is now being performed in a swarm context.

As discussed in Section 4.3.4, a long time window allows faulty and non-faulty classes of behaviour to separate, making them easier to differentiate, therefore improving the chances of detecting subtle faults. However, a long time window would also suffer from drift due to the reality gap, if implemented on real robots. Although the behaviour of non-focal robots is replayed based on fixed observations, there will still be drift in the focal robot's simulated behaviour, due

Parameter	Value
Internal simulation repeat runs	30
Fault detection time window	5 s
Fault detection cycle interval	5 s
Kernel density estimation α threshold scale factor	1.0

Table 5.3: Fault detector default parameter values.

to the reality gap (if implemented on physical hardware). This must be taken into consideration, even though no reality gap is simulated.

Furthermore, a long time window increases the probability that the focal robot will interact with other robots during the internal simulation. Every interaction with another robot causes uncertainty in the predicted behaviour, which accumulates over time and increases the spread of the predicted endpoints. This, in turn, will necessitate more repeat runs of the internal simulator, to produce a representative sampling of the underlying probability distribution of endpoints.

A short time window will mitigate this cumulative uncertainty, and would limit drift due to the reality gap, but will only allow obvious faults to be detected. The optimal time window length will be different for detecting each type of fault, and will also depend on the swarm behaviour, so there may not be a window length that guarantees good performance in all scenarios. A short time window also increases the chance of a robot being observable at both the start and end of the window, and therefore allowing it to be classified.

A further consideration is that the time window length affects the amount of memory required by each robot, as it controls the number of observations that must be stored. It also affects the computational expense of the internal simulation, because it defines the duration of each internal simulation repeat run. In terms of resource usage, it is therefore desirable to use a short time window.

The default length chosen for the fault detection time window is 5 seconds. Given that the ω-algorithm aggregation timer threshold ω is set to 3 seconds, each robot is guaranteed to either perform a coherence manoeuvre or avoid another robot within this 5 second time window. The effect of different time window lengths on fault detection performance is analysed in Section 6.7.

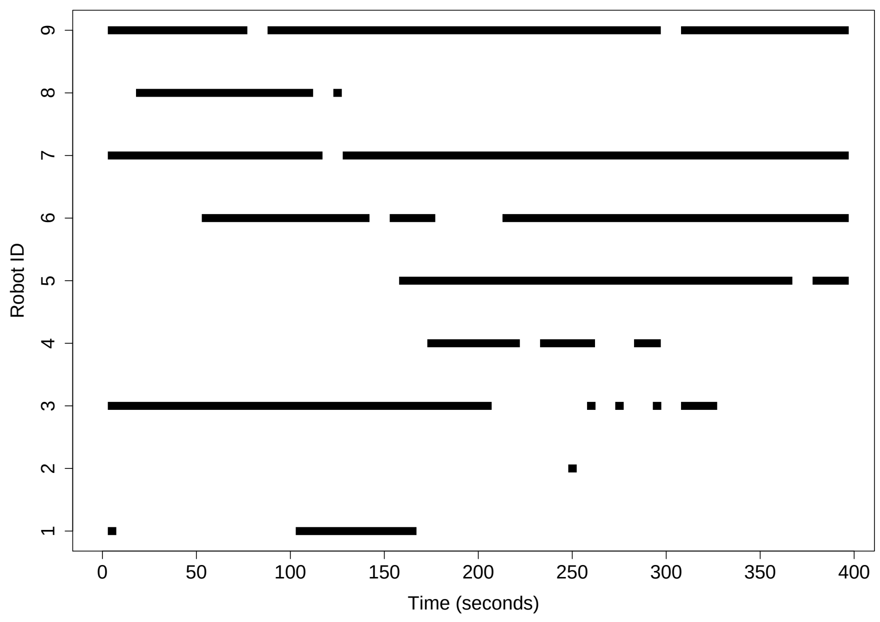


Figure 5.8: Fault detection cycles when each robot in the swarm was classified by observing robot with ID 0, during a single example run.

5.4.7 *Fault detection cycle interval*

In an ideal world, fault detection would be performed at every control cycle. Unfortunately, this would be very computationally expensive, so a larger interval between fault detection cycles is required. The first fault detection cycle begins as soon as a full time window's worth of observations have been collected, and fault detection is performed at regular intervals thereafter. It is important this interval does not exceed the length of the fault detection time window, otherwise there will be times when an observer does not monitor the behaviour of its neighbours, which may result in faulty behaviour being missed.

The default cycle interval was set to 5 seconds, to match the default fault detection time window length, allowing robots to constantly monitor the behaviour of their neighbours with the least computational expense. The trade-off being that the latency of fault detection will be at least the length of this interval. If lower latency were required, then the interval could be reduced (at additional computational expense), such that the time windows overlap. A shorter cycle interval will also increase the likelihood of observing a particular robot at both the start and end of a time window, and therefore being able to classify it.

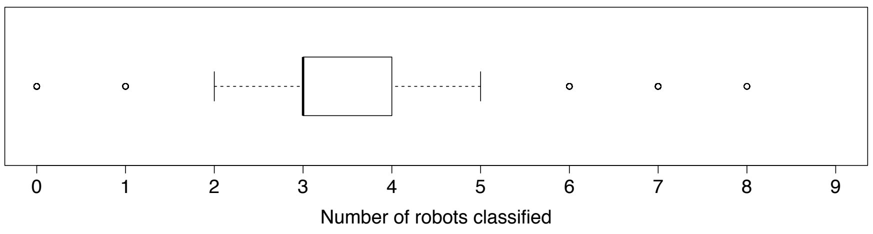


Figure 5.9: Number of robots classified in each fault detection cycle, from the perspective of any observer, across 100 repeat runs.

5.5 EXPERIMENT DURATION

In order to thoroughly test the fault detector's performance, the external simulation must be run for long enough that any faulty robots are classified by their neighbours several times. The required experiment duration will depend on the swarm algorithm, the speed of the robots, their sensing range, the fault detection time window, and fault detection cycle interval. Figure 5.8 shows an example of how often an observing robot is able to classify the other robots in the swarm. The robots that were classified often remained adjacent to the observer throughout the experiment. Those that were classified relatively infrequently changed positions in the swarm, and drifted in and out of the observer's sensor range. Although some robots are rarely classified by this observer, they will likely be classified by another observer with a different perspective. Figure 5.9 shows that, on average, an observing robot will classify 3 neighbours in each fault detection cycle. Depending on a robot's location in the swarm, it may be able to observe more robots, but the sensing remains local.

An appropriate experiment duration was determined empirically by running the external simulation 100 times with different random seeds, for 6,000 ticks (10 minutes of real time), and calculating the cumulative number of classifications of each robot at every tick. Figure 5.10 shows that an experiment duration of 400 seconds (6 minutes and 40 seconds) is sufficient for any given robot to be classified at least 100 times by its neighbours, in the worst case, regardless of the initial position/orientation of the robots. This provides enough classification data to evaluate the fault detector's performance, so using a longer experiment duration would not have much benefit. With an experiment duration of 400 seconds and a fault detection interval of 5 seconds, each observing robot will perform 80 fault detection cycles before the experiment terminates.

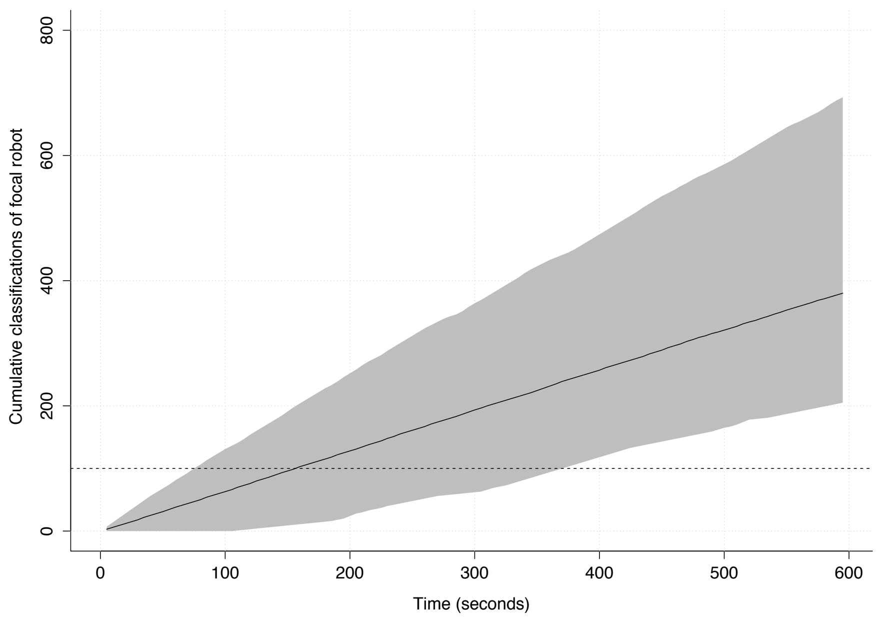


Figure 5.10: Cumulative classifications of any particular robot over time. A robot is only classified if it was observed both at the start and end of a fault detection cycle. The solid black line represents the median of 100 repeat runs (every robot in the swarm is analysed). The grey region represents the minimum and maximum values.

5.6 fault injection

Faults must be injected into robots in the external simulation, so that the fault detector's performance can be tested. However, it is important that these faults do not cause any robots to become lost from the swarm, as an absence of neighbouring robots would mean that behaviour cannot be classified.

5.6.1 *Failure modes*

The following 7 failure modes were devised to test the fault detector:

Motor (complete)

In the event of complete motor failure, both of the robot's wheel motors become completely unresponsive, and the robot is unable to move. The robot will continue to communicate with neighbouring robots via its range and bearing sensor/actuator. This failure mode is implemented by intercepting the desired wheel velocities set by the robot controller, and setting them both to zero before the motor speeds are updated.

Motor (partial)

A partial motor failure causes the robot's right wheel motor to operate at only 50% of its intended speed. This is implemented by intercepting the desired right wheel velocity set by the robot controller, and scaling it by half before the motor speeds are updated. This failure mode will cause the faulty robot to veer off to the right when attempting to move in a straight line.

Sensor (complete)

Complete sensor failure causes the robot's range and bearing sensor to stop working, such that it cannot receive packets from other robots — effectively making the faulty robot completely blind and unable to avoid neighbouring robots that come too close. In the absence of sensor data, the ω-algorithm controller will cause the robot to simply turn 180° whenever it attempts to perform a coherence manoeuvre, as it will not be able to determine the centroid of its neighbours. Therefore, this failure mode will cause the robot to simply move back and forth with a period of ω, ignoring neighbouring robots.

Sensor (partial)

In the event of partial sensor failure, the robot's range and bearing sensor will have a blind-spot in its field of sensing. The robot will be unable to sense neighbouring robots between -45° and 45 °, where 0° is the robot's current heading. In terms of the effect on ω-algorithm aggregation, this means that the robot will perform coherence manoeuvres with erroneous centroid calculations, and will fail to avoid robots directly in front of it.

Communication failure

Communication failure corrupts the internal state data broadcast in the range and bearing packet. Each internal state variable (listed in Section 5.2.3) is set to a random value, uniformly selected from the range of possible values that variable can take. The rest of the data broadcast by the robot is unaffected. This is perhaps a little contrived, but it allows the effect of random state data on the fault detector to be tested. Note that this failure mode will not affect the robot's behaviour, as internal state data is only used for fault detection.

Controller failure

When a robot suffers controller failure, there is a 10% chance at every control step (100 ms) that the robot controller will instantaneously transition to a random state (forward/avoidance/coherence). This failure mode will cause the robot to sometimes behave erratically, as its current intended behaviour may be interrupted.

Power failure

At every control step there is a 5% chance that a power failure will occur, which will cause both of the robot's motors to stop working for 2 seconds, then resume normal operation. This failure mode is typical of the behaviour observed in real e-puck robots when their battery power is low. There will be enough power to allow inter-robot communication, but not enough to power the motors reliably, resulting in a start-stop behaviour.

5.6.2 *Injecting faults*

The faults to be injected during a particular experiment are specified via the ARGoS configuration file. Each fault is listed separately, and must either specify a particular failure mode, or that the failure mode should be selected at random. The robot that the fault will be injected into is randomly selected from the pool of non-faulty robots. This means that a single robot will never suffer multiple simultaneous failure modes. Following Tarapore et al. [12], faults are injected from the outset of the experiment and persist throughout, providing the most opportunity for the faults to be detected by neighbouring robots.

5.7 SUMMARY

This chapter introduced a revised version of the exogenous fault detection system first proposed in Chapter 3. Instead of using an internal simulation to predict future behaviour, each observing robot now uses it to analyse the past behaviour of neighbouring robots. This new approach reduces uncertainty by reproducing the behaviour of each robot in isolation, in the context of known past events. Although computationally more expensive, this technique allows non-faulty endpoints to be predicted with greater accuracy, thus affording more reliable fault detection.

This chapter has also detailed the implementation of the fault detection system in simulation, and the experimental infrastructure required to test its performance. The failure modes defined in Section 5.6.1 are used in the next chapter, to analyse the fault detector's performance and sensitivity to certain parameters.

6 FAULT DETECTION PERFORMANCE

This chapter presents the results of experiments designed to assess the performance of the fault detection system described in Chapter 5. The fault detector's ability to detect each failure mode listed in Section 5.6.1 is tested, as well as its tolerance to multiple faults of random types. The results of scalability and global sensitivity analyses are also presented.

6.1 QUANTIFYING PERFORMANCE

The efficacy of a fault detection system should be judged not only on its ability to correctly detect faulty robots, but also its ability to avoid misclassifying non-faulty robots. It may be costly, in terms of both time and energy resources, to initiate a recovery mechanism in response to the detection of a fault. If a robot is genuinely faulty, then this cost may be justified. However, if recovery mechanisms are triggered due to the misclassification of non-faulty robots, this may prove more costly than not performing fault detection at all.

The fault detector's performance is measured in terms its True Positive Rate (TPR) and False Positive Rate (FPR), as defined in Section 4.3.3. The TPR concerns only classifications of faulty robots (true positives and false negatives), while the FPR pertains only to classifications of non-faulty robots (false positives and true negatives). Random classification will result in average TPR and FPR values of 0.5. Therefore, a TPR value higher than 0.5 indicates better than random performance when classifying faulty robots. Similarly, a FPR value lower than 0.5 indicates better than random performance when classifying non-faulty robots. A perfect classifier would have a TPR of 1 and an FPR of 0, however this is unrealistic to expect in practice.

Each observing robot carries out exogenous fault detection independently of its neighbours, therefore the fault detection performance will be different for each robot in the swarm, and will depend on their own unique perspective. As discussed in Section 5.5, a particular observer may have little opportunity to classify faulty robots before the experiment terminates. This means that insufficient classification data (true positives and false negatives) will be available to properly as-

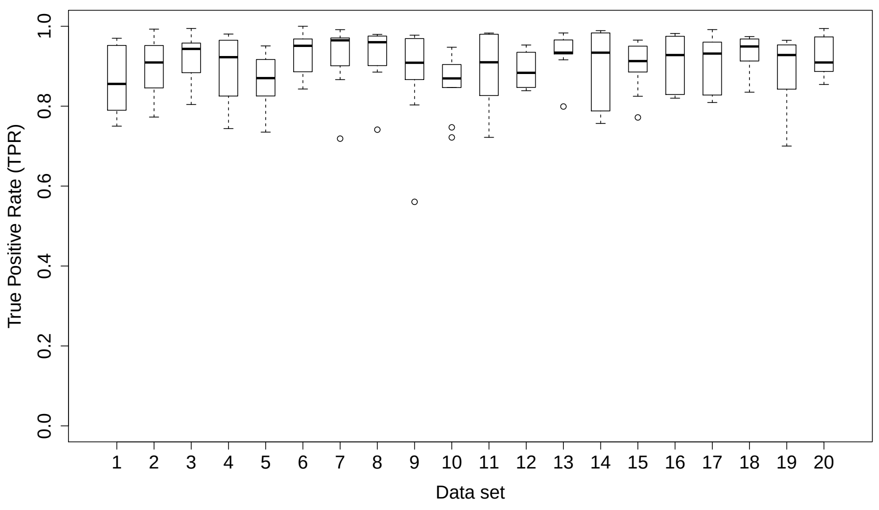


Figure 6.1: Variation in the TPR distributions when a sample size of 10 is used, and 3 faults of random types are injected. Every box represents 10 repeat runs of the simulation, each executed with a different random seed.

sess the TPR for some observers. With a different random seed, they might classify many faulty robots over the course of the experiment, however it makes little sense to aggregate the performance of a particular observer over multiple repeat runs, as every robot is identical, and will start from a random initial position and orientation every time. Instead, the overall performance of the fault detection system is assessed by collating classification data from all of the non-faulty robots in the swarm, across multiple repeat runs, and calculating the TPR and FPR from this aggregated data. This provides a performance measure that is independent of any particular observer's perspective.

6.2 CONSISTENCY ANALYSIS

Both the external and internal simulations are stochastic, due to the noise applied to the robots' sensors and actuators. Consequently, even if the robots start with the same initial positions and orientations, their behaviour will change when the internal simulation is initialised with a different random seed. In addition to this variation due to noise, the initial positions and orientations of the robots in the external simulation are randomised for every repeat run, to ensure that the results obtained are not an artefact of a particular initial configuration. Furthermore, faults are injected into robots selected at random, and may even be of a random type. All of these random effects mean that even if the same set of parameter values are used, any two executions

Effect size	Large	Medium	Small	None	Small	Medium	Large
***A* test score**	< 0.29	< 0.36	< 0.44	0.5	> 0.56	> 0.64	> 0.71

Table 6.1: Relationship between A test score and effect size [101].

of the external simulation (with different random seeds) may result in very different fault detection performance. It is therefore necessary to perform several repeat runs, in order to obtain a representative measure of the system's performance.

To determine an appropriate number of repeat runs, a technique called *consistency analysis* was used. This technique was originally developed by Read et al. [99] in the context of agent-based simulations of biological systems, however the concepts generalise to the swarm robotics experiments carried out for this research project. Consistency analysis works by first generating several distributions of experimental results, each containing the same number of repeat runs, which use the same set of parameter values. The analysis then compares these distributions to assess how similar they are. By varying the sample size that is used to generate each distribution, the number of repeat runs required to produce statistically consistent distributions may be determined [100]. This ensures that the experimental results can be attributed to the to the fault detection system, and its particular parametrisation, rather than any random aspects of the simulation.

Following the recommendation of Read et al. [99], 20 different distributions were generated to assess each sample size. As an example, Figure 6.1 shows the amount of variation in the TPR distributions when a sample size of 10 is used, and 3 faults of random types are injected. Each repeat run was executed with the default parameter values summarised in Tables 5.1, 5.2, and 5.3 (see Chapter 5). Given that the same parameter values were used for each repeat run, the differences in the fault detection performance are attributable to the different fault detection scenarios that were generated by each random seed. Larger sample sizes result in greater similarity between the distributions, thus mitigating the influence of random effects.

The consistency analysis proceeds by comparing the first distribution to distributions 2-20, using the Vargha-Delaney A test [101]. This is a non-parametric effect magnitude test that compares two distributions and returns the probability that a randomly selected sample from one distribution will be larger than a sample randomly selected from the other. A probability of 0.5 indicates that there is no difference between the two distributions, while values above/below cer-

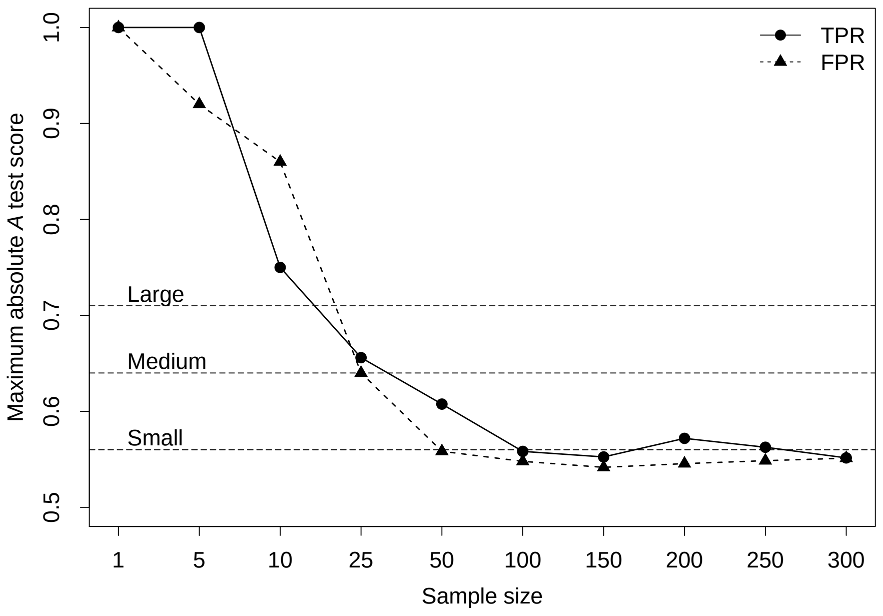


Figure 6.2: Consistency analysis results for the TPR and FPR of the fault detection system. Only absolute A test scores are plotted, as the sign is irrelevant - only the magnitude of the effect is important. Small, medium, and large effect size thresholds are shown.

tain thresholds (listed in Table 6.1) indicate either a small, medium, or large difference. A sample size should be selected that minimises the maximum absolute difference between the first distribution and the remaining 19 distributions.

The analysis was carried out with the Spartan R package [100], which implements the statistical techniques first proposed by Read et al. [99]. The results presented in Figure 6.2 show the relationship between sample size and the effect of simulation stochasticity on the fault detection performance. A sample size of 100 repeat runs was chosen, as it strikes an appropriate balance between computational expense and consistency in the results.

It is worth noting that the consistency of the results will vary depending on the type and number of faults injected, and the parameter values used for a particular experiment. The analysis presented here is specific to 3 faults of random types, and the default set of parameter values. Ideally, consistency analysis would be re-run for every experimental condition, to determine an appropriate sample size. Unfortunately, this would be infeasible, as testing sample sizes of up to 300 requires 6,000 repeat runs, each of which take at least an hour to complete. Even when run on a high performance computing cluster, this is prohibitively computationally expensive. Consequently, it cannot be guaranteed that a sample size of 100 will always produce results as consistent as those presented here if the experimental conditions are changed. However, it should be sufficient for the analyses carried out in this chapter.

6.3 SINGLE FAULTS

To assess the fault detector's ability to detect each failure mode defined in Section 5.6.1, a single fault of each type was tested in isolation. Based on the results of consistency analysis, 100 repeat runs were performed for each failure mode, with the fault injected into a random robot in each run. Figure 6.3 shows the resulting TPR and FPR for each failure mode.

The FPR is unaffected by the type of fault injected, because false positives and true negatives can only come from classifications of non-faulty robots. When a non-faulty robot is being classified, it will be the focal robot of an observer's internal simulation. Thus, any faulty robot that could affect the classification of the focal robot will be a non-focal robot. Only inconsistencies in the observed position or orientation of the faulty robot could therefore influence the classification. If a robot were to fail in such a way that its range and bearing actuator completely stopped working, then it would become invisible to neighbouring robots. This would affect the classification of non-faulty robots, and consequently the FPR, as the faulty robot would not be modelled in any internal simulations.

The rest of this section will discuss the effect of each failure mode on the fault detection system, and the resulting influence on the TPR.

Motor (complete)

Winfield and Nembrini [6] and Bjerknes [7] have shown that this failure mode is the most detrimental to overall collective behaviour when translation of the swarm is required, in the context of the β-algorithm and ω-algorithm. Yet, it is reliably detected by the proposed fault detection system with a median TPR of 0.99, because the discrepancy between the expected and observed behaviours is quite extreme. The faulty robot will remain completely stationary, but its predicted non-faulty behaviour will be to move away from its starting point. However, note that this is specific to the ω-algorithm. In a different swarm behaviour, where the robots perhaps do not move as much, it may

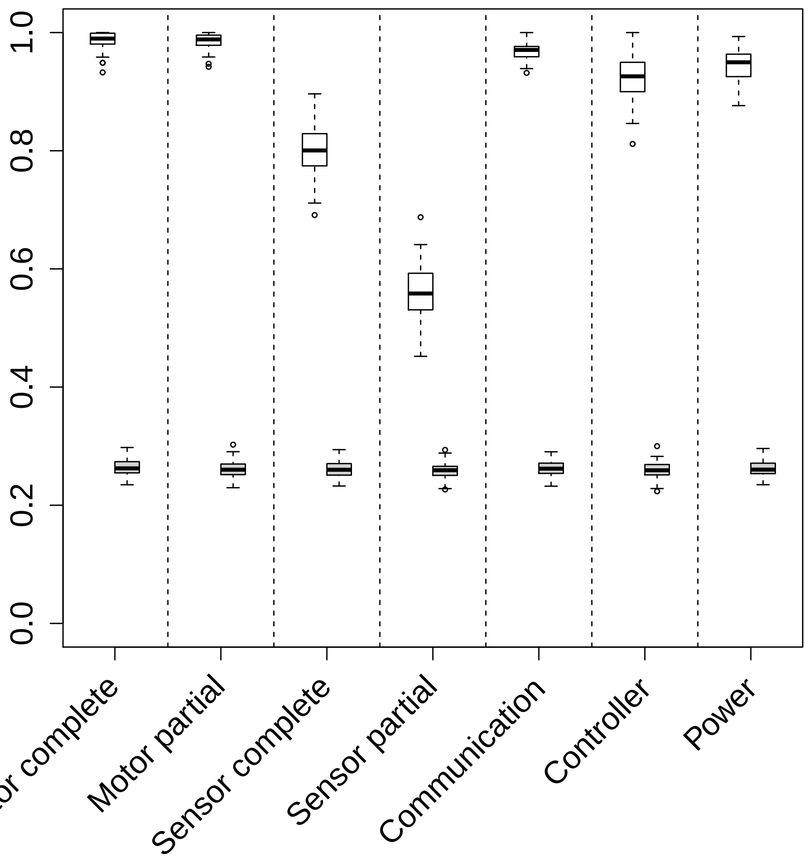


Figure 6.3: TPR (white) and FPR (grey) for a single fault of each type.

be more difficult to distinguish between motor failure and non-faulty behaviour.

Although the faulty robot is immobile, its internal state will still update according to the ω-algorithm controller's reaction to sensory inputs. Therefore, when an observer attempts to internally simulate the faulty robot, the internal state may indicate that it is supposed to be turning on the spot. If the fault detection time window is too short, then there may not be time for the predicted region of non-faulty behaviour to move far enough from the starting point, resulting in false negatives, which will drag the TPR down. False negatives will also occur if the internally simulated focal robot doubles back on itself, and ends near its starting position at the end of the time window. For the default parametrisation of the fault detector, these are rare occurrences, so the TPR remains high.

Motor (partial)

Figure 6.3 shows that partial motor failures can be detected reliably with a median TPR of 0.99. This fault strongly affects the robot's behaviour, making it easy to distinguish from the expected non-faulty behaviour. The faulty robot's trajectory will veer off to the right, causing it to fall outside the region of predicted non-faulty endpoints, resulting in a high proportion of true positive classifications. Due to interactions with neighbouring robots throughout the fault detection time window, there may be times when the true endpoint of the faulty robot falls within the region of expected non-faulty behaviour. However this is a rare occurrence, so the overall TPR is high.

Sensor (complete)

Complete sensor failure is detected with a lower median TPR of 0.80. This is partly because the resulting faulty behaviour of the ω-algorithm controller is similar to that of the expected non-faulty aggregation behaviour, resulting in many false negatives. However, this failure mode is also difficult to detect because it affects the observing robot's internal simulation.

Normally, the focal robot would be able to share its own observations with the observer, to ensure that the internal simulation contains the necessary information to reproduce its behaviour. However, given that the faulty robot is essentially blind, this secondary observation data is unavailable. Consequently, the observing robot must rely solely upon its own local observations of its neighbours to internally simulate the faulty robot's expected non-faulty behaviour. The internal simulation assumes that the focal robot can sense other robots, so it cannot distinguish between the absence of observations of a particular robot due to faulty sensors, or because there was actually no robot within sensor range to be observed. The expected and observed behaviours of the faulty robot will therefore be very similar, resulting in false negatives, unless there are robots within the observer's own sensor range that influence the internally simulated expected behaviour. However, the focal robot should at least react to the observer, as they must be within sensor range of each other for classification to be initiated. This means that the faulty behaviour can often still be distinguished from non-faulty behaviour.

Sensor (partial)

As shown in Figure 6.3, partial sensor failure is only detected with a median TPR of 0.56. This is primarily because only a quarter of the robot's field of sensing is affected by the fault, so the effect on the robot's behaviour is subtle, which makes it difficult to distinguish from non-faulty behaviour.

As with complete sensor failure, this fault will affect the availability of secondary observation data. The faulty robot will be unable to sense some of its neighbours, so these robots will not be present in the observer's internal simulation, unless they have been directly observed. This means that the expected non-faulty behaviour may be consistent with the observed faulty behaviour, resulting in false negatives, and therefore a low TPR.

Communication failure

This is perhaps the most interesting failure mode, as it does not affect the faulty robot's behaviour, yet it is detected with a median TPR of 0.97. This is because when an observing robot initialises its internal simulation, although the initial position and orientation of the faulty robot may be correct, its internal state will be completely random due to corrupted packet data. This means that the simulation of expected non-faulty behaviour will be incorrect, and the predicted region of endpoints will be in the wrong place. The true endpoint of the faulty robot is therefore likely to lie outside this region, resulting in a true positive, despite the fact that the robot's behaviour is unaffected by the fault.

Controller failure

The results presented in Figure 6.3 show that this failure mode is detected with a median TPR of 0.93, as the faulty robot will behave normally most of the time, so the effect on the robot's behaviour is subtle.

Although the internal simulation will begin with the predicted behaviour being consistent with the faulty robot's true behaviour, the 'real' robot's internal state may randomly change during the time window, resulting in an endpoint outside of the predicted region. Depending on when this occurs within the time window determines whether there is enough time left for the faulty and non-faulty classes to separate sufficiently for the fault to be detected.

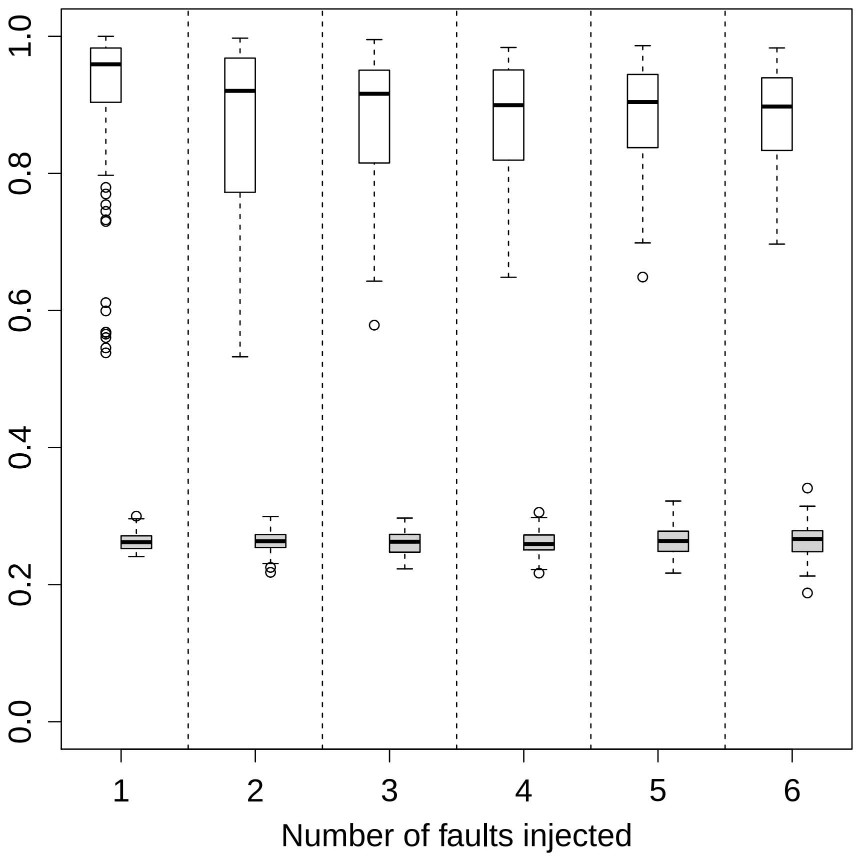


Figure 6.4: TPR (white) and FPR (grey) for increasing numbers of faults of random types.

Power failure

This failure mode is essentially equivalent to intermittent complete motor failure, so is detected with a median TPR of 0.95. The faulty robot will behave mostly as expected, except that it will suddenly stop moving occasionally. Like controller failures, the success of detecting this fault will depend on when it occurs within the time window. If it occurs right at the end of a time window, then it may be difficult to distinguish from non-faulty behaviour, resulting in a false negative.

6.4 MULTIPLE RANDOM FAULTS

Next, the fault detector's ability to cope with multiple simultaneous faults was tested. Rather than injecting faults of a specific type, the failure modes were selected at random in each of the 100 repeat runs. This provides an overall assessment of the fault detector's performance, irrespective of the failure modes present in the swarm.

Figure 6.4 shows the effect on the TPR and FPR for increasing numbers of faults. Unlike data-driven methods that use the behaviour of neighbouring robots as a model of normal behaviour, the exogenous fault detection approach proposed here is relatively unaffected by multiple faults. This is because fixed observations of neighbouring robots are replayed in the internal simulation, so the behaviour of faulty robots will simply be replayed in the same way as non-faulty robots. The results show that the fault detector is still able to achieve a high median TPR (0.9 or greater), even when over half of the swarm of 10 robots is faulty.

The outliers present in the TPR for a single fault are due to sensor failures that only represent 2/7 of the possible failure modes. When two or more faults are injected, enough sensor failures are injected that they are no longer considered outliers. The decreasing spread of the TPR and increasing spread of the FPR as more faults are injected is attributable to the change in the ratio of faulty to non-faulty robots.

6.5 SCALABILITY

All of the results presented so far have been based on a swarm size of 10 robots. To demonstrate that the results observed are not simply an artefact of a particular swarm size, a comparison of results for different swarm sizes are presented here. Only swarms of up to 25 robots were tested, due to the computational expense of running internal simulations for every observer on the same machine.

The number of faults injected is proportional to the size of the swarm, to ensure fair comparisons. For each swarm size, 1/5 of the robots were injected with faults of random types. For example, 4 faults were injected into a swarm of 20 robots. The area of the region within which the swarm is initialised was also scaled, such that the density of robots per square metre remained constant, regardless of the swarm size.

As shown in Figure 6.5, the median TPR remains high across all swarm sizes tested (0.9 or greater). The spread of the TPR decreases, as there are more observing robots performing more classifications. Again, the FPR is largely unaffected. This consistency in the results is to be expected, as local sensing and decentralised control are enforced, thus ensuring scalability.

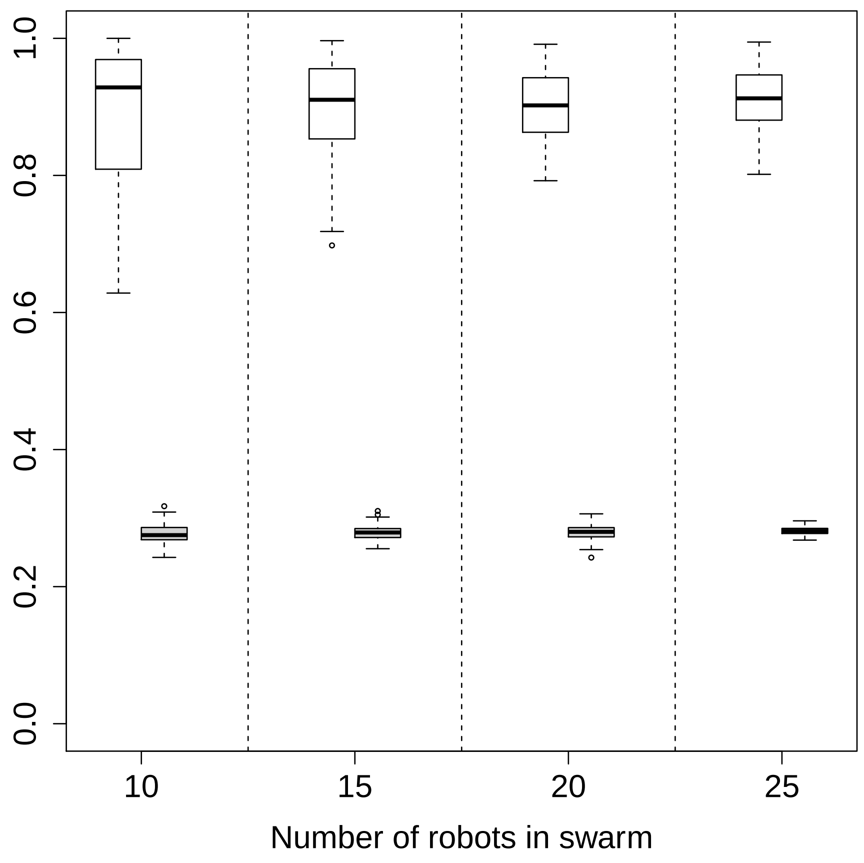


Figure 6.5: TPR (white) and FPR (grey) for swarms of increasing size.

6.6 LATENCY

Although not explicitly analysed, it is worth discussing the latency of fault detection. In all the experiments carried out to test the fault detector, faults were injected from the outset, and persisted throughout each experiment. However, if faults were to be injected during an experiment, there would be some delay before the fault is detected.

The main cause of latency is the fault detection cycle interval (see Section 5.4.7). If there is a long delay between fault detection cycles, then there may be a long delay before a fault is detected. However, even if the cycle interval is short enough that fault detection is performed at every time step, some latency will still be present. This is because once a fault occurs, a certain amount of time must pass (depending on the failure mode) before the fault is even detectable, as the faulty and non-faulty classes of behaviour must separate enough to become distinguishable. Even then, the faulty robot must have been observed at both the start and end of a time window for an observer to even consider classifying them. Beyond this initial minimum delay, the latency of the fault detector should be quite low, as a fault will potentially be detected as soon as the next fault detection cycle is complete.

6.7 GLOBAL SENSITIVITY ANALYSIS

In order to determine the influence of various parameters on the performance of the fault detection system, sensitivity analysis [102] was carried out. One-at-a-time sensitivity analysis, whereby each parameter is perturbed independently of the rest (which remain at their baseline values), is relatively naive as it ignores interactions and dependencies between parameters. Instead, *global* sensitivity analysis [103] was performed, which varies the values of every parameter under consideration simultaneously. This reveals whether a particular parameter has a strong influence on the fault detection performance, even when the values of other parameters are also changing.

The focus here is not on finding parameter values that result in optimal performance, rather to show how sensitive the proposed exogenous fault detection system is to various parameters. Robustness to parameter perturbations indicates that the system should generalise to scenarios other than the case study presented in this thesis. Sensitivity to particular parameters highlights areas where efforts should be focused in order to improve the robustness of the fault detector.

The parameters analysed via global sensitivity analysis, and the minimum and maximum values considered for each one are listed in Table 6.2. In addition to parameters of the fault detector, simulation parameters concerning the accuracy and reliability of the robots' sensors were analysed. The rest of the parameters remained fixed at their default values listed in Tables 5.1, 5.2 and 5.3.

6.7.1 *Latin hypercube sampling*

As with consistency analysis, the global sensitivity analysis presented here was carried out using the Spartan R package [100]. The approach employs Latin hypercube sampling [104], which is a technique that allows the space of possible parameter values to be sampled whilst ensuring good coverage, even with a relatively small sample size.

Figure 6.6 shows an example of Latin hypercube sampling. The range of values between the upper and lower limit for each parame-

Parameter	**Min**	**Max**
Internal simulation repeat runs	10	50
Fault detection time window (seconds)	1	10
Kernel density estimation α threshold scale factor	0	1
Position noise standard deviation (mm)	0	5
Orientation noise standard deviation (degrees)	0	5
Range and bearing packet drop probability	0	0.25

Table 6.2: Parameters analysed via global sensitivity analysis, and the minimum and maximum values considered for each.

ter is divided into N intervals. The Latin hypercube design ensures that the each of these N intervals is sampled exactly once for every parameter [105]. Although the example shown in Figure 6.6 comprises only two dimensions, Latin hypercube sampling generalises to an arbitrary number of dimensions. This technique was used to generate 100 sets of parameter values, within the specified upper and lower bounds, across the 6 parameters listed in Table 6.2.

Latin hypercube sampling produces continuous parameter values, but the number of internal simulation repeat runs and the fault detection time window length (actually specified as a number of simulation ticks) are both discrete-valued parameters. Therefore, after generating the Latin hypercube design, the sampled parameter values for these discrete parameters had to be rounded to the nearest integer. This affects the distribution of sampled values across the parameter space slightly, however the Latin hypercube design does not need to be perfect for the global sensitivity analysis to be effective.

For each of the 100 sets of parameter values generated via Latin hypercube sampling, 100 repeat runs were conducted to produce a consistent result (totalling 10,000 runs), in accordance with the consistency analysis presented in Section 6.2. In every repeat run, 3 faults of random types were injected to test the fault detector. The external simulation was executed with the same 100 random seeds for every set of parameter values, to mitigate some of the random effects of the simulation (particularly the initial positions and orientations of the robots) This gives greater confidence that observed differences in fault detection performance are actually attributable to the changes in parameter values.

The median performance of the 100 repeat runs was calculated for each sample in the Latin hypercube design, and plotted against each parameter separately to visually reveal any correlations present. The

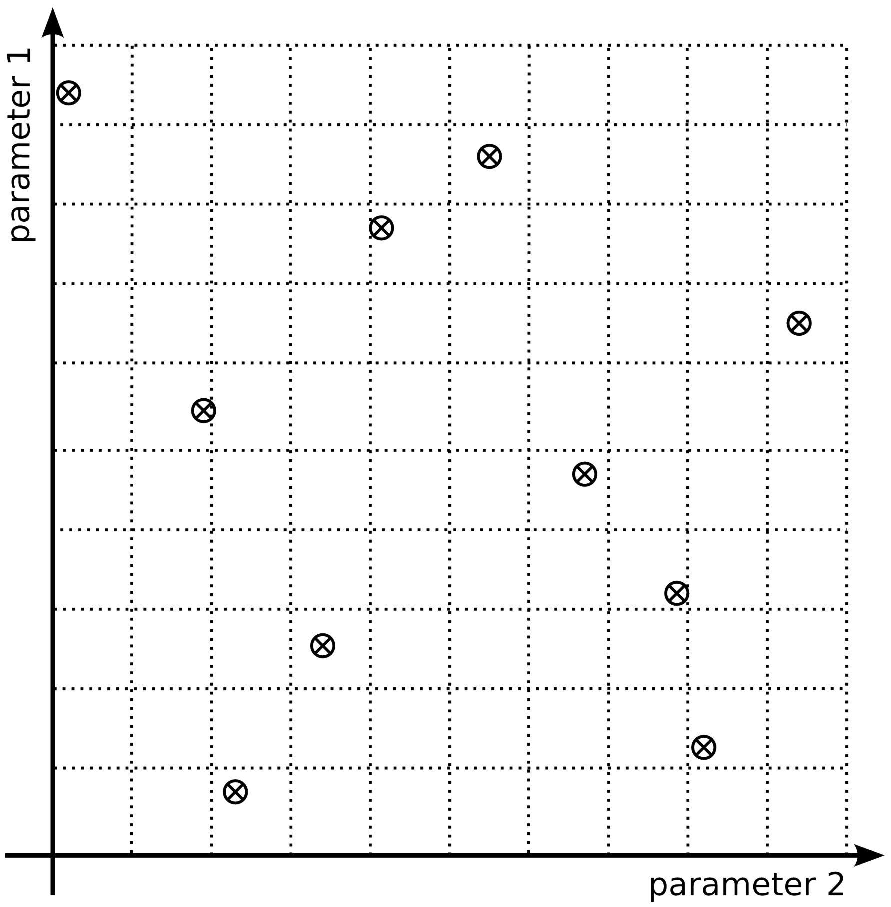


Figure 6.6: An example of Latin hypercube sampling. Ten sets of parameter values are sampled across two parameters. Taken from Read et al. [99].

Spartan [100] package also calculates the partial rank correlation coefficient (PRCC) [105], which was included in each scatter plot to aid interpretation of the results. The PRCC provides a quantitative measure of any correlation between a parameter and the fault detector's performance, after the effects of every other parameter have been statistically removed.

At first glance of the scatter plots, there may or may not appear to be a visible correlation, however they can be misleading so should be read with caution. Instead, the PRCC value should be trusted to provide a true measure of any correlation, which may not be apparent from the scatter plot alone.

6.7.2 *Sensitivity to each parameter*

The sensitivity of the fault detector with respect to each of the parameters analysed is discussed in this section. Only plots of strong

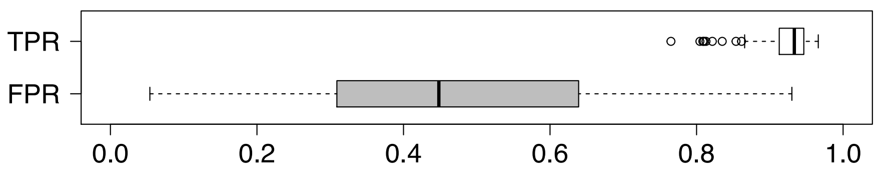


Figure 6.7: Summary of the results of global sensitivity analysis. This boxplot shows the effect of parameter variation on the TPR (white) and FPR (grey) over the 100 sets of parameter values tested.

correlations between a parameter and fault detection performance are presented here — the full results of global sensitivity analysis can be found in Appendix B.

A summary of the effect of parameter variation on the TPR and FPR over the 100 sets of parameter values tested is presented in Figure 6.7. This shows that the parameter values have a much greater influence over the FPR than the TPR. This is to be expected, as faulty robots will be correctly classified so long as the internal simulation does not predict something similar to their faulty behaviour, whereas the behaviour of non-faulty robots must be predicted very accurately to prevent the occurrence of false positives.

Figure 6.8 shows an overview of the absolute PRCC between each parameter and the TPR and FPR. The correlations between the fault detection time window length and packet drop probability on the TPR were not statistically significant. It is important to note that the PRCC is a measure of correlation only — it reveals the strength of influence of each parameter, regardless of the range of the data values. Although this figure may initially seem to indicate that some parameters have a strong influence on the TPR, when read in conjunction with the scatter plots and the overview presented in Figure 6.7, it can be seen that there is actually little overall influence on the TPR.

Internal simulation repeat runs

By default the number of internal simulation repeat runs is set to 30, which should produce a reasonable estimate of the region of non-faulty endpoints. The lower and upper limits were set to 10 and 50 repeat runs, respectively, to see whether values above or below the default would affect the fault detection performance.

It was found that the TPR of the fault detector is robust to changes in the number of repeat runs of the internal simulation (see Figure B.1 in Appendix B). There is a moderate negative correlation, which shows that the TPR actually decreases slightly when the in-

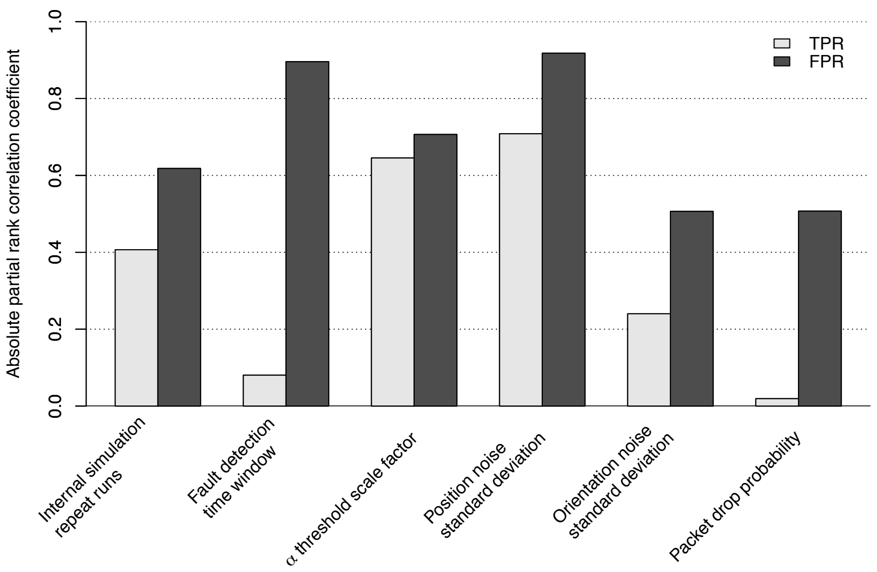


Figure 6.8: Overview of the absolute PRCC between each parameter and the TPR and FPR. The correlations between the fault detection time window length and packet drop probability on the TPR were not statistically significant.

ternal simulation samples a greater number of endpoints. This is because a greater number of internal simulation repeat runs increases the likelihood of the predicted region of non-faulty endpoints growing and engulfing the true endpoint of the faulty robot, resulting in more false negatives. There is also a slight negative correlation between the FPR and the number of internal simulation repeat runs, indicating that the FPR improves slightly with a higher number of repeat runs (see Figure B.2 in Appendix B). Again, this is due to the growth of the region of predicted non-faulty endpoints, which increases the likelihood of a non-faulty robot being classified correctly (true negative).

Clearly, there is a trade-off to be made here, as a greater number of repeat runs will lower the FPR at the cost of a lower TPR. Overall, this parameter does not have a strong influence on the fault detector's performance. Achieving good performance largely depends on the other parameters being set to favourable values.

Fault detection time window

The default fault detection time window is 5 seconds, so this parameter was analysed over the range of 1 to 10 seconds. As discussed in Section 5.4.6, the length of this time window is an important consideration. The results of global sensitivity analysis confirm that this parameter has a strong influence on fault detection performance.

As shown in Figure 6.9, the TPR suffers if the time window length is too short. This is because there is not enough time for the faulty and non-faulty classes of behaviour to separate, causing faulty robots to be incorrectly classified as non-faulty (false negatives). Longer time windows allow enough time for the classes to separate, resulting in a higher TPR. The time window length has a much stronger influence on the FPR, as can be seen in Figure 6.10. The strong negative correlation shows that the FPR rapidly decreases as the time window length increases. This is because a longer time window causes more uncertainty in the predicted endpoints, thus causing the expected region of non-faulty endpoints to grow, and increasing the chance that the focal robot's true endpoint will fall within this region.

Although the sensitivity analysis suggests that increasing the fault detection time window further might produce even better performance, this is a somewhat artificial result because there is no 'reality gap' between the external and internal simulations. If the proposed fault detection system were to be implemented on real robots, it would suffer the effects of drift due to the reality gap, as discussed in Chapter 4. Therefore, fault detection performance would deteriorate if the time window length is too long.

Kernel density estimation α threshold scale factor

By default, the α threshold scale factor is set to 1. Any value greater than 1 would reduce the size of the region of expected non-faulty endpoints, and could result in some of the sampled endpoints falling outside the region. Therefore, only values between 0 and 1 are considered, which will increase the size of the region.

The fault detection performance is not particularly sensitive to this parameter. The TPR is slightly worse for low α scale factors (see Figure B.5 in Appendix B), as this increases the size of the predicted region of non-faulty endpoints, thus increasing the likelihood of false negatives. There is a moderate positive correlation between the FPR and the scale factor, showing that the FPR generally improves as the scale factor decreases (see Figure B.6 in Appendix B). This is to be expected, because it causes an increase in the size of the predicted non-faulty endpoint region, resulting in fewer false positives.

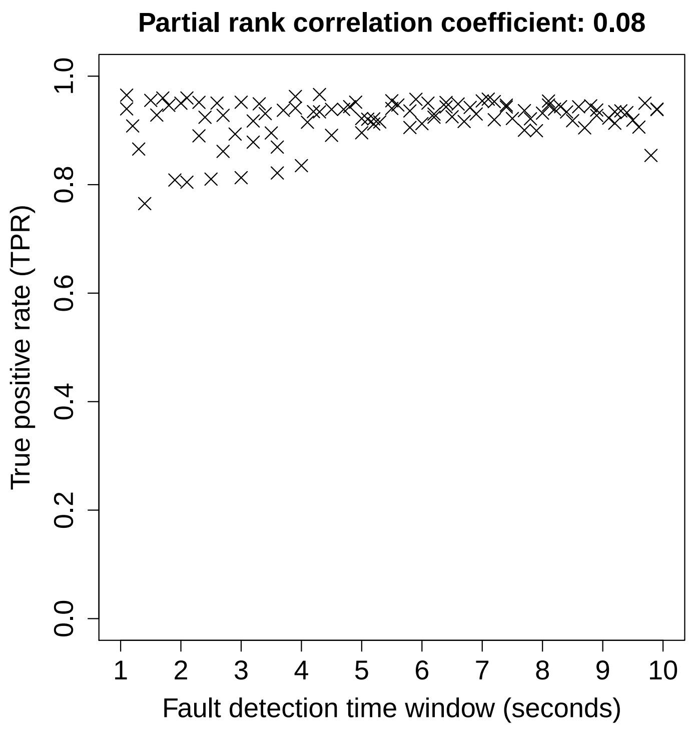


Figure 6.9: Influence of the fault detection time window length on the TPR.

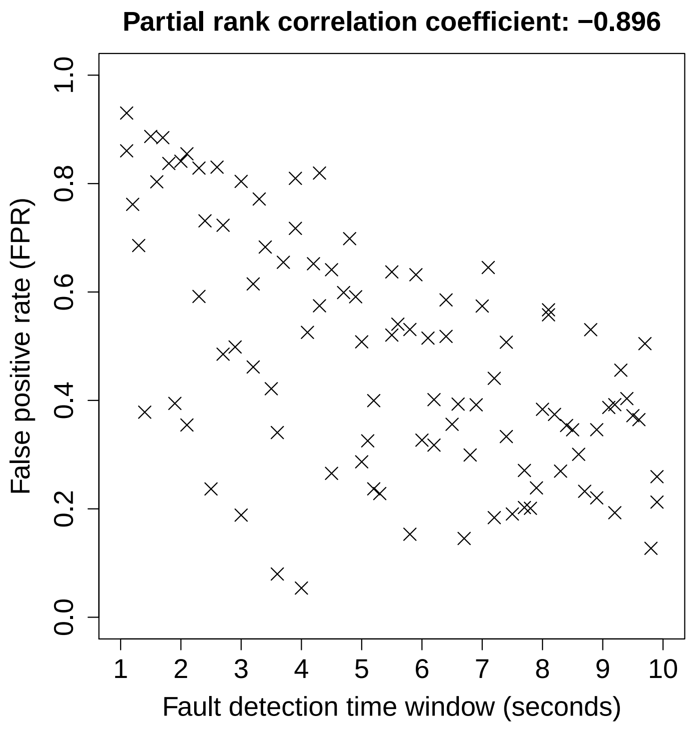


Figure 6.10: Influence of the fault detection time window length on the FPR.

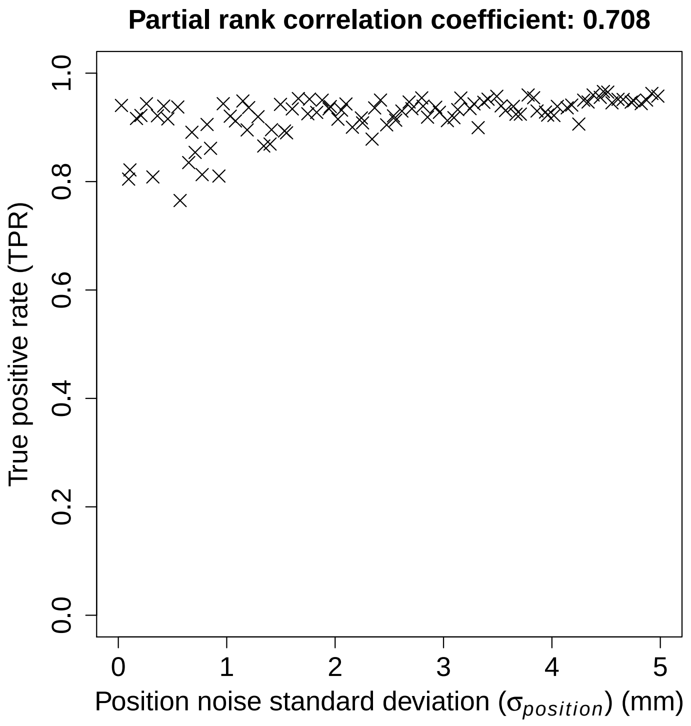


Figure 6.11: Influence of the position noise standard deviation ($\sigma_{position}$) on the TPR.

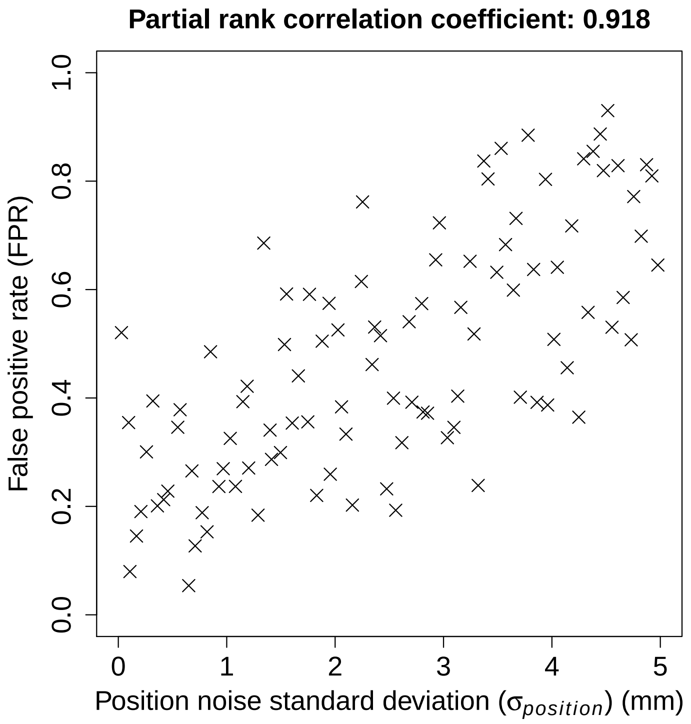


Figure 6.12: Influence of the position noise standard deviation ($\sigma_{position}$) on the FPR.

There is a trade-off to be made here, as a scale factor less than 1 can be used to improve the FPR by adding a 'buffer region' that prevents overfitting the sampled endpoints, at the cost of a lower TPR.

Position noise standard deviation

The default value used for the position noise standard deviation, $\sigma_{position}$, is 1 mm. The fault detector requires accurate sensor readings to work properly, so only standard deviations up to 5 mm were analysed. The global sensitivity analysis results reveal that this parameter has a strong influence on the performance of the fault detector.

Figure 6.11 shows that the TPR improves as the amount of noise in the sensed position increases. This is because sensor noise will not only affect the initialisation of the internal simulation, but also the replaying of non-focal robot behaviour, as every observation of a robot's position/orientation will suffer from error. Consequently, high sensor noise results in internal simulations that are inconsistent with reality, and a predicted non-faulty endpoint region that is in the wrong place. Therefore, the true endpoint of a faulty robot is even less likely to fall within this region, and be misclassified. For similar reasons, the FPR is very sensitive to position noise standard deviation, as evidenced by the strong correlation in Figure 6.12. If the predicted region of non-faulty endpoints is in the wrong place due to a noisy internal simulation, then non-faulty robots will fall outside this region and be misclassified. Even with a $\sigma_{position}$ of only 5 mm, the FPR is too high for the fault detector to be useful.

There is a trade-off to be made here, although in general, low sensor noise should be preferred, as the impact on the TPR is much lower than the FPR. It is worth noting that the effect of sensor noise would likely be even more extreme if relative coordinate systems were used, as error would accumulate over time, rather than being constrained to each independent observation.

Orientation noise standard deviation

The default value used for the orientation noise standard deviation, $\sigma_{orientation}$, is 1 degree. Standard deviations up to 5 degrees were analysed, to see whether the fault detector was sensitive to a small amount of error in the sensed orientation of a robot.

The results show that the fault detector is more tolerant to noise in the sensed orientation of a robot than the sensed position. The TPR is quite robust to changes in this parameter (see Figure B.9 in

Appendix B), and the FPR is only slightly affected — higher levels of noise result in a higher FPR (see Figure B.9 in Appendix B).

Range and bearing packet drop probability

The packet drop probability was set to 0.01 by default, which is very low. To increase realism, packet drop probabilities up to 0.25 were analysed. The results of global sensitivity analysis showed that the fault detector is quite robust to changes in this parameter (see Figure B.11 and Figure B.12 in Appendix B). As the amount of packet loss increases, the FPR increases only slightly. Despite observation data being lost due to dropped packets, there is clearly still enough information to produce a reasonably accurate internal simulation, and achieve good fault detection performance.

6.8 SUMMARY

The experimental results presented in this chapter have shown that the proposed fault detection system is able to reliably detect various different types of fault, when tested on a robot swarm performing ω-algorithm aggregation in simulation. It performs particularly well when a fault results in behaviour that is dissimilar to the expected non-faulty behaviour, such as motor failure. The fault detector does not perform as well when the change in behaviour is relatively subtle (for example, partial sensor failure).

It was also shown that the fault detection system is relatively unaffected by multiple simultaneous faults, or an increase in swarm size. Global sensitivity analysis revealed that the fault detector is quite robust to most of the parameters analysed, but is sensitive to the length of the fault detection time window and sensor noise, particularly with respect to the FPR. In terms of future improvements or optimisations, these are the parameters that are worth focusing efforts on.

Part IV

EVALUATION AND CONCLUSIONS

7 | EVALUATION AND CONCLUSIONS

This chapter concludes the thesis by summarising the main contributions of the research presented, before discussing its limitations and suggesting potential avenues of future work. Finally, the general research hypothesis defined in Chapter 1 is revisited, which the success of the thesis is evaluated against.

7.1 SUMMARY AND CONTRIBUTIONS

This section provides a summary of each chapter in this thesis, along with their main contributions.

Part I - Introduction

CHAPTER 2 - BACKGROUND & RELATED WORK:

This chapter reviewed the fault tolerance and reliability of robot swarms, and highlighted problems that can be caused by partially failed robots, motivating the need for an explicit approach to fault tolerance. Potential solutions to this problem were then reviewed in the context of natural/artificial immunity, with a focus on immune-inspired fault recovery mechanisms. Finally, endogenous and exogenous fault detection approaches were examined, in the context of swarm robotic systems.

Contribution: A review of fault tolerance and fault detection in swarm robotic systems, and a critique of existing approaches. The final section of this chapter draws together all known approaches to exogenous fault detection in the context of robot swarms, and discusses their limitations.

Part II - Fault detection via prediction of future behaviour

CHAPTER 3 - PREDICTING FUTURE BEHAVIOUR:

This chapter proposed an exogenous fault detection system that is based on the comparison of expected and observed robot behaviours, specifically with a focus on predicting future behaviour. Related work on robots with internal models was reviewed, including embedded simulations for online evolution, and predicting consequences of future behaviour for engineering safe and/or ethical robots. The experimental infrastructure required to implement the proposed fault detection approach was also discussed.

Contribution: A novel fault detection approach, which represents the first known example of predicting future behaviour via internal simulation for the purpose of exogenous fault detection in swarm robotic systems. This work brings together existing research concerning robots with internal models, and anomaly detection techniques, to produce a novel fault detection system that represents a first step towards engineering fault tolerant swarms. This chapter also includes a discussion of key issues that must be considered when attempting to implement embedded simulations on physical robot hardware, with the aim of precisely predicting future behaviour.

CHAPTER 4 - SINGLE ROBOT FAULT DETECTION:

This chapter presented the results of initial experimental work carried out to investigate the viability of fault detection based on simulated predictions of a single robot's future behaviour, as an intermediate step towards implementing the exogenous fault detection system proposed in Chapter 3 in a swarm context. The chapter also discussed open problems with fault detection based on the prediction of future behaviour, and proposed potential solutions.

Contribution: The experimental results presented in this chapter showed that simulation can be used to successfully predict real robot behaviour, however drift between simulation and reality occurs over time due to the reality gap. This necessitates periodic reinitialisation of the simulation to reduce false positives. It was shown that selecting the length of this reinitialisation time period is non-trivial, and that there exists a trade-off between minimising drift and the ability to detect the presence of faults. The open problems discussed at the end of this chapter also apply to other researchers interested in predicting the future behaviour of individual robots in a swarm.

Part III - Fault detection via analysis of past behaviour

CHAPTER 5 - ANALYSING PAST BEHAVIOUR:

This chapter presented a variation on the exogenous fault detection system originally proposed in Chapter 3, which is instead based on the analysis of past behaviour. This allows concrete observations of neighbouring robots to be collected over a past time window before attempting to discriminate between normal and abnormal behaviour, which can be used to mitigate uncertainty. This chapter also detailed the implementation of the fault detection system in simulation, and the experimental infrastructure required to test its performance.

Contribution: The revised exogenous fault detection system presented in this chapter builds on the novel contributions of Chapter 3, and is the main contribution of this thesis. This new approach reduces uncertainty by reproducing the behaviour of each robot in isolation, in the context of known past events. Although computationally more expensive, this technique allows non-faulty behaviour to be predicted with greater accuracy, thus affording more reliable fault detection. This architecture could also potentially be used to perform fault diagnosis once a fault has been detected, thus solving the first two stages of an explicit fault detection, diagnosis, and recovery process that would afford swarm robotic systems a high degree of fault tolerance.

CHAPTER 6 - FAULT DETECTION PERFORMANCE:

This chapter presented the results of experimental work carried out to assess the performance of the fault detection system proposed in Chapter 5. The fault detector's ability to detect various failure modes was tested, as well as its tolerance to multiple faults of random types. The results of scalability and global sensitivity analyses were also presented.

Contribution: The experimental results presented in this chapter showed that the revised fault detection system proposed in Chapter 5 is able to reliably detect various different types of fault, and can cope with multiple simultaneously failed robots. It was also shown that the fault detection performance scales with increasing swarm size, and that the true positive rate is robust to changes in parameter values. This work also represents the first known application of consistency analysis to swarm robotics research, which provides greater confidence in the results obtained.

7.2 LIMITATIONS

This section discusses the limitations of the work presented in this thesis, along with proposed solutions.

7.2.1 *Reliance on prior knowledge of robot controller*

The primary limitation of the exogenous fault detection approach proposed in this thesis is that the controller code of neighbouring robots being classified must be known *a priori*. This means that the model of normal behaviour is unable to change online at run-time, thus precluding its use with swarms of robots that adapt and learn during their lifetime.

Learning a model of robot behaviour online

This limitation could perhaps be overcome by learning the model of normal behaviour online via observations of neighbouring robots. If the structure of the robot controller is fixed, and available before deployment, then this problem reduces to discovering the parameters of each robot's controller.

Blum [77] presents initial work in this direction, in the context of the Consequence Engine architecture. A simple optimisation algorithm is used to estimate the parameter values of robot controllers with a known structure. Six robots perform obstacle avoidance, each with a different straight line speed, which must be learned in order to accurately predict their behaviour. Simulation is used to predict the behaviour of each robot using estimated parameter values, and the error between the predicted and observed behaviour is minimised by the optimisation algorithm.

If the structure of the robot controller is not known *a priori*, then learning a model of normal behaviour online is much more challenging. This could perhaps be achieved by training an Artificial Neural Network (ANN) to learn the input/output function encoded by a robot's controller. This would certainly be possible for simple Braitenberg vehicles, but it may be difficult to rediscover complex behaviour-based controllers.

Unfortunately, if the learning process assumes that neighbouring robots are non-faulty, then the model of normal behaviour will be susceptible to distortion by faulty robots. This may result in erroneous classifications after the model has been learnt. One potential solution might be to allow an initial bootstrapping phase where the behaviour of the robots can be guaranteed to be non-faulty, and thereafter only allow gradual changes in normal behaviour to be learned online.

Broadcasting robot controller information

Instead of attempting to learn the model of behaviour online, a simpler solution might be to have robots broadcast information about their robot controller to each other. For example, O'Dowd et al. [71] program e-puck robots to broadcast ANN controller information to their neighbours in the form of the synaptic weights. This assumes an ANN controller with a fixed topology, however the structure of the controller could also potentially be communicated.

If a robot is adapting its behaviour online, then constantly broadcasting the current version of its controller to neighbouring robots would be one way of ensuring that it is internally simulated correctly. This would allow the proposed fault detection system to be used with swarms of robots that change their behaviour after deployment.

7.2.2 *Reliance on data from potentially faulty robots*

Part of the motivation for pursuing exogenous fault detection is that it can mitigate reliance on trusting information from potentially faulty robots. However, the approach presented in this thesis is currently highly dependent on data received from neighbouring robots.

This may not be a serious issue, as the fault detection performance results for communication failure presented in Figure 6.3 demonstrate that erroneous data is unlikely to result in misclassifications — it may even help matters. If a faulty robot fails to reliably communicate the data required to internally simulate its behaviour, then it will likely be correctly classified as faulty anyway. Regardless of this result, reducing communication between robots is desirable.

Ideally, exogenous fault detection would be based solely on the outwardly observable behaviour of neighbouring robots. This section discusses what could potentially be achieved using the observer's onboard sensor hardware alone.

Position and orientation

The revised fault detection system proposed in Chapter 5 uses a simulated model of a range and bearing sensor to observe the position of neighbouring robots. This is an *active* IR sensor, in the sense that the position of neighbouring robots can only be calculated if they are actively emitting IR light. Unfortunately, this means that completely failed robots become invisible, and therefore cannot be classified.

One potential alternative might be to fit each robot with a *passive* sensor, such as an omnidirectional camera, which does not rely on actively emitted signals to detect neighbouring robots. The main disadvantage with the use of a camera, is that uniquely identifying each observed robot is much more difficult — in the proposed approach robots transmit their unique IDs via the range and bearing sensor/actuator. However, Olson et al. [106] showed that it was possible for a team of 14 autonomous robots to uniquely identify each other using AprilTag fiducial markers [107] and on-board cameras.

Neither range and bearing sensors, nor cameras can directly sense the orientation of a particular robot (unless sophisticated image processing were to be used). The approach presented in this thesis instead assumes that the robots are able to proprioceptively sense their current orientation using an on-board compass, and broadcast this information to neighbouring robots. In order to remove this dependency of communicated orientation data, an observer could infer the orientation of neighbouring robots by calculating their direction of travel from consecutive observations.

Internal controller state

As mentioned in Section 5.2.3, the internal state of a robot's controller must be known in order to initialise the internal simulator. In the approach proposed in Chapter 5, each robot simply broadcasts this data to neighbouring robots.

To remove this dependency on communicated internal state data, an observing robot could potentially infer the internal state of a neighbouring robot based on observations of its behaviour. This could be achieved by using the internal simulator to execute multiple *'what if?'* scenarios, to test several estimated internal states from the same initial conditions. The error between the predicted and observed behaviour could then be used as feedback to an optimisation algorithm, which would allow the true internal state of a robot to be determined.

Secondary observation data

Removing the dependency on broadcast secondary observation data would be much more difficult. As explained in Section 5.1, each robot broadcasts its observations of neighbouring robots, which allows an observer to internally simulate every individual that may have influenced the behaviour of that robot during the past time window.

Without this information, the observer would be strictly limited to its own local observations, which would often be insufficient to accurately reproduce the behaviour of neighbouring robots within its internal simulation. This would therefore result in more frequent misclassification of robot behaviour — particularly an increase in the number of false positives due to unexplained deviations from expected behaviour. Nevertheless, exogenous fault detection would still be possible, albeit at a degraded level of performance.

7.2.3 *Sensitivity to sensor noise*

As shown in Section 6.7, the FPR of the fault detection system is highly dependent on accurate observations of robot behaviour, so is very sensitive to even low levels of noise in both the sensed position and orientation of neighbouring robots. If the focal robot is initialised in the internal simulation with an incorrect pose, then the predictions of its behaviour will be incorrect. Therefore, noisy observation data will significantly affect the false positive rate of the classifier.

Consequently, either high accuracy sensor hardware must be used, or some method of mitigating the effects of noise is required. One possible solution might be for each observing robot to cross-reference the secondary observation data broadcast by its neighbours, to find duplicate observations of each neighbouring robot. For any particular robot, duplicate observations of its position and orientation could be averaged, to provide a better estimate of its true location.

7.2.4 *Problems with detecting certain failure modes*

As mentioned in Section 7.2.2, the use of active range and bearing sensors precludes the detection completely failed individuals. This is not a major concern, as it has been shown by Winfield and Nembrini [6] and Bjerknes [7] that the effect of completely failed robots on swarm behaviour is relatively benign. That being said, unless the failed robots are modelled as environmental obstacles within the internal simulation, then the observed behaviour of neighbouring robots will not be fully accounted for.

Robots that suffer a failure in their range and bearing actuators, such that they stop emitting IR light, will similarly become undetectable. In this case, the partially failed robot may have some negative influence on swarm behaviour, while remaining invisible to neighbouring robots. As discussed in Section 7.2.2, this problem could potentially be solved through the use of passive sensors, such as cameras, so that the position/orientation of robots suffering these failure modes can still be detected.

Like the work of Tarapore et al. [12], the exogenous fault detection system presented in this thesis also struggles to detect faults which result in behaviour that is similar to non-faulty behaviour. Therefore, the detectable failure modes will very much depend on the definition of normal behaviour.

The results presented in Chapter 6 are specific to ω-algorithm aggregation, for which it is easy to detect motor failures, because the default behaviour of each robot is to be constantly moving. However, if the expected non-faulty behaviour were for a robot stop moving for extended periods of time (to manipulate objects in the environment, for example), it would be more difficult to detect motor failures — at least until the robot is expected to move again.

The results in Chapter 6 also show that sensor failures (both complete and partial) are more difficult to detect than other failure modes, due to their subtle affect on robot behaviour. Sensor failures could perhaps be detected more easily by simply checking for inconsistencies between secondary and direct observation data. For example, if robot A does not report observations of robot B, but robot C can see both robots A and B, and can tell that robot A should be able to see robot B, then it may infer that robot A has faulty sensors. However, this would only work if there was some overlap in the observation data. To increase the chance of overlap, data could be pooled from neighbouring robots, and cross-referenced for inconsistencies.

7.2.5 *Computational expense*

As discussed in Section 5.1, the revised exogenous fault detection approach trades-off mitigation of uncertainty for computational expense. This is because the behaviour of each neighbouring robot must be predicted in isolation, thus increasing the number of internal simulation executions per fault detection cycle.

There is a theoretical upper bound on the maximum computational expense incurred by neighbouring robots, based on how many could physically fit within an observer's shared sensor range, and only those directly observable would be classified. Beyond this limit, an observer's internal simulation will not consume further computational resources, regardless of the number of robots in the swarm. The average computational expense will be significantly lower in practice, provided that the swarm is not very densely aggregated.

If the hardware resources available are unable to cope with the computational demand of the proposed exogenous fault detection approach, then a low fidelity simulation, such as the minimal e-puck simulator developed by O'Dowd [70] (used in Chapter 4), could be used instead of a general purpose robot simulator like ARGoS. In the event that even a low fidelity simulation is too demanding, then various optimisations could be made to reduce computational expense.

Given that most of the robots in a swarm robotic system will be non-faulty for the vast majority of the time, constantly running internal simulations to verify normal behaviour is probably unnecessary. Instead, it may be more appropriate to use a lighter-weight *anomaly* detection system in the first instance, to detect whether anomalous behaviour has occurred in the observer's immediate neighbourhood, without necessarily indicating which robot is specifically at fault. This could then trigger execution of the exogenous fault detection system proposed in this thesis, to perform fine-grained fault detection.

Each observing robot does not necessarily even need to classify every single one of its neighbours at each fault detection cycle. Instead, it could classify a random sample of its neighbours each time, to reduce computational load. The disadvantage of this approach, is that faulty robots may go unnoticed until they are randomly selected for classification, thus increasing the latency of fault detection. Similarly, the fault detection cycle interval could also be increased, to achieve a similar trade-off of computational expense and latency.

7.2.6 *Generalisability*

It is important to note that the experimental results presented in Chapter 6 are limited to simulations of ω-algorithm aggregation. The fault detection performance is expected to differ when tested on other swarm behaviours, due to the fact that the performance depends on how easily faulty behaviour can be distinguished from non-faulty behaviour. For example, it would be difficult to detect motor failures in swarm behaviours where the individual robots do not move very often. This is because the predicted non-faulty behaviour may be for a robot to remain stationary, which is indistinguishable from faulty behaviour due to motor failure, resulting in false negatives until the robot attempts to move again.

Furthermore, poorer performance would be expected when applied to probabilistic robot behaviour such as a random walk. This is because non-deterministic behaviours are more difficult to predict, and the increased uncertainty will result in a larger region of predicted non-faulty endpoints. This, in turn, will cause an increase in the number of false negatives, and consequently a lower TPR.

However, the importance of reliably detecting faults will also vary depending on the swarm behaviour, as each will be affected by faulty robots in different ways. Collective behaviours that critically depend on self-organisation to achieve locomotion, such as β-algorithm and ω-algorithm phototaxis, are highly susceptible to motor failures in individual robots (as discussed in Section 2.1). Whereas, swarm behaviours that do not rely on co-operation, such as foraging algorithms where individual robots act independently, are unlikely to be significantly affected by motor failure in a single robot.

Without further testing, it is difficult to speculate about the generalisability of the results to other swarm behaviours. However, at least for deterministic behaviours, the fault detector should perform well on other swarm algorithms, provided that the faults to be detected result in significant deviations from the expected non-faulty behaviour.

7.3 FUTURE WORK

This section suggests potential avenues of future work.

7.3.1 *Improving fault detection performance*

The research presented in this thesis has focused on demonstrating a proof of principle, rather than tuning the fault detector to produce optimal performance. The experimental results presented in Chapter 6 could potentially be improved in a number of ways, as will be discussed throughout this section.

Trajectory analysis

The implemented approach only compares the endpoints of the predicted and observed behaviour, completely ignoring the trajectories of the robots. Performance could perhaps be improved by taking this trajectory data into consideration. This is especially important if the fault detection time window is long enough for a robot to double back on itself, such that it ends up very close to where it started. This kind of scenario is difficult to distinguish from a lack of movement due to motor failure, based on endpoint data alone.

Without even performing sophisticated trajectory analysis, simply comparing the expected and observed distance travelled by a robot over the time window would allow the number of false negatives to be reduced. Performance improvements could also be gained by comparing the predicted and observed orientations of a robot during the fault detection time window, as this provides further information that can be used to discriminate between borderline cases. It may even be sufficient to compare only the final orientations.

Discontinuous observation data

As mentioned in Section 5.4.3, in the event of discontinuous observations, no interpolation between available data points is performed. Instead, non-focal robots simply disappear from the internal simulation when they cannot be observed, and reappear if observed again within the fault detection time window. This makes it difficult to reproduce the focal robot's behaviour accurately, because predictions must be performed using incomplete information.

The main cause of discontinuous observation data is packet loss, which has been shown to cause an increase in the false positive rate of the classifier (see Section 6.7). It is therefore desirable to mitigate the effects of this discontinuous data in some way.

One solution might be to use interpolation to fill in the gaps in the data, by estimating the missing positions/orientations. This would prevent non-focal robots from disappearing and reappearing in the internal simulation, which should result in more reliable prediction of behaviour. Simple linear interpolation may be sufficient, however the copy of the robot controller may even prove useful for filling in the blanks, as it should be able to predict the missing behaviour.

An alternative approach might be to collate secondary observation data from neighbouring robots, in the hope that at least one of the robots will have made the missing observations themselves.

Lowering the false positive rate

The fault detection performance results presented in Chapter 6 show that the default FPR is relatively high — around 25%. As discussed in Section 2.3, the misclassification of robot behaviour can be costly. For example, if a non-faulty robot is erroneously classified as faulty this may result in the instigation of unnecessary collective recovery mechanisms, thus wasting time and energy. Conversely, partially failed robots left undetected may have a detrimental effect on swarm behaviour. Therefore, parameter values must be selected such that these two cases are traded-off based on their relative importance.

For situations where a low FPR is more important, the number of false positives can be reduced in a few different ways, at the expense of a lower TPR. Firstly, the fault detection performance presented in this thesis is based on the raw output of the classifier — each fault detection cycle is completely independent of those that came before it, so the classifier essentially has no memory. As demonstrated by Christensen et al. [61], thresholding a moving average of the output of a fault detector can reduce the number of false positives, by filtering out brief anomalies in the data, so that it will only detect persistently faulty behaviour. This technique was applied successfully in Section 4.4, in the context of predicting future behaviour, and can also be applied to the analysis of past behaviour.

Figure 7.1 shows the effect of different filtering window sizes on smoothing out false positives. For a window size of N, an observer's classifications of a particular robot from the past N fault detection

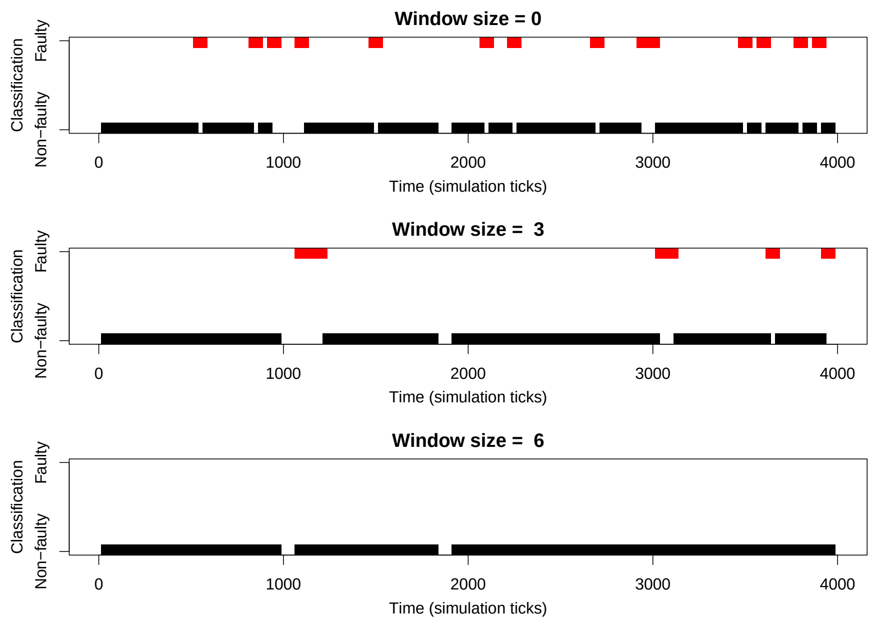


Figure 7.1: The effect of different filter window sizes on smoothing out false positives. Classifications over time from a single observer that was observing a non-faulty robot. Any faulty classifications are therefore false positives.

cycles are considered, and the majority classification is taken as the filtered result. A window size of zero has no effect, so corresponds to the raw classification data. The majority of the classifications are correct (true negatives), but there are brief periods where the non-faulty robot is classified as faulty (false positives). In this example, many of the false positives can be filtered out with a window size of three, and completely removed with a window size of six. However, this reduction in the number of false positives comes at the cost of increased false negatives and fault detection latency [12].

Instead of simply filtering out false positives, the root cause of the misclassifications may be addressed. Figures C.1 and C.2 in Appendix C show that false positives are often the result of borderline cases, which can be avoided by lowering the α threshold scale factor to increase the size of the predicted non-faulty endpoint region (at the expense of the TPR). The effect of this parameter on the FPR is shown in Figure B.6 of Appendix B.

Additionally, faulty classifications could be confirmed via targeted monitoring. For example, if an observing robot suspects that a neighbouring robot is faulty, then it could ask that robot to run a series of diagnostic tests to determine whether or not it is actually faulty. This would reduce the likelihood of expensive recovery mechanisms being triggered prematurely due to false positives.

Parameter value optimisation

As demonstrated in Section 6.7, the parameters of the fault detection system (listed in Table 5.3) dictate the level of performance attained. The optimal values for these parameters will depend on many factors, such as the swarm behaviour, the failure modes to be detected, the reality gap, the amount of sensor noise, the computational resources available, and the desired ratio of TPR to FPR.

The application of Multi-Objective Optimisation [108] would therefore seem appropriate, as the selected parameter values must trade-off multiple conflicting objectives. The results of global sensitivity analysis presented Chapter 6 may also inform parameter optimisation, as they indicate which parameters have the most influence on fault detection performance, and which will therefore benefit from optimisation more than others.

Given enough computational power, an ensemble of fault detectors could perhaps be used, each tuned to detect different types of fault under varying conditions. The results of each could even be combined via some form of majority voting mechanism to improve overall fault detection performance.

7.3.2 *Modelling the environment*

As mentioned in Section 5.2, the experimental work presented in Part III of this thesis was carried out in an empty simulated arena, free from obstacles. This allowed the problem of internally modelling the environment to be sidestepped for this proof of principle. If the proposed fault detection approach is to be useful in a realistic application scenario, it will be necessary for the robots to internally model their environment, so that they are able to accurately predict the true behaviour of their neighbours in response to obstacles.

This is a particularly challenging problem, and the accuracy of the environmental modelling largely depends upon the robots' sensing capabilities. Well-established techniques such as Simultaneous Localisation and Mapping (SLAM) [109] could potentially be used build an internal model of the robot's local environment, within which neighbouring robots are simulated. Dynamic environments should not present an issue, so long as the dynamic elements can be recorded by each robot over the course of each fault detection time window. Again, the fidelity of the modelling is an important consideration, as it must trade off accuracy of behavioural prediction with computational expense.

It may even be possible to achieve satisfactory fault detection performance without internally modelling every aspect of the environment. Taking phototaxis in the ω-algorithm as an example (described in Section 2.1.1), given that individual robots cannot sense the direction or angle to the beacon, it follows that they will not be able to internally simulate its presence or predict its effect on robot behaviour. Given that the exogenous fault detection approach proposed in Chapter 5 is based on the analysis of past behaviour, fixed observations of robots are available, so any influence the beacon had on their behaviour over the past time window is already known.

Only the beacon's influence on the focal robot's predicted behaviour will be unaccounted for, which will be relatively minor as it only controls the avoidance radius of the robot. Fault detection despite incomplete internal models may therefore be entirely possible. Given sufficient computational resources the internal simulator could perhaps also be used the run many *'what if?'* scenarios in order to infer the location of environmental features that cannot be locally observed.

7.3.3 *Implementation on physical hardware*

Although the experimental work presented in Chapter 4 was carried out on a physical e-puck robot, the revised exogenous fault detection system proposed in Chapter 5 has not yet been implemented and tested on real robots. This is an important next step, in order to validate the results observed in simulation.

Experimental infrastructure required

Instead of immediately attempting to implement a fully-embedded solution (as envisaged in Chapter 3), the use of an experimental infrastructure like that used by Winfield et al. [75] to implement the Consequence Engine architecture [76] on real robots would initially more feasible. This would enhance both the sensing capabilities and computational resources of the e-puck robots, beyond that which can be achieved with the current hardware platform.

The tracking infrastructure presented in Appendix A could be used to implement a range-limited virtual sensor that allows the e-puck robots to observe the position and orientation of their neighbours, which would otherwise only be possible with a range and bearing extension board [87] or an omnidirectional camera [29].

Also, as mentioned in Chapter 3, Winfield et al. [75] implement each robot's internal model using the Stage robot simulator, which runs on a separate server in a service-oriented architecture, accepting requests from robots over a network connection. The revised exogenous fault detection system presented in Chapter 5 could be implemented in the same way, using instances of ARGoS for each robot's internal simulation.

Conceptually, the exogenous fault detection system would run onboard each robot, despite the use of remote processing and virtual sensing. This would be permitted under Sharkey's definition of Scalable Swarm Robotics [26], as local sensing and decentralised control would still be enforced.

The reality gap

As discussed in Section 3.3.2, if the proposed exogenous fault detection system were to be implemented on physical robots, there would be a reality gap between the internal simulation and the real world. It was shown in Chapter 4 that this reality gap causes drift, thus necessitating periodic reinitialisation of the simulator. The experimental work presented in Part III of this thesis does not suffer from this problem, due to the use of nested identical instances of ARGoS.

Although the results presented in Chapter 6 seem to indicate that a longer fault detection time window will result in improved performance, this does not take into account the effects of drift due to the reality gap. Once implemented on real robots, this drift will mean that performance gains obtained by increasing the time window length will eventually fall off, as shown in Section 4.4.1.

The effects of drift could be mitigated with better calibration of the internal simulation. If the three categories of correspondence between simulation and reality enumerated by O'Dowd et al. [72] can be improved, then this will close the reality gap further, thus allowing more accurate predictions of robot behaviour. However, there exists a trade-off between simulation fidelity and computational expense.

Rather than increasing fidelity, automated methods could be used to optimise parameters of the simulator, to reduce the error between predicted and observed behaviour. As mentioned in Appendix A, the ground-truth data required for these comparisons could be obtained via a tracking infrastructure.

Fully-embedded solution

Once the exogenous fault detection system has been demonstrated to work on physical hardware using remote processing and virtual sensing, the implementation of a fully-embedded solution could then be attempted. This may be quite challenging due to the computational expense of predicting the behaviour of each robot in isolation.

In order for the proposed fault detection approach to be useful in practice, each fault detection cycle must complete in real-time before the next cycle is executed. If this presents problems, then the fault detection cycle interval can be tuned to accommodate for longer processing times, at the expense of fault detection latency.

As mentioned in Section 5.2, the aim of this research was to develop a fault detection system that would adhere to the Scalable Swarm Robotics principles of decentralised control and local sensing [26], without being constrained to a particular hardware platform. Therefore, the target platform chosen for a fully-embedded solution may not be an e-puck robot, especially given the age of the hardware.

General-purpose computing on graphics processing units (GPGPU)

The fault detection method presented in this thesis is 'embarrassingly parallel', in the sense that each repeat run of the internal simulation is entirely independent and could therefore be executed concurrently. The minimal e-puck simulator developed by O'Dowd [70] was recently rewritten by Poulding [110] using the CUDA parallel programming architecture [111], for the purpose of GPU-accelerated testing of robot controllers. This allows whole instances of the simulator to be executed in parallel, so could be used to analyse the past behaviour of every neighbouring robot simultaneously.

If a higher-fidelity simulation is required, an alternative approach would be to accelerate some aspect of the simulator using GPGPU. Jones et al. [112] investigated the use of GPGPU on autonomous mobile robots, for the acceleration of parallel *'what if?'* scenarios, specifically motivated by the ethical robot work of Winfield [76]. They show that the time consuming ray tracing operation of the Stage robot simulator can be GPU-accelerated using OpenCL [113], and executed on System-on-Chip devices that incorporate powerful GPUs with low power consumption. Jones et al. [112] tested the concept on laptops, but state that they intend to equip e-puck robots with mobile GPU hardware in future, to enable accelerated embodied simulation for the Consequence Engine architecture.

7.3.4 *Acting upon classifier output*

As mentioned in Chapter 5, the exogenous fault detection method implemented for this thesis is entirely passive — the robots do act upon their classifications of neighbouring robots. An appropriate method of acting upon this information must be determined, and whether it should be shared within the swarm to make collective decisions.

Collective consensus

Collective consensus could be achieved either *implicitly* or *explicitly*. In the implicit case, each robot would independently decide what to do in response to its own faulty classifications of neighbouring robots. Again, taking ω-algorithm phototaxis as an example, if each observing robot chooses to ignore neighbouring robots that it has determined to be faulty, this would result in partially failed robots being left behind. This would represent a form of emergent collective consensus, as it does not require explicit communication of classification results within the swarm.

Alternatively, an explicit collective consensus could be achieved via some form of majority voting mechanism amongst robots in the local neighbourhood. This could potentially help to filter out false positives, because each classification would be performed from a different perspective. However, it remains to be seen whether cross-referencing classification data with that of neighbouring robots would improve the reliability of classifications. If multiple robots independently arrive at the same incorrect answer, then this may be reinforced by a majority vote — the effect of which could be costly. This issue would be exacerbated by faulty robots broadcasting erroneous classification data — depending on the failure mode, faults in an observing robot may prevent it from classifying neighbouring robots correctly.

Nested and recursive internal simulations

A significant open problem with the proposed exogenous fault detection system, is that of nested and recursive internal simulations. For example, robot A internally simulates robot B, who is internally simulating robot C. Robot B determines that robot C is faulty, and initiates a fault recovery behaviour. The behaviour of robot B will not be anticipated by the internal simulation of robot A, unless it also internally simulates robot B internally simulating robot C. Therefore, robot A will classify robot B as faulty.

This issue of nested internal simulations becomes recursive when robot *A* internally simulates robot *B*, who is internally simulating robot *A*. Without accounting for the internal simulations of other robots, any fault recovery behaviour will be unexpected, and therefore every robot will end up classifying each other as faulty.

One potential solution to this problem might be to program the robots to explicitly signal when a fault recovery action is going to be taken, so that fault detection based on internal simulation can be temporarily disabled, before returning to the task at hand. However, this will not allow faults that occur during fault recovery to be detected.

An alternative approach might be to allow a certain depth of nested internal simulations, as sufficient prediction accuracy may be achieved with only shallow nesting. Unfortunately, this would again increase the computational complexity of the solution. This remains an open problem, as the best solution is not immediately clear.

Fault diagnosis and recovery

The output of the classifier should feed into fault diagnosis and recovery mechanisms, in order to complete the explicit process of fault tolerance. The architecture based on internal models presented in this thesis could potentially be used to implement both fault diagnosis, and to predict the outcome of various fault recovery mechanisms to assess their cost/viability.

In terms of fault diagnosis, the internal simulator could be used to simulate the injection of various different types of fault into the focal robot, to predict their effect on its behaviour from the same initial conditions. This would result in a separate set of predicted endpoints for each failure mode considered. Rather than semi-supervised anomaly detection used for fault detection, the observer is now faced with a multi-class classification problem, where each fault has its own class.

Each of these classes of behaviour could still be modelled using KDE, but rather than using a threshold to define a bounded region of normal behaviour, non-linear decision boundaries would implicitly be defined where the probability density of each estimated distribution overlaps. The region of this pattern space that the focal robot's true endpoint falls into would determine which fault is diagnosed.

Following fault diagnosis, the internal simulation architecture may be used for the purpose of determining the most appropriate fault recover mechanism to use. However, it would would not be possible to do this via the analysis of past behaviour — instead the prediction of future behaviour would be required, as the outcome must be *imagined*. The preceding fault diagnosis should have determined the cause of the failure, and therefore whether the fault is likely to be repairable.

Like the Consequence Engine architecture [76], the internal simulation could be used to predict the consequences of various actions, so that the cost of repairing a particular fault is worth the time/energy investment can be assessed. The success of various different options could also be weighed against each other, allowing the most appropriate course of future action to be taken.

7.4 CONCLUSION

Chapter 1 defined the following general research hypothesis that guided the research presented in this thesis:

> *Individual robots in a swarm robotic system can use internal simulations to predict the behaviour of their neighbours, and through the comparison of expected and observed behaviour, can exogenously detect the presence of faults in those robots.*

It was shown that this behavioural prediction can be achieved in two different ways — via the prediction of future behaviour in Part II, and via the analysis of past behaviour in Part III of this thesis. For both approaches, it was demonstrated that faults can be detected via the comparison of expected and observed robot behaviours. Although the work in Part II was only tested on a single robot, the work in Part III was demonstrated to allow exogenous fault detection in a swarm robotics context, albeit only in simulation.

The initial aims of this thesis have been met, and the research presented has demonstrated a valuable new approach to exogenous fault detection based on the use of internal models. This is in line with a recently emerging trend towards internal modelling, which is believed to be required for the field of swarm robotics to make a leap forward.

The use of an internal simulation may seem heavyweight in comparison to existing data-driven approaches, however the same architecture could also be used by a robot throughout both fault diagnosis and recovery. The research presented in this thesis therefore represents a significant step towards a novel integrated solution for explicit fault tolerance in swarm robotic systems.

Part V

APPENDIX

A | TRACKING INFRASTRUCTURE

This appendix presents a low-cost tracking infrastructure that was built for the purpose of implementing the exogenous fault detection system proposed in Chapter 3. Considerations that were taken when building and configuring the system are highlighted, using a swarm of e-puck robots as a case study. Potential applications of the system are discussed, the cost of the equipment is documented, and its known limitations are addressed.

A.1 OPTICAL TRACKING SYSTEMS

Optical tracking systems are used in many research laboratories for monitoring and recording the movements of mobile robots. The data gathered by these systems is invaluable for offline post-experiment analysis, such as measuring the area coverage of a robot swarm. These systems can also be used to provide robots with online feedback about their current position and orientation, for the purpose of indoor localisation. Unfortunately, obtaining precise and reliable tracking data often comes at the cost of expensive equipment.

Commercial motion capture systems typically work by attaching retroreflective markers to the robots, and using special cameras that illuminate the scene with IR light. These cameras observe the light reflected from the markers through an IR filter, allowing them to efficiently locate the markers in the scene using simple image processing algorithms. A pattern of markers is recognised by the tracking system, allowing the position and orientation (or *pose*) of a robot to be detected. Vicon motion capture systems take such an approach, and are able to track robots precisely and reliably. The Vicon-based experimental infrastructure built at the Bristol Robotics Laboratory (BRL) [74] is of particular relevance, as it has been used successfully in a number of swarm robotics studies [72, 114, 8, 75]. Unfortunately, for many research laboratories, Vicon systems comprising even a small number of cameras are prohibitively expensive.

In contrast, IRIDIA's Arena Tracking System [115] does not use retroreflective markers to detect robots. Instead, an overhead array of 16 visible-light cameras is used to decode printed markers placed on top of each robot, in order to determine their pose. Although this system is cheaper to build than a Vicon system, the high-resolution cameras used are still expensive, and many man-hours would be required to develop the custom software necessary to drive the system. Alternatively, an open-source vision-based tracking software solution could be used, such as SwisTrack [116] or AprilTag [107], which were developed as cost-effective alternatives to expensive motion capture systems like Vicon. However, the precision of the tracking data obtained using visible-light cameras is often inferior to that acquired from commercial motion capture systems that use retroreflective markers.

OptiTrack[1] is a recent competitor in the motion capture market, and offers a cost-effective alternative to Vicon systems, whilst still delivering precise and reliable tracking data. This potentially provides an affordable solution for many research laboratories. An OptiTrack system for real-time tracking of ground-based robot swarms was built for this research at the York Robotics Laboratory (YRL), which is described in the following sections.

A.2 SYSTEM OVERVIEW

Figure A.1 shows the tracking infrastructure built at YRL. This is similar to the Vicon-based experimental infrastructure constructed at BRL [74]. The OptiTrack motion capture system provides high precision tracking of the robots. This consists of three cameras that are mounted on a 3.5 m $\times$ 3.5 m $\times$ 2.5 m truss rig, and are angled down to face the arena below. A swarm of robots can be tracked anywhere within the boundaries of the 2.5 m square arena. OptiTrack offer a wide range of different cameras, with varying levels of performance. Flex 13 cameras were chosen, as they are designed to provide medium volume motion capture at a reasonable cost. These cameras have a resolution of 1.3 megapixels, a maximum frame rate of 120 frames per second, and a horizontal field of view of 56°. In order to track the robots, the cameras have a ring of 28 IR LEDs that emit light with a wavelength of 850 nm. This light bounces off retroreflective markers mounted on the robots, and returns to the cameras. IR filters and image processing algorithms on-board the cameras are used to detect these markers.

1 `https://www.optitrack.com`

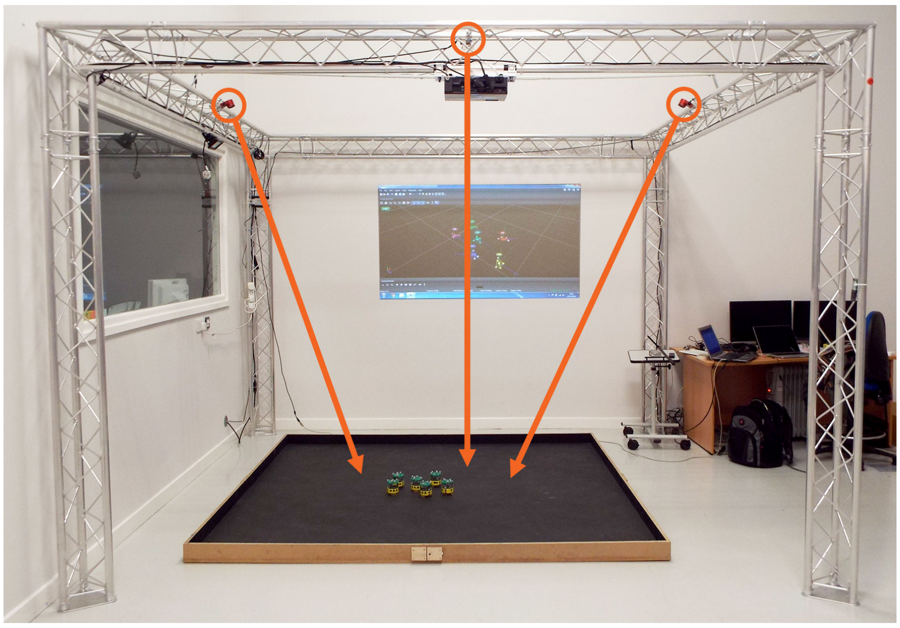


Figure A.1: Tracking infrastructure monitoring a swarm of six e-puck robots. The three OptiTrack cameras are circled. The arrows indicate the angle of each camera.

At least two cameras are required for the tracking software to triangulate marker coordinates, but a greater number of cameras is typically used to improve coverage. This is especially important if a swarm needs to be tracked over a large area. Using more cameras also allows robots to be tracked reliably even in the presence of marker occlusion — where line of sight between a camera and a marker is obstructed. However, when tracking ground-based robots on a 2D plane, the likelihood of marker occlusion is relatively low. For this research, three cameras arranged in a triangular layout was found to provide adequate coverage at a low cost. The positioning of the cameras is important — they are set up such that their fields of view overlap with each other, and are placed overhead to reduce the chance of marker occlusion. The cameras are mounted on a permanent truss rig so that the cameras do not move, removing the need for repeated recalibration of the system.

The e-puck robots move at a relatively low maximum speed of 13 cm/s, which is easy for the tracking system to handle. Figure A.2 provides an overview of the experimental infrastructure, and illustrates data flow between system components. The three Flex 13 cameras are connected via USB to an OptiHub 2 — a custom USB hub that allows up to six cameras to synchronise with each other. This, in turn, connects via USB to the OptiTrack server (a Windows machine with

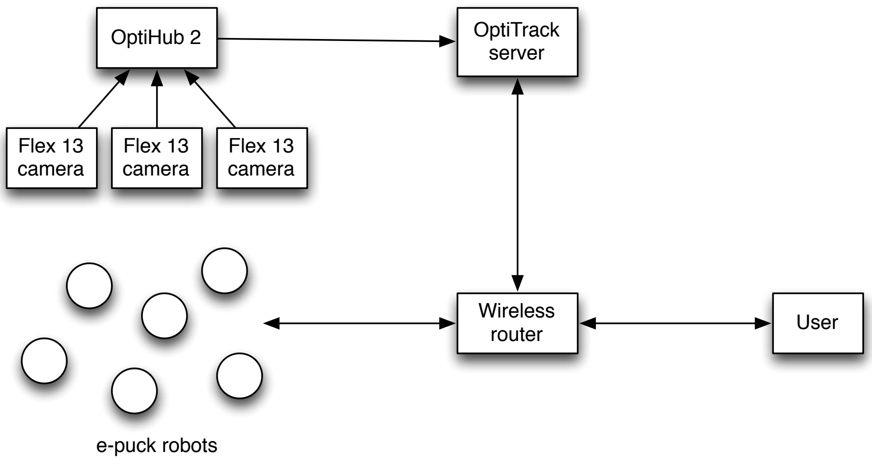


Figure A.2: Overview of the tracking infrastructure, showing data flow between system components.

tracking software installed), allowing tracking data to be obtained from the cameras. The server processes the tracking data, logs it for post-experiment analysis, and makes it available to e-pucks and users in real-time via the wireless LAN. Each robot is assigned a static IP address, and connects to the LAN via a wireless router. This allows networked computers to connect to any robot using the SSH protocol.

A.3 TRACKING HATS

Retroreflective markers must be attached to the robots so that they can be tracked by the OptiTrack cameras. Liu and Winfield [74] solved this problem for the Vicon system installed at BRL by fitting e-pucks with 3D-printed hats (shown in Figure A.3), upon which the markers were mounted. The markers have a hole in their underside, allowing them to be placed on pins protruding from the top of the hat. The 3D-printed hats have a 4×6 matrix of pins, providing 24 possible marker positions. This allows each e-puck to be assigned a unique pattern of markers, so that it can be identified by the tracking system.

Initially, the same hat design was tested with the OptiTrack system at YRL. Unfortunately, this was unsuccessful for a number of reasons. Firstly, OptiTrack markers are slightly larger than Vicon markers. Secondly, the OptiTrack cameras are lower resolution than the Vicon MX40 cameras used at BRL (1.3 megapixels vs 4 megapixels). The result is that the 3D-printed hats place the markers too closely together for the OptiTrack cameras to distinguish between them. It was

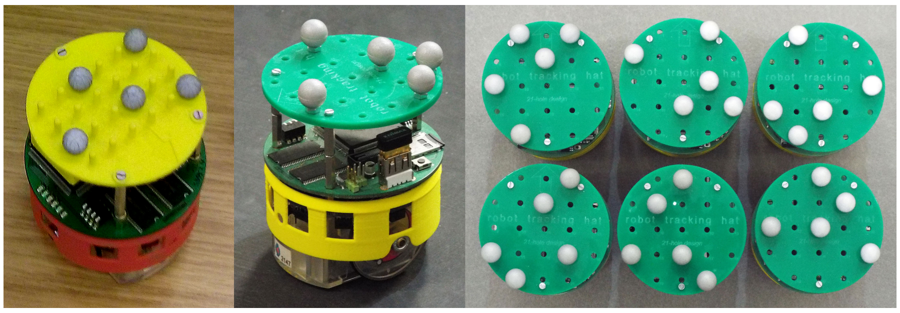


Figure A.3: *Left:* 3D-printed hat design with Vicon markers. *Centre:* Laser-cut hat design with OptiTrack markers. *Right:* Six example marker patterns.

therefore necessary to design a new tracking hat that would allow the markers to be spaced farther apart.

A.3.1 *Laser-cut hat design*

The new tracking hat design[2] is shown in Figure A.3. It is made from a single laser-cut piece of plastic 74 mm in diameter, to match the size of the e-puck so that it does not affect the robot's behaviour. The hat has 21 holes in it, horizontally/vertically separated by 14 mm. The number of holes was chosen as a compromise between maximising the number of marker positions, and maximising the separation between markers. The spacing is sufficient that the tracking system is able to distinguish between diagonally adjacent markers when the cameras are mounted 2.5 m from the ground as shown in Figure A.1.

The original 3D-printed design is quite fragile, and the marker mounting pins are prone to snapping off. The new design overcomes this by instead using bolts pushed upwards through the holes and secured with nuts. The OptiTrack markers are 7/16" (11.11 mm) in diameter and have a hole in their underside, so they rest on the bolts. This method of mounting markers is slightly less convenient, as the bolts must be repositioned when a new pattern is chosen. However, once a set of patterns have been decided upon, it is unlikely that they will need to be changed. This method also has the advantage that the patterns are retained if the markers fall off the hat (a common problem when picking up the robots).

In contrast to the original 3D-printed design, the holes are not laid out in a rectangular grid constrained by the diameter of the hat. Instead they are positioned such that they maximise the available space on the hat. Another difference is that the 3D-printed hat has a slot on

2 Designed by James Hilder — York Robotics Laboratory technician

the underside for mounting a USB Wi-Fi adapter, which connects to the Linux extension board. This slot is not required in the laser-cut design, because an Edimax EW-7811UN nano USB Wi-Fi adapter is used instead, which is small enough to fit under the hat. The new laser-cut design is also cheaper, and faster to produce. The hat is attached to the e-puck with 3 pairs of 15 mm PCB standoffs. It is important to provide sufficient clearance for the Wi-Fi adapter, but the hat should not be mounted too high, otherwise the e-puck will be top-heavy and rock back and forth as it moves.

A.3.2 *Marker patterns*

The tracking software requires that each marker pattern comprises between 3 and 7 markers. Due to the low resolution of the Flex 13 cameras, the tracking system is unable to reliably distinguish between 3 or 4-marker patterns at a distance of 2.5 m from the ground, as the shapes that the markers form appear too similar. Instead, 5-marker patterns were used, which are more easily differentiated at this distance. Due to constraints on the physical size of the hat, and therefore the distance between marker positions, patterns with horizontally/vertically adjacent markers are prohibited. Only diagonally adjacent markers can be resolved by the cameras at this distance. This constraint reduces the number of possible unique marker patterns.

Creating marker patterns is non-trivial, as there are certain situations that must be avoided to prevent ambiguity. Initially, consider a single marker pattern in isolation. If it is is rotationally symmetric, then the tracking software will be unable to determine the true orientation of the robot. Similarly, if the pattern looks identical when flipped, the tracking software may become confused and think that the robot is upside down. For example, a 4-marker pattern that forms a square is symmetrical in both the horizontal and vertical axes, and self-similar under every possible rotation, which is very confusing for the tracking software. When considering a set of marker patterns for tracking a swarm of robots, it is then also necessary to ensure that the patterns appear as distinct as possible, otherwise the robots may be incorrectly identified by the tracking system. It is therefore important to check whether a marker pattern looks similar to another under any possible transformation.

A.3.3 *Evolving marker patterns*

It is infeasible to solve this problem by hand, especially for larger swarm sizes. Instead, Evolutionary Search [117] was used to automatically generate sets of marker patterns that are distinct from each other under any transformation. Each individual in the population comprises a set of marker patterns — one for each robot in the swarm. Each pattern is represented as a vector of integers between 1 and 21, corresponding to the position of the marker holes on the hat. The size of the genome is therefore a product of the number of patterns and the number of markers. For example, for 6 patterns each comprising 5 markers, the genome will be of length 30. Individuals are initialised by randomly generating marker positions for each pattern, with the constraint that a candidate marker position is not already in use, and is not horizontally/vertically adjacent to another marker that has already been placed.

The crossover operator only allows whole marker patterns to be exchanged between individuals, to prevent invalid patterns from being created. The mutation operator iterates over the set of patterns in an individual, and probabilistically decides whether or not to mutate each pattern. If a pattern is chosen for mutation, a single marker within the pattern is selected at random, and a new marker position is randomly generated whilst ensuring that it conforms to occupancy and adjacency constraints. The Multi-Objective Optimisation algorithm NSGA-II [118] was used to maximise the following objective functions:

WORST CASE SHORTEST DISTANCE: The shortest distance between any pair of markers in a pattern, over all patterns in the individual. Maximising this distance ensures that the markers are spread as far apart as possible, making it easier for the tracking system to distinguish between them. The worst case (smallest) value is used, so that the fitness of an individual is limited by the pattern with the most tightly packed markers.

WORST CASE SELF SIMILARITY: Similarity between two marker patterns under a particular transformation is defined as follows: For every marker in pattern A, find the shortest distance to another marker in pattern B. The distance calculated for each marker is then summed. The minimum value of zero is obtained if patterns A and B are identical. The more dissimilar they are, the larger the sum will be. Similarity is calculated under all possible translations, rotations, and reflections, and the worst case value is recorded. Again, the worst case is maximised to ensure that the fitness of an individual is limited by the most self similar pattern it contains.

OVERALL WORST CASE SIMILARITY: This objective is calculated in the same way as self similarity, except that the similarity of each pattern is checked against every other pattern in the individual, and the worst case similarity is recorded. It is desirable to maximise this value, to ensure that each pattern in an individual is as dissimilar to others as possible.

Figure A.3 shows an example set of six marker patterns that were evolved using this approach, and can be reliably differentiated by the tracking system. Due to the problem constraints, and the necessity for patterns to appear distinct from each other under any transformation, only around 10 unique patterns can be evolved. Beyond this number, the evolved patterns begin to look similar, which can result in the tracking system incorrectly identifying robots. This limitation is influenced by the size of the hats, and therefore the separation between markers, as well as the distance, angle, and number of cameras. For a smaller arena, with the cameras closer to the ground, any placement of markers on the hats could be used without enforcing adjacency constraints, which would allow for more unique patterns to be evolved.

Although the focus here is on e-puck robots, it is important to note that the tracking system is robot platform agnostic. Any robot can be tracked provided that a pattern of markers can be mounted on it. The only limitation is that there must be sufficient spacing between the markers, so that the tracking system can distinguish between them. In order to track robots smaller than the e-puck, it may be necessary to use smaller markers that OptiTrack offer for tracking facial expressions, either 3 or 4 mm in diameter, and to position the cameras closer to the ground. For significantly larger robots, 5/8" or 3/4" diameter markers are available, which can be tracked at a greater distance.

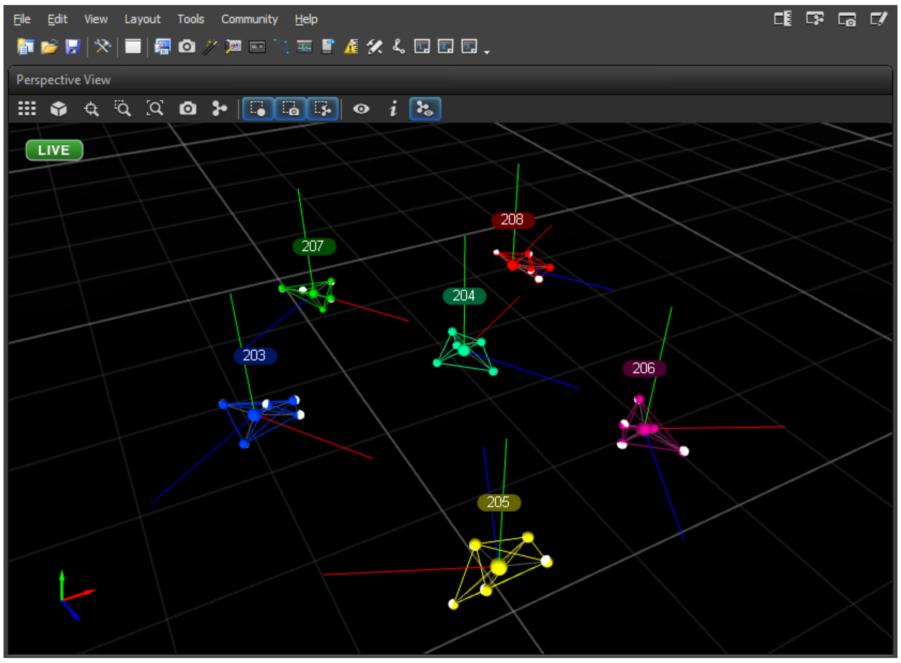


Figure A.4: Screenshot of the Motive:Tracker software that shows the position and orientation of six rigid bodies (one for each e-puck) being tracked. Each rigid body comprises 5 markers. The pivot point of each rigid body is also displayed.

A.4 TRACKING SOFTWARE

The tracking software (Motive:Tracker[3]) is capable of tracking objects in six degrees of freedom. Figure A.4 shows a screenshot of the graphical user interface (GUI) when tracking six e-puck robots. The minimum system requirements to run software version 1.5 are a 2.5 GHz i5 processor and 4 GB of memory. This section provides a brief overview of the workflow when working with the tracking software.

The tracking system must be calibrated before it is first used. This is achieved using an OptiWand and Calibration Square (see Figure A.5). First, the OptiWand is waved through the entire capture volume while the cameras observe its movement by detecting the three retroreflective markers mounted on it. Once sufficient coverage is achieved, the wanding data is used to automatically calculate the physical position, orientation, and lens characteristics of the cameras. Projecting the Motive:Tracker GUI onto the wall as shown in Figure A.1 makes calibration significantly easier, as the user can easily observe the extent of capture volume coverage during the wanding process. This is

3 `http://optitrack.com/products/motive/tracker`

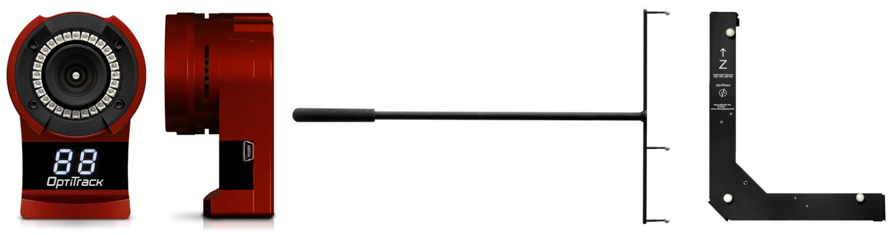


Figure A.5: *Left:* Flex 13 camera. *Centre:* OptiWand. *Right:* Calibration Square[4].

more difficult when the GUI is displayed on a monitor, as it is too small to see from a distance.

After the cameras have been calibrated, the Calibration Square is placed on the floor to align the 3D coordinate system with the physical capture volume. The calibration process is then complete, and the calibration tools are removed from the capture volume so that rigid bodies can be defined. Note that if the cameras are moved after calibration, then the system must be recalibrated. This is why the cameras are mounted on a permanent truss rig. The initial calibration results can be saved and reused for each subsequent capture session.

Following calibration, the GUI is used to select markers belonging to a robot in the software and create rigid body from it. This allows the tracking software to calculate the position and orientation of the robot based on the coordinates of individual markers in each frame. Figure A.4 shows the rigid bodies corresponding to the marker patterns shown in Figure A.3. After creating a rigid body, it is then necessary to adjust the orientation and pivot point of each rigid body to match that of the real robot. The Motive:Tracker GUI can be used to record tracking data for post-experiment analysis, but the true potential of the tracking system is realised through integration with custom code via the API. The API is written in C++, and provides direct real-time access to rigid body tracking data. However, it is first necessary to perform calibration, create rigid bodies, and adjust any parameters via the GUI, then save the project to a file which can be loaded using the API. The tracking data can then be used in various applications, as detailed in the next section.

4 Images taken from `https://www.optitrack.com`

A.5 APPLICATIONS

Tracking infrastructures have many potential applications in swarm robotics research beyond the fault detection approach proposed in this thesis, a few of which are described below. Although similar tracking systems have already been used in most of these application areas, the system presented in this thesis offers comparable functionality at a relatively low cost.

Post-experiment analysis

The most obvious application of the tracking infrastructure is recording the behaviour of a robot swarm over time. If desired, the robots can additionally transmit data about their current state to the OptiTrack server, which can be aggregated with tracking data. The data can then be used for post-experiment analysis, and to verify the success of an experiment using statistical hypothesis testing. For example, Bjerknes and Winfield [8] used the Vicon system at BRL to analyse the fault tolerance of a robot swarm.

Validation

The tracking infrastructure can be used to carry out repeated experiments, by commanding robots to move to specific starting positions to initialise each test run. This is particularly important for validating swarm algorithms that rely on self-organisation, as they must be analysed over repeated experimental runs, to check that undesirable behaviours do not emerge. Indeed, Winfield et al. [22] argue that the ability to repeatably test real robot swarm systems is important if dependable swarms are ever to be developed.

Localisation

The tracking system can be used to provide real-time feedback to the robots about their current position and orientation. This gives the robots awareness of their location within their environment. The Vicon-based system at the GRASP Laboratory at the University of Pennsylvania has been for localisation with a swarm of 16 quadrocopters, to implement centralised formation control [119].

Rapid prototyping

Tracking data can be used to initially provide robots in a swarm with global information, to quickly test new concepts. This approach to development could similarly be carried out purely in simulation, but using a tracking infrastructure makes the system more grounded in reality, due to the use of real robot sensors and actuators. The Flying Machine Arena [120] built at ETH Zurich uses Vicon cameras to track quadrocopters, for rapid-prototyping and performance validation and evaluation.

Virtual sensors

The tracking infrastructure can be used to implement virtual sensors for the robots. This is particularly useful if the robot platform used has limited sensing capabilities. For example, an omnidirectional camera could be emulated, allowing a robot to determine the relative position of other robots. This can be limited to local sensing, by calculating the distance between pairs of robots, and transmitting a limited view of the world to each robot in the swarm. The Vicon system at BRL has been used successfully to implement virtual sensors for swarms of e-puck robots, allowing for online evolution of collective behaviours [72, 114].

Automated calibration of simulation

Tracking data could also potentially be used for the validation and calibration of robot simulators. This would be of great benefit, as the reality gap of existing simulators is often large, and swarm algorithms developed in simulation often do not work without significant modification when deployed on real robots.

A.6 COST

Table A.1 shows the cost of the OptiTrack hardware and software required to build a tracking infrastructure similar to the one presented here. The only difference is that the cost of 10 ft (3 m) stands to mount the cameras on have been included, as these are significantly cheaper than a truss rig. There are some overhead costs associated with the calibration tools and the tracking software. The hardware key is required for licensing — the Motive:Tracker software may be installed on multiple machines at no cost, but can only be used when a licensed

Item	Unit Cost	Quantity	Subtotal
Flex 13 camera	$1098	3	$3294
OptiHub 2	$299	1	$299
Hardware Key	$99	1	$99
Motive:Tracker Software	$999	1	$999
Calibration Square	$99	1	$99
OptiWand Kit	$249	1	$249
Reflective Markers	$20	10	$200
Camera Stand: 10 ft	$99	3	$297
SLIK Clamp Head 38 mm	$69	3	$207
USB Cable: 5 m	$10	3	$30
USB Uplink Cable: 16 ft	$5	1	$5
		Total:	**$5778**

Table A.1: Cost of OptiTrack hardware and software for a three camera system, with enough markers for tracking 10 robots. All prices quoted are in USD.

hardware key is connected via USB. To extend the system after these initial overheads, it would only be at the expense of additional cameras, hubs, and cables. The cost of tracking swarms of larger sizes is dictated only by the cost of extra markers.

A.7 LIMITATIONS

While it has many benefits, the tracking infrastructure presented in this thesis has some limitations. Firstly, the tracking software will only run under the Windows operating system, and the API is written in C++. This is not a major issue for this research, as the tracking data can be transmitted to Linux systems via network sockets, but it is something worth considering when planning to build an OptiTrack system. Secondly, a maximum of 32 rigid bodies can be defined in Motive:Tracker. This imposes an upper limit on the size of swarms that can be tracked.

Finally, the IR light from the cameras can interfere with a robot's IR sensors, potentially resulting in erratic behaviour. This is one major disadvantage that OptiTrack systems have in comparison to Vicon systems, which offer cameras with 'visible red light' and 'near-infrared' modes of operation that do not cause IR interference. Despite this limitation, strobing the Flex 13 cameras much lower than their maximum frame rate of 120 fps greatly alleviates these issues.

The Motive:Tracker software imposes a minimum frame rate of 30 fps, but the cameras can be strobed at an even lower frame rate if desired, by connecting an external signal generator to the OptiHub 2. Another potential solution might be to use the Flex 13 cameras in passive mode, where instead of emitting IR light themselves, they sense light from active IR LED markers mounted on the robots. However, the tracking infrastructure presented in this appendix has been able to successfully track the behaviour of an e-puck swarm running the ω-algorithm algorithm (described in Section 2.1.1), which relies entirely on IR sensing and communication. This was achieved with minimal IR interference by strobing the cameras at 30 fps, although the ω-algorithm does perform filtering on the raw IR sensor readings. An external signal generator could potentially also be used to synchronise the strobing of the cameras with the IR sensing of the e-pucks, to ensure that they are mutually exclusive, and therefore do not interfere with each other.

A.8 SUMMARY

This appendix has given details of the tracking infrastructure built for the research presented in this thesis, and the challenges encountered when constructing and configuring the system. The tracking infrastructure has many potential uses beyond the research presented in this thesis, and provides a cost-effective alternative to more expensive commercial motion capture systems on the market.

In the context of exogenous fault detection, the tracking system allows the true behaviour of a robot to be recorded, and compared against its expected behaviour predicted in simulation. The observation data made available to each robot can also be range-limited by implementing virtual sensors, thus enabling decentralised and scalable fault detection.

B | GLOBAL SENSITIVITY ANALYSIS

This appendix presents the full results of the global sensitivity analysis that was carried out on the fault detector. For a detailed discussion of the results, see Section 6.7 of Chapter 6.

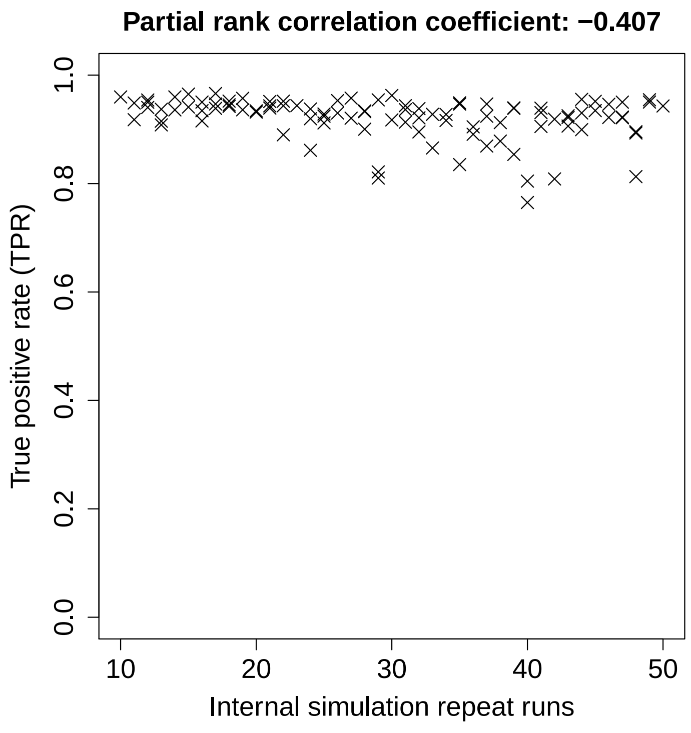


Figure B.1: Influence of the number of internal simulation repeat runs on the TPR.

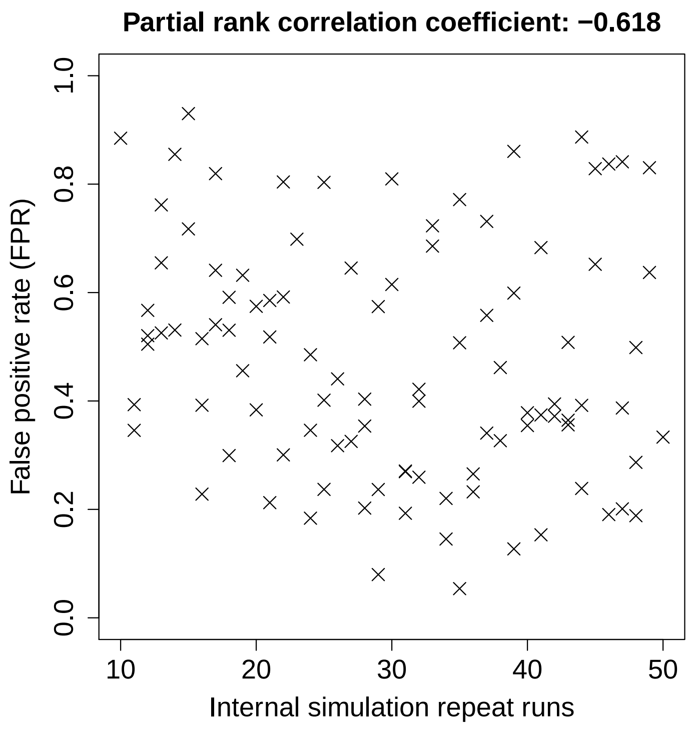


Figure B.2: Influence of the number of internal simulation repeat runs on the FPR.

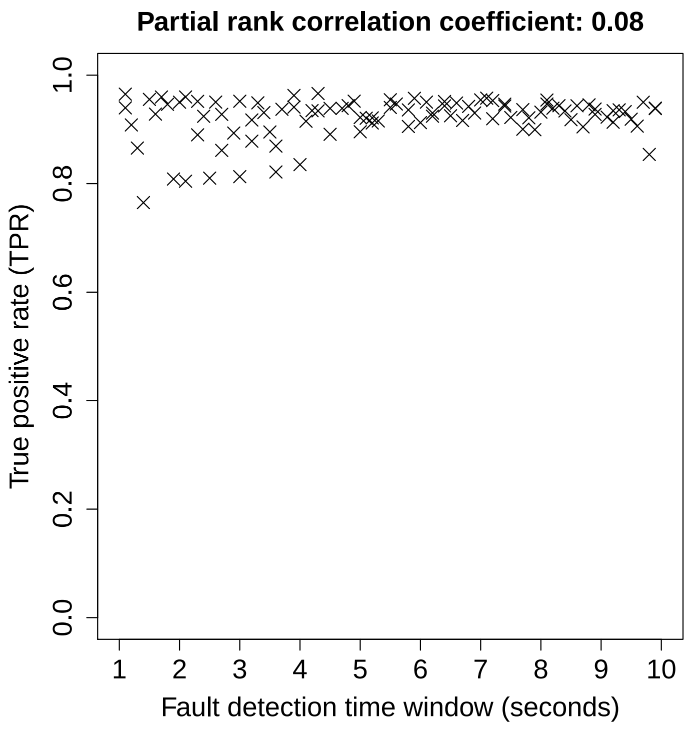


Figure B.3: Influence of the fault detection time window length on the TPR.

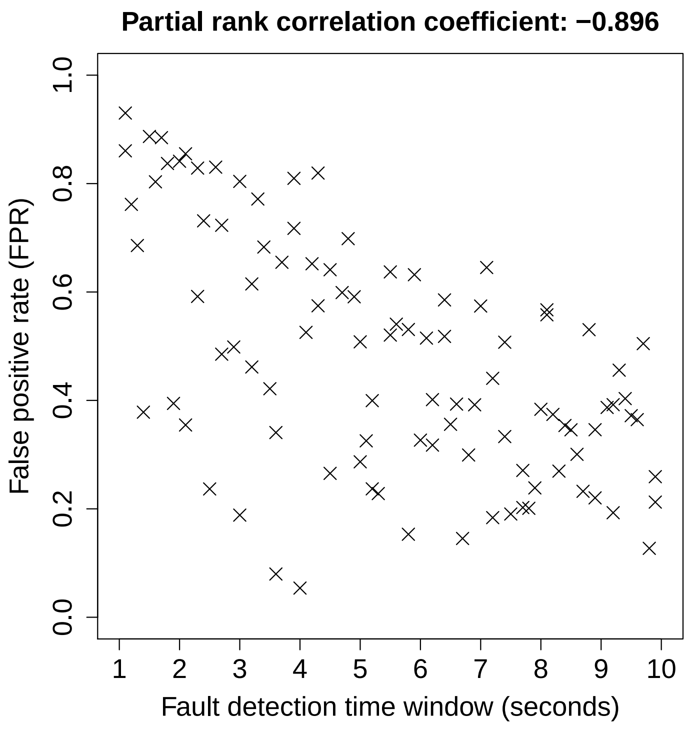


Figure B.4: Influence of the fault detection time window length on the FPR.

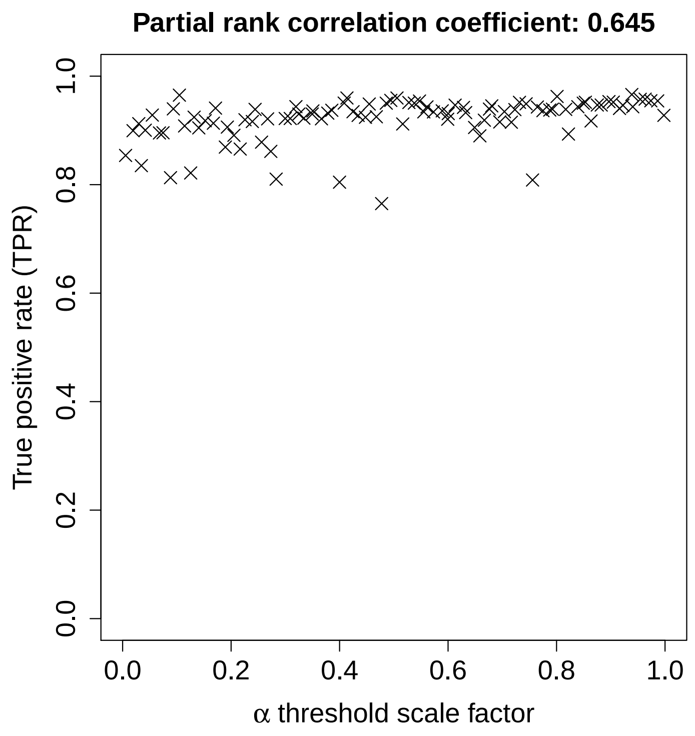


Figure B.5: Influence of the α threshold scale factor on the TPR.

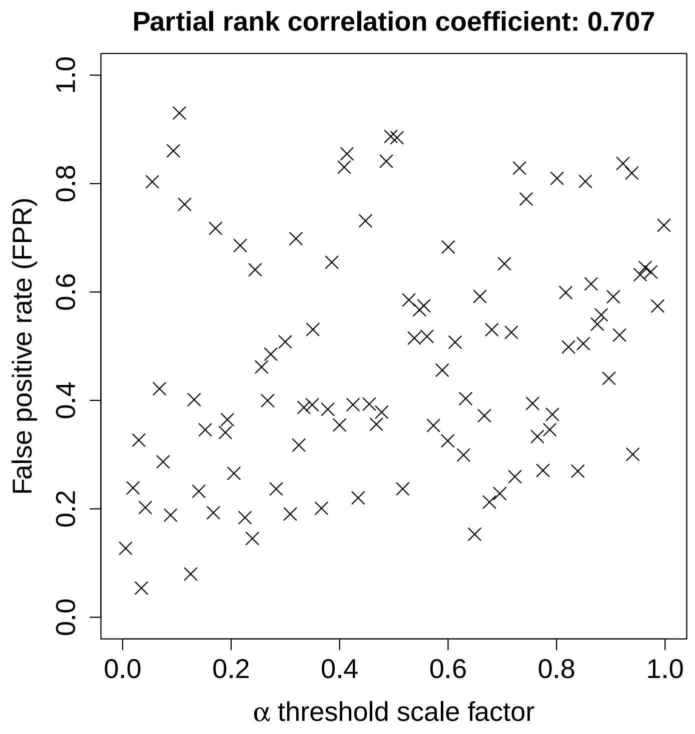


Figure B.6: Influence of the α threshold scale factor on the FPR.

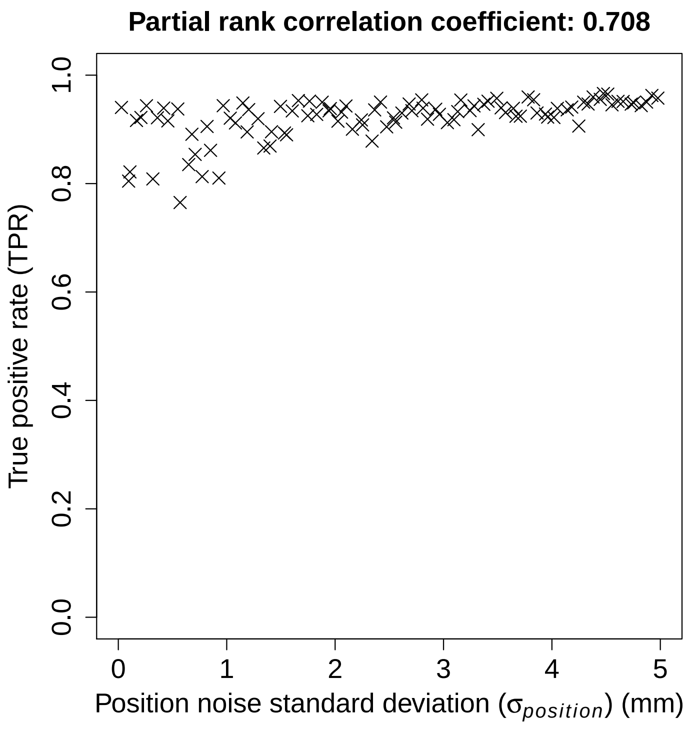


Figure B.7: Influence of the position noise standard deviation ($\sigma_{position}$) on the TPR.

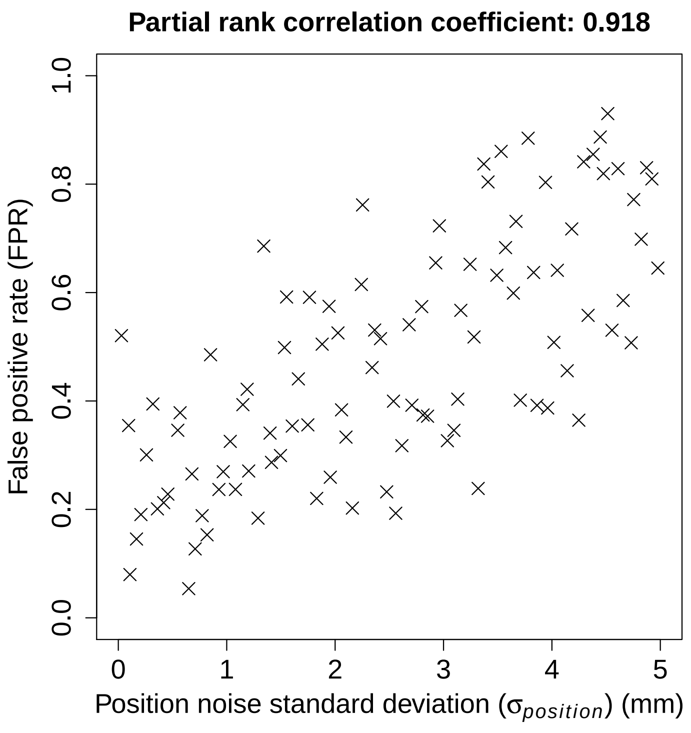


Figure B.8: Influence of the position noise standard deviation ($\sigma_{position}$) on the FPR.

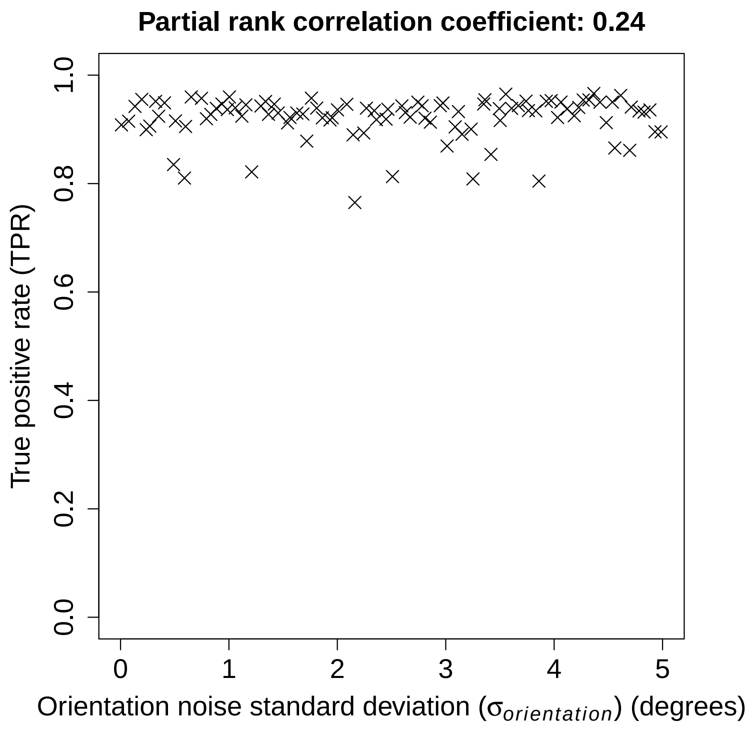


Figure B.9: Influence of the orientation noise standard deviation ($\sigma_{orientation}$) on the TPR.

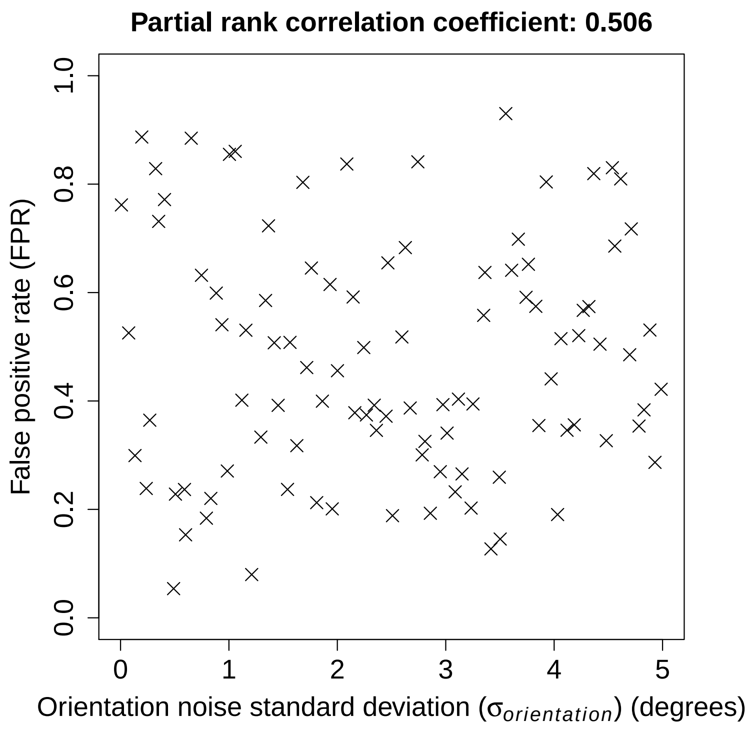


Figure B.10: Influence of the orientation noise standard deviation ($\sigma_{orientation}$) on the FPR.

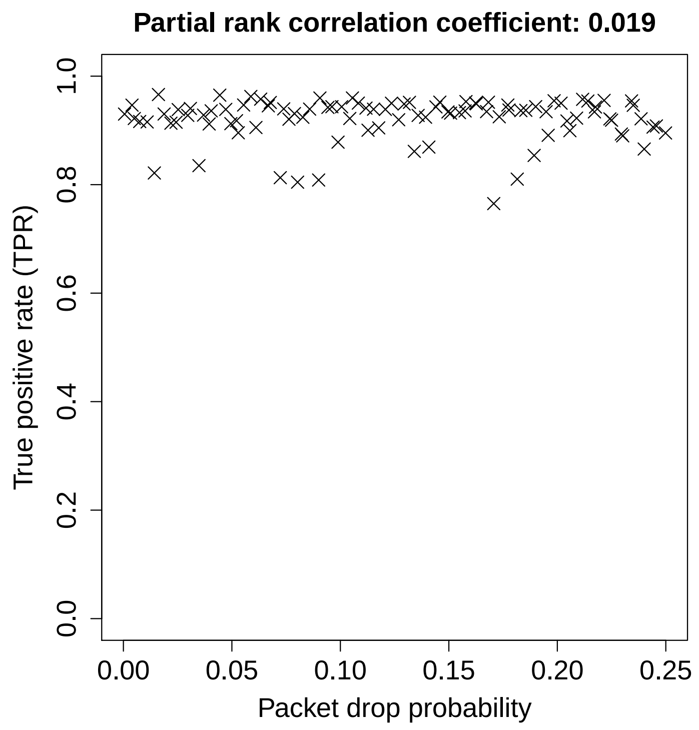


Figure B.11: Influence of the packet drop probability on the TPR.

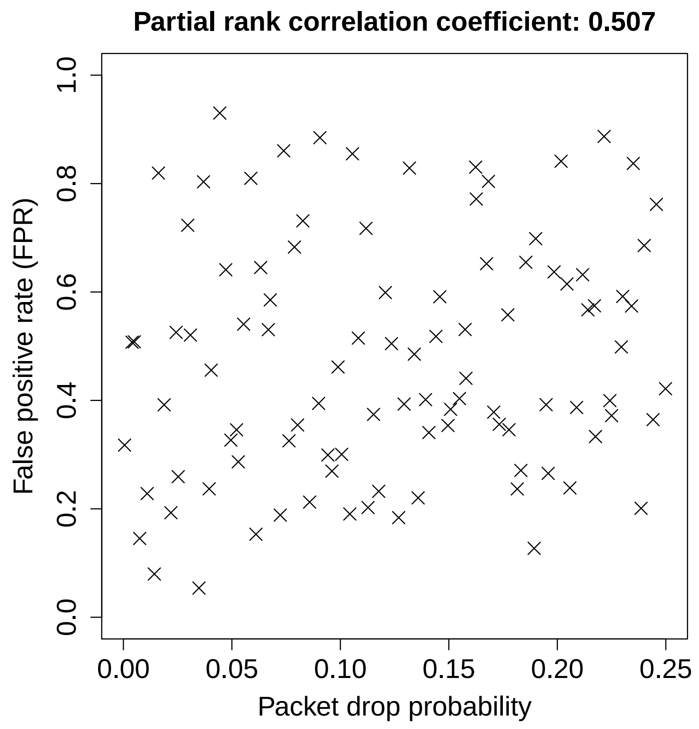


Figure B.12: Influence of the packet drop probability on the FPR.

C FALSE POSITIVE ANALYSIS

This appendix presents example scenarios that resulted in false positive classifications. Each scenario visualises the output of a single observer's internal simulation of a non-faulty robot. The square represents the focal robot's starting point, and its predicted endpoints are shown with crosses. The grey region indicates the area within which the focal robot was expected to end up, under the assumption that was non-faulty. The true endpoint of the focal robot is represented by the circle.

In each of these scenarios, the focal robot's true endpoint lies outside the non-linear decision boundary that is created by thresholding the kernel density estimate of the predicted endpoint distribution — suggesting that the robot is faulty. The focal robot was actually non-faulty in every scenario, therefore each classification is incorrect (a false positive). However, note that the focal robot's true endpoint is often very close to the edge of the expected non-faulty region. These borderline cases could be avoided by lowering the α threshold scale factor to increase the size of the predicted non-faulty endpoint region (at the expense of the TPR), as shown in Figure B.6 of Appendix B.

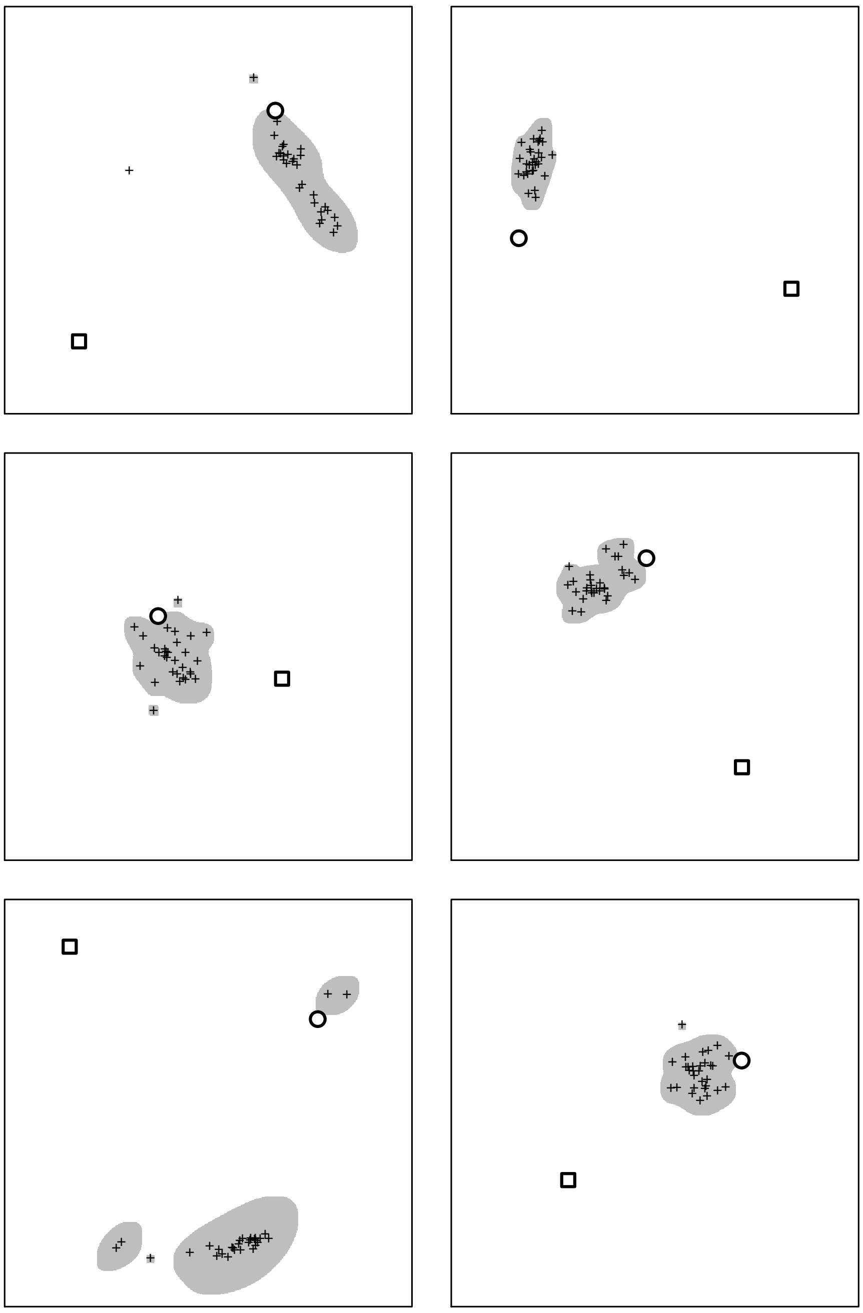


Figure C.1: False positive analysis — example scenarios 1–6.

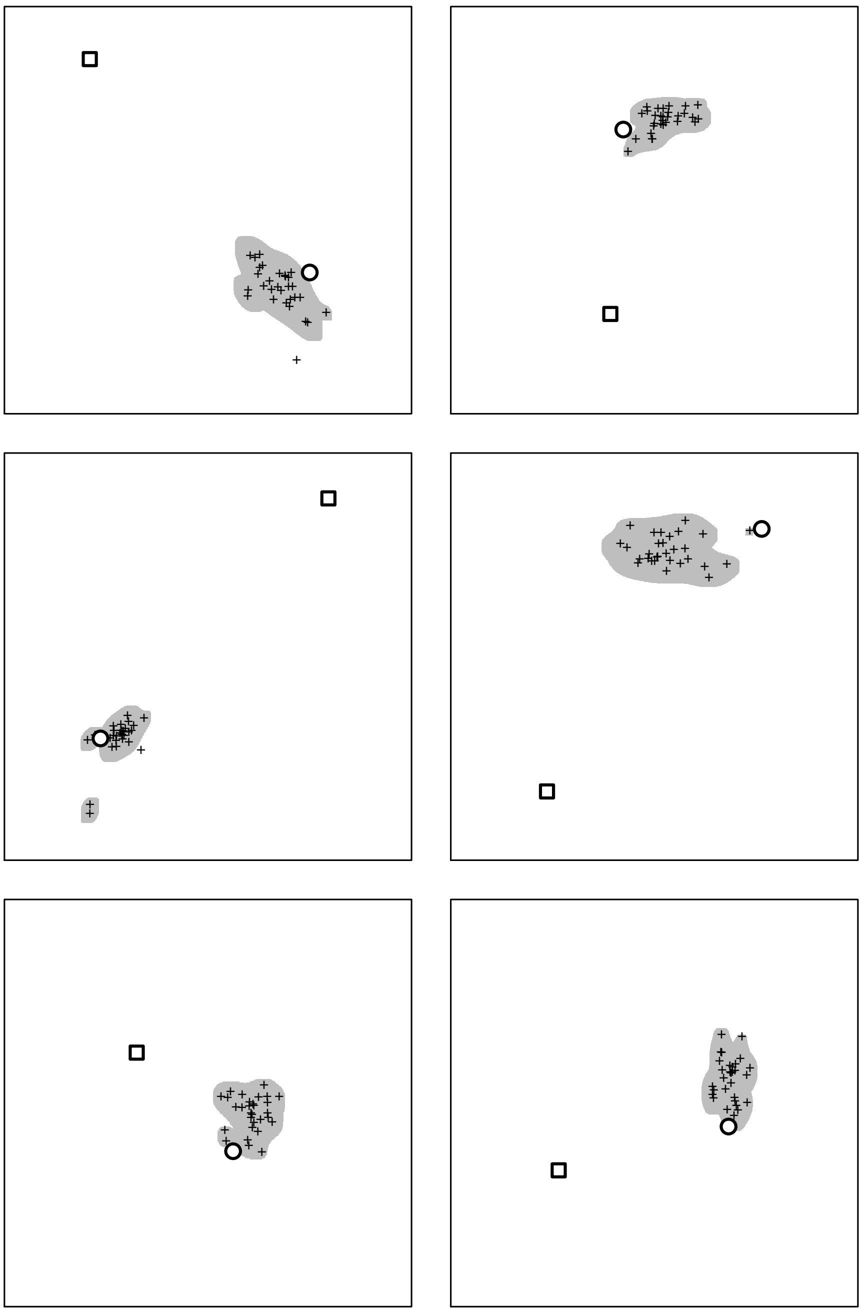


Figure C.2: False positive analysis — example scenarios 7–12.

Part VI

REFERENCES

[1] A. G. Millard, J. Timmis, and A. F. T. Winfield, "Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2014, pp. 3720–3725.

[2] ——, "Towards exogenous fault detection in swarm robotic systems," in *Proceedings of Towards Autonomous Robotic Systems (TAROS).* Springer, 2014, pp. 429–430.

[3] A. G. Millard, J. A. Hilder, J. Timmis, and A. F. T. Winfield, "A low-cost real-time tracking infrastructure for ground-based robot swarms," in *Proceedings of the 9th International Conference on Swarm Intelligence (ANTS).* Springer, 2014, pp. 278–289.

[4] ——, "A low-cost real-time tracking infrastructure for ground-based robot swarms," in *Technical Report YCS-2014-489.* University of York, 2014, pp. 1–13.

[5] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics: SAB 2004 International Workshop, Revised Selected Papers.* Springer, 2005, pp. 10–20.

[6] A. F. T. Winfield and J. Nembrini, "Safety in numbers: fault-tolerance in robot swarms," *International Journal of Modelling, Identification and Control*, vol. 1, no. 1, pp. 30–37, 2006, Inderscience.

[7] J. D. Bjerknes, "Scaling and fault tolerance in self-organized swarms of mobile robots," Ph.D. dissertation, University of the West of England, Bristol, UK, 2010.

[8] J. D. Bjerknes and A. F. T. Winfield, "On fault tolerance and scalability of swarm robotic systems," in *Distributed Autonomous Robotic Systems: The 10th International Symposium.* Springer, 2013, pp. 431–444.

[9] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Exogenous fault detection in a collective robotic task," in *Proceedings of the 9th European Conference on Advances in Artificial Life (ECAL).* Springer, 2007, pp. 555–564.

[10] A. L. Christensen, R. O'Grady, and M. Dorigo, "From fireflies to fault-tolerant swarms of robots," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 754–766, 2009, IEEE.

[11] A. Khadidos, R. M. Crowder, and P. H. Chappell, "Exogenous fault detection and recovery for swarm robotics," in *Proceedings of the 15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, vol. 48, no. 3. Elsevier, 2015, pp. 2405–2410.

[12] D. Tarapore, P. U. Lima, J. Carneiro, and A. L. Christensen, "To err is robotic, to tolerate immunological: fault detection in multirobot systems," *Bioinspiration & Biomimetics*, vol. 10, no. 1, pp. 1–19, 2015, IOP Publishing.

[13] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014, American Association for the Advancement of Science.

[14] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013, Springer.

[15] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 3293–3298.

[16] E. Şahin and A. F. T. Winfield, "Special issue on swarm robotics," *Swarm Intelligence*, vol. 2, pp. 69–72, 2008, Springer.

[17] G. Beni, "From swarm intelligence to swarm robotics," in *Swarm Robotics: SAB 2004 International Workshop, Revised Selected Papers*. Springer, 2005, pp. 1–9.

[18] K. Petersen, R. Nagpal, and J. Werfel, "TERMES: An autonomous robotic system for three-dimensional collective construction," in *Proceedings of Robotics: Science & Systems VII*. MIT Press, 2011, pp. 257–264.

[19] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014, American Association for the Advancement of Science.

[20] V. Trianni and A. Campo, "Fundamental collective behaviors in swarm robotics," in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 1377–1394.

[21] L. Bayındır, "A review of swarm robotics tasks," *Neurocomputing*, vol. 172, pp. 292–321, 2016, Elsevier.

[22] A. F. T. Winfield, C. J. Harper, and J. Nembrini, "Towards dependable swarms and a new discipline of swarm engineering," in *Swarm Robotics: SAB 2004 International Workshop, Revised Selected Papers*. Springer, 2005, pp. 126–142.

[23] L. Bayındır and E. Şahin, "A review of studies in swarm robotics," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 15, no. 2, pp. 115–147, 2007, The Scientific and Technological Research Council of Turkey.

[24] L. E. Parker, "Reliability and fault tolerance in collective robot systems," in *Handbook of Collective Robotics: Fundamentals and Challenges*. Pan Stanford Publishing, 2013, pp. 167–204.

[25] L. Murray, "Fault tolerant morphogenesis in self-reconfigurable modular robotic systems," Ph.D. dissertation, University of York, York, UK, 2013.

[26] A. J. Sharkey, "Swarm robotics and minimalism," *Connection Science*, vol. 19, no. 3, pp. 245–260, 2007, Taylor & Francis.

[27] J. Nembrini, "Minimalist coherent swarming of wireless networked autonomous mobile robots," Ph.D. dissertation, University of the West of England, Bristol, UK, 2005.

[28] K. W. Dailey, *The FMEA Handbook*. DW Publishing, 2004.

[29] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, no. 1. IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[30] J. Carlson and R. Murphy, "Reliability analysis of mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2003, pp. 274–281.

[31] J. Carlson, R. Murphy, and A. Nelson, "Follow-up analysis of mobile robot failures," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 5. IEEE, 2004, pp. 4987–4994.

[32] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach.* Springer, 2002.

[33] J. Timmis, A. Tyrrell, M. Mokhtar, A. R. Ismail, N. Owens, and R. Bi, "An artificial immune system for robot organisms," in *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, ser. Cognitive Systems Monographs. Springer, 2010, vol. 7, pp. 282–306.

[34] C. A. Janeway, P. Travers, M. Walport, and M. J. Shlomchik, *Immunobiology: The Immune System in Health and Disease*, 6th ed. Garland Science, 2006.

[35] I. Cohen, *Tending Adam's Garden: Evolving the Cognitive Immune Self.* Academic Press, 2004.

[36] S. Cremer, S. A. O. Armitage, and P. Schmid-Hempel, "Social immunity," *Current Biology*, vol. 17, no. 16, pp. R693–R702, 2007, Elsevier.

[37] S. Stepney, R. E. Smith, J. Timmis, A. M. Tyrrell, M. J. Neal, and A. N. W. Hone, "Conceptual frameworks for artificial immune systems," *International Journal of Unconventional Computing*, vol. 1, no. 3, pp. 315–338, 2005, Old City Publishing.

[38] J. Timmis, P. Andrews, N. Owens, and E. Clark, "An interdisciplinary perspective on artificial immune systems," *Evolutionary Intelligence*, vol. 1, pp. 5–26, 2008, Springer.

[39] V. Trianni, "On the evolution of self-organising behaviours in a swarm of autonomous robots," Ph.D. dissertation, Université Libre de Bruxelles, Brussels, Belgium, 2006.

[40] S. Cremer and M. Sixt, "Analogies in the evolution of individual and social immunity," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1513, pp. 129–142, 2009, The Royal Society.

[41] L. N. de Castro and J. Timmis, "Artificial immune systems: a novel paradigm to pattern recognition," *Artificial neural networks in pattern recognition*, vol. 1, pp. 67–84, 2002, Springer.

[42] X. Wang, X. Z. Gao, and S. J. Ovaska, "Artificial immune optimization methods and applications-a survey," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4. IEEE, 2004, pp. 3415–3420.

[43] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont, "An artificial immune system architecture for computer security applications," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 252–280, 2002, IEEE.

[44] J. Kim, P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross, "Immune system approaches to intrusion detection – a review," *Natural Computing*, vol. 6, no. 4, pp. 413–466, 2007, Springer.

[45] N. Bayar, S. Darmoul, S. Hajri-Gabouj, and H. Pierreval, "Fault detection, diagnosis and recovery using artificial immune systems: A review," *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 43–57, 2015, Elsevier.

[46] J. Timmis, P. Andrews, and E. Hart, "On artificial immune systems and swarm intelligence," *Swarm Intelligence*, vol. 4, no. 4, pp. 247–273, 2010, Springer.

[47] S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, D. Laneri, L. Winkler, T. Schmickl, R. Thenius *et al.*, "Symbiotic robot organisms: REPLICATOR and SYMBRION projects," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. ACM, 2008, pp. 62–69.

[48] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, H. Raja, W. Liu, F. Caparrelli, J. Jemai, J. Havlik *et al.*, "Multi-robot organisms: state of the art," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 1–10.

[49] M. Mokhtar, R. Bi, J. Timmis, and A. Tyrrell, "A modified dendritic cell algorithm for on-line error detection in robotic systems," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2009, pp. 2055–2062.

[50] J. Greensmith, U. Aickelin, and J. Twycross, "Articulation and clarification of the dendritic cell algorithm," *Artificial Immune Systems*, pp. 404–417, 2006, Springer.

[51] A. R. Ismail, J. D. Bjerknes, J. Timmis, and A. F. T. Winfield, "An artificial immune system for self-healing in swarm robotic systems," in *Information Processing in Cells and Tissues*. Springer, 2015, pp. 61–74.

[52] M. C. Sneller, "Granuloma formation, implications for the pathogenesis of vasculitis," *Cleveland Clinic Journal of Medicine*, vol. 69, pp. SII–40–SII–43, 2002, Cleveland Clinic.

[53] J. L. Flynn, "Mutual attraction: Does it benefit the host or the bug?" *Nature Immunology*, vol. 5, pp. 778–779, 2004, Nature Publishing Group.

[54] J. Timmis, E. Hart, A. Hone, M. Neal, A. Robins, S. Stepney, and A. Tyrrell, "Immuno-engineering," in *Biologically-Inspired Collaborative Computing*. Springer, 2008, pp. 3–17.

[55] A. R. Ismail, "Immune-inspired self-healing swarm robotic systems," Ph.D. dissertation, University of York, York, UK, 2011.

[56] H. K. Lau, I. Bate, P. Cairns, and J. Timmis, "Adaptive data-driven error detection in swarm robotics with statistical classifiers," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1021–1035, 2011, Elsevier.

[57] R. Isermann, "Supervision, fault-detection and fault-diagnosis methods — an introduction," *Control Engineering Practice*, vol. 5, no. 5, pp. 639–652, 1997, Elsevier.

[58] A. L. Christensen, "Fault detection in autonomous robots," Ph.D. dissertation, Université Libre de Bruxelles, Brussels, Belgium, 2008.

[59] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009, ACM.

[60] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004, Springer.

[61] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Automatic synthesis of fault detection modules for mobile robots," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2007, pp. 693–700.

[62] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989, IEEE.

[63] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo, "SWARM-BOT: A new distributed robotic concept," *Autonomous Robots*, vol. 17, no. 2-3, pp. 193–221, 2004, Springer.

[64] 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, Springer.

[65] H. K. Lau, "Error detection in swarm robotics: A focus on adaptivity to dynamic environments," Ph.D. dissertation, University of York, York, UK, 2012.

[66] N. D. Owens, A. Greensted, J. Timmis, and A. Tyrrell, "T cell receptor signalling inspired kernel density estimation and anomaly detection," in *Artificial Immune Systems*. Springer, 2009, pp. 122–135.

[67] H. K. Lau, I. Bate, and J. Timmis, "Immune-inspired error detection for multiple faulty robots in swarm robotics," in *Proceedings of the 12th European Conference on the Synthesis and Simulation of Living Systems (Advances in Artificial Life, ECAL)*, vol. 12. MIT Press, 2013, pp. 846–853.

[68] K. Leon, R. Perez, A. Lage, and J. Carneiro, "Modelling T-cell-mediated suppression dependent on interactions in multicellular conjugates," *Journal of Theoretical Biology*, vol. 207, no. 2, pp. 231–254, 2000, Elsevier.

[69] D. Tarapore, A. L. Christensen, and J. Timmis, "Abnormality detection in robots exhibiting composite swarm behaviours," in *Proceedings of the 13th European Conference on Artificial Life (ECAL)*. MIT Press, 2015, pp. 406–413.

[70] P. J. O'Dowd, "An embodied simulation approach to distributed evolution for swarm robotic systems," Ph.D. dissertation, University of the West of England, Bristol, UK, 2012.

[71] P. J. O'Dowd, A. F. T. Winfield, and M. Studley, "Towards accelerated distributed evolution for adaptive behaviours in swarm robotics," in *Proceedings of Towards Autonomous Robotic Systems (TAROS)*. Univeristy of Plymouth, 2010, pp. 169–175.

[72] P. O'Dowd, A. F. T. Winfield, and M. Studley, "The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 4995–5000.

[73] P. J. O'Dowd, M. Studley, and A. F. T. Winfield, "The distributed co-evolution of an on-board simulator and controller for swarm robot behaviours," *Evolutionary Intelligence*, vol. 7, no. 2, pp. 95–106, 2014, Springer.

[74] W. Liu and A. F. T. Winfield, "Open-hardware e-puck Linux extension board for experimental swarm robotics research," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 60–67, 2011, Elsevier.

[75] A. F. T. Winfield, C. Blum, and W. Liu, "Towards an ethical robot: internal models, consequences and ethical action selection," in *Advances in Autonomous Robotics Systems*. Springer, 2014, pp. 85–96.

[76] A. F. T. Winfield, "Robots with internal models: a route to self-aware and hence safer robots," in *The Computer After Me: Awareness and Self-Awareness in Autonomic Systems*. Imperial College Press, 2014, pp. 232–257.

[77] C. Blum, "Self-organization in networks of mobile sensor nodes," Ph.D. dissertation, Humboldt University of Berlin, Berlin, Germany, 2015.

[78] R. Vaughan, "Massively multi-robot simulation in Stage," *Swarm Intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008, Springer.

[79] "Wikimedia Commons: Photograph of the e-puck mobile robot," `https://commons.wikimedia.org/wiki/File:E-puck-mobile-robot-photo.jpg`.

[80] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," *Advances in Artificial Life*, pp. 704–720, 1995, Springer.

[81] N. Jakobi, "Evolutionary robotics and the radical envelope-of-noise hypothesis," *Adaptive Behavior*, vol. 6, no. 2, pp. 325–368, 1997, SAGE Publications.

[82] M. Quinn, L. Smith, G. Mayley, and P. Husbands, "Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1811, pp. 2321–2343, 2003, The Royal Society.

[83] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*. IEEE, 2003, pp. 317–323.

[84] "The Player Project, Player: Cross-platform robot device interface & server," `http://playerstage.sourceforge.net/index.php?src=player`.

[85] "The Player Project, Stage: 2D multiple-robot simulator," `http://playerstage.sourceforge.net/index.php?src=stage`.

[86] A. F. T. Winfield and M. D. Erbas, "On embodied memetic evolution and the emergence of behavioural traditions in robots," *Memetic Computing*, vol. 3, no. 4, pp. 261–270, 2011, Springer.

[87] Á. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range & bearing miniaturized board for local communication in swarm robotics," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 3111–3116.

[88] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*. MIT Press, 1986.

[89] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. D. Caro, F. Ducatelle, T. Stirling, Á. Gutiérrez, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 5027–5034.

[90] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a

modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012, Springer.

[91] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy *et al.*, "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics & Automation Magazine*, vol. 20, no. 4, pp. 60–71, 2013, IEEE.

[92] L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, and M. Birattari, "Software infrastructure for e-puck (and TAM)," Université Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2015-004, 2015.

[93] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962, Institute of Mathematical Statistics.

[94] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, 2nd ed. Wiley, 2015.

[95] K.-R. Koch, *Introduction to Bayesian statistics*. Springer, 2007.

[96] T. Duong, "ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R," *Journal of Statistical Software*, vol. 21, no. 1, pp. 1–16, 2007, American Statistical Association.

[97] D. Eddelbuettel, *Seamless R and C++ Integration with Rcpp*, ser. Use R! Springer, 2013, ch. RInside, pp. 127–137.

[98] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2008, `http://www.r-project.org`.

[99] M. Read, P. S. Andrews, J. Timmis, and V. Kumar, "Techniques for grounding agent-based simulations in the real domain: a case study in experimental autoimmune encephalomyelitis," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 18, no. 1, pp. 67–86, 2012, Taylor & Francis.

[100] K. Alden, M. Read, J. Timmis, P. S. Andrews, H. Veiga-Fernandes, and M. Coles, "Spartan: a comprehensive tool for understanding uncertainty in simulations of biological systems," *PLOS Computational Biology*, vol. 9, no. 2, pp. 1–9, 2013, PLOS.

[101] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000, Sage Publications.

[102] A. Saltelli, K. Chan, and E. M. Scott, *Sensitivity Analysis*. Wiley, 2008.

[103] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global Sensitivity Analysis: The Primer*. Wiley, 2008.

[104] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000, Taylor & Francis.

[105] S. Marino, I. B. Hogue, C. J. Ray, and D. E. Kirschner, "A methodology for performing global uncertainty and sensitivity analysis in systems biology," *Journal of Theoretical Biology*, vol. 254, no. 1, pp. 178–196, 2008, Elsevier.

[106] E. Olson, J. Strom, R. Goeddel, R. Morton, P. Ranganathan, and A. Richardson, "Exploration and mapping with autonomous robot teams," *Communications of the ACM*, vol. 56, no. 3, pp. 62–70, 2013, ACM.

[107] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3400–3407.

[108] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[109] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Springer Handbook of Robotics*. Springer, 2008, pp. 871–889.

[110] S. M. Poulding, "The use of automated search in deriving software testing strategies," Ph.D. dissertation, University of York, York, UK, 2013.

[111] NVIDIA, "CUDA Compute Unified Device Architecture programming guide," 2007.

[112] S. Jones, M. Studley, and A. F. T. Winfield, "Mobile GPGPU acceleration of embodied robot simulation," in *Artificial Life and Intelligent Agents*. Springer, 2014, pp. 97–109.

[113] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 1-3, pp. 66–73, 2010, IEEE.

[114] N. Bredeche, J.-M. Montanier, W. Liu, and A. F. T. Winfield, "Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 18, no. 1, pp. 101–129, 2012, Taylor & Francis.

[115] A. Stranieri, A. E. Turgut, G. Francesca, A. Reina, M. Dorigo, and M. Birattari, "IRIDIA's Arena Tracking System," Université Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2013-013, 2013.

[116] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "SwisTrack - a flexible open source tracking software for multi-agent systems," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 4004–4010.

[117] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer, 2015.

[118] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002, IEEE.

[119] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013, Springer.

[120] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The Flying Machine Arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014, Elsevier.