Error Detection in Swarm Robotics: A Focus on Adaptivity to Dynamic Environments

Hui Keng Lau

PhD

University of York
Department of Computer Science

February 2012

Abstract

This thesis examines the problem of adaptive error detection in swarm robotics. As part of the challenges for the transition of current swarm robotics research into the real world implementation, the ability to differentiate between changes to the behaviour due to faulty components and environmental is important. This is a requirement to ensure that robot swarms deployed are fault-tolerant to internal faults as well as external perturbations. Previous work has investigated this issue from a perspective of a single robot but has largely ignored the aspect of adaptivity to environmental changes. By contrast, this work approaches the problem from a perspective of a collective and explicitly addresses the issue of adaptive detection. A collective self-detection scheme called the CoDe scheme is proposed and developed. This scheme is demonstrated to work in detecting errors in dynamic environments with the use of various classifiers. This approach has potential to be applied for other domains that share similar characteristics to swarm robotics in which adaptivity to dynamic environments is crucial. Motivated by the potential resource limitations in swarm robotic systems, this thesis also investigates other aspects related to minimising resource usage such as reducing the number of false positives and communication overhead.

Contents

List of Tables

List of Figures

List of Abbreviations

ACO - Ant Colony Optimisation
APC - Antigen Presenting Cell
AIS - Artificial Immune Systems
ANN - Artificial Neural Networks
CoDe - Collective Self-Detection Scheme
CST - Constant OPR
DCA - Dendritic Cell Algorithm
EDR - Error Detection and Recovery
ESD - Extreme Studentized Deviate
FMEA - Failure Mode and Effect Analysis
`FPR` - False Positive Rate
GA - Genetic Algorithms
IR - Infrared
KDE - Kernel Density Estimation
MCC - Matthews correlation coefficient
MHC - Major Histocompatibility Complex
NSA - Negative Selection Algorithm
`OpCom` - Optimistic Communication Strategy
`OPR` - Object Replenishing Rate
`PeCom` - Pessimistic Communication Strategy
pMHC - peptide:MHC
pRSD - Percentage Relative Standard Deviation
RDA - Receptor Density Algorithm
RSD - Relative standard deviation
SI - Swarm Intelligence
SOM - Self-Organising Map
SRS - Swarm Robotic Systems
TCR - T-cell Receptor

TDNN - Time-Delay Neural Network
TMR - Triple Modular Redundancy
TPR - True Positive Rate
TSP - Travelling Salesman Problem
V_{ODS} - Varying Object Distribution
V_{OPR} - Varying Object Replenishing Rate

Acknowledgements

To God for the gift of life.
To Jon and Iain for everything during this PhD.
To my family for the support, love, and understanding.
To my friends for the discussions, ideas, food, and companionship.
To the M.O.H.E of Malaysia, UMS, and SEIT for the financial support.

Special thanks to my examiners Dr. Roderich Groß and Dr. Mark Neal for their time, insightful comments, recommendations for improvement, and a nerve-wreaking *viva voce*.

Declaration

The work in this thesis has been carried out by the author between May 2008 and January 2012 at the Department of Computer Science, University of York. Apart from work whose authors are clearly acknowledged, all other work presented in this thesis was carried out by the author. The results of this work have been previously published by the author. A complete list of refereed publications is as follows:

- Lau, H.K, Timmis, J., and Bate, I. (2008). Anomaly Detection Inspired by Immune Network Theory: A Proposal. Congress on Evolutionary Computation 2008 (CEC2008). Trondheim, Norway. IEEE press, pp. 3045-3041.

- Lau, H.K, Bate, I. and Timmis, J. (2009). An Immuno-engineering Approach for Anomaly Detection in Swarm Robotics. International Conference on Artificial Immune Systems (ICARIS2009). Edinburgh, Scotland. Springer. LNCS 5666. pp. 136-150.
  The paper is based on the work in Chapter 4 and Chapter 5.

- Lau, H.K, Bate, I. and Timmis, J. (2011). Collective Monitoring to Achieve Self-Detection of Errors in Swarm Robotics. International Conference on Artificial Immune Systems (ICARIS2011). Cambridge, UK. Springer. LNCS 6826, pp. 254-267.
  This paper is based on the work presented in Chapter 5 and Chapter 6.

- Lau, H.K, Bate, I., Cairns, P. and Timmis, J. (2011). Adaptive Data-Driven Error Detection in Swarm Robotics with Statistical Classifiers. Robotics and Autonomous Systems. **59**(12). pp. 1021-1035. DOI: doi.org/10.1016/j.robot.2011.08.008.
  This paper is a summary of the work presented in Chapters 6, 7, and 8.

CHAPTER

ONE

Introduction

This chapter provides an overview of the research reported in this thesis. Section 1.1 presents the background of the related topics which motivated this work. Then in Section 1.2 the research question is formally stated. To investigate the research question, the research objectives are presented in Section 1.3. This is followed by an overview of the contents of this thesis in Section 1.4.

1.1 Background and Motivation

The need to adapt to changing environments has always been associated with the need for fault tolerance. In many systems, e.g. the web, mobile ad-hoc network, swarm robotics, the operational environment is often dynamic and unpredictable. Thus, the operation of these systems is often also affected by the environment in which they are deployed. In operation, these systems may experience undesirable behaviours, or more precisely anomalies, for a variety of reasons. These can be caused by faults in the system or be due to interactions with the environment [1, 2]. Therefore, the ability to tolerate failures as well as the interference from the environment is a sought-after feature in most systems. In swarm robotics, a robot swarm has always been characterised as having built-in fault tolerance capabilities due to the redundancy of robots in the system [1, 3, 4]. However, work in Winfield and Nembrini [1] has demonstrated that there are cases when redundancy alone is insufficient, in particular with partially failed robots in the system. Therefore, there is a need to provide an additional level of fault tolerance on top of those provided through redundancy. In addition, tolerance to interference from the environment should also be considered.

1.1.1 Swarm Robotics and Fault Tolerance

Swarm robotics is a research field that has become increasingly popular for its potential use in many real-world applications [5, 4, 6], and has benefited from advances in engineering on miniature devices [4]. In general, a swarm robotic system (SRS) consists of a collection of homogeneous robots which interact with each other as well as with the environment to perform various tasks [3, 4]. A robot swarm has the potential to be deployed for a variety of tasks such as environmental monitoring, space and deep-sea explorations, removing land-mines, and search-and-rescue missions [3]. However, to date, most of the existing SRSs have only been built and tested in ideal and fault-free settings [1, 4].

To translate these SRSs for real world applications is challenging and not as straightforward as might be expect [1]. First, any system (even with a given and reasonably well-understood operational environment) is likely to experience undesirable behaviour. These undesirable behaviours can be caused by a variety of factor such as random errors, systematic errors or deliberate sabotage [1]. Therefore, one of the immediate challenges is how to ensure that the SRSs can continue to operate in the event of failures to some individuals within the swarm. Second, the operational condition for SRSs is often dynamic and thus cannot be predicted. Therefore, the SRSs not only have to be able to deal with the failure of individuals, they also need to be able to cope with changing environmental conditions.

1.1.2 An Explicit Approach to Fault Tolerance

Typically, an SRS is implicitly fault-tolerant because of the redundancy of robots in the swarm. However, there are exceptions. First, redundancy is irrelevant (as it is not necessarily required) if a minimum amount of healthy agents for a particular task are still operational [2]. Second, it is assumed that the affected robots do not interfere with or influence the behaviour of healthy robots. However, recent research (e.g. [1, 7]) has demonstrated that failed robots *can* and *will* significantly affect the completion of a task. In particular, a fault to the wheels while other components are still operational causes physical anchoring of the robot swarm in a swarm taxis scenario. Although this observation is specific for swarm taxis (swarm motion toward a beacon), the likelihood for faulty robots interfering with other robots in other tasks can also be high. In situations where faulty robots interfering with the task, having redundant robots alone is insufficient and it is beneficial to equip the SRS with other fault-tolerance measures. Explicit error detection and recovery (EDR) is one such measure, and it is commonly used in engineering fields [8].

There are generally three stages in an EDR: error detection, fault diagnosis and recovery [8]. Error detection detects erroneous behaviour in the system. This is essentially a two-class anomaly detection problem to determine whether the system is currently in a normal state. Fault diagnosis identifies the cause of an error including the nature and the exact location of the faults. When a fault has been identified, recovery measures can then be carried out to prevent the fault from reoccurring. The recovery measure can be as simple as shutting down the failed agents. Error detection is a crucial first step in an EDR as the activation of the other stages (fault diagnosis and recovery) is only initiated when an error is detected.

In swarm robotics, error detection is a challenging task particularly when the behaviour of the robots is also affected by changes in their environment. There are various ways of detecting errors in a swarm robot. A model of how a robot should behave can be built and the actual behaviour is then compared to the predicted behaviour by the model (model-driven). Alternatively, data can be collected during normal operation. On the basis of this data the presence of a fault can be inferred (data-driven). The problems with model-driven error detection is that the development of accurate models is often difficult, if not impossible, because the environment is not static and generally unpredictable [9]. Due to the interactions between robots and the environment, as well as other natural factors, the state of the environment can change and this in turn can affect the behaviour of the robots. For example, an SRS is involved in a foraging task: the state of a robot may be determined by some task-related measures such as the quantity of objects collected. If there is a fault to the wheels, the number of objects collected over a fixed period by that robot will be fewer and likely to deviate significantly from a fault-free condition. However, in this example, the quantity of objects that can be collected is also dependent on the state of the arena such as the number of objects in the arena, and the physical distribution of those objects, which are likely to change as time progresses (time-varying). Therefore, data-driven approaches are preferable.

Existing work on data-driven error detection in swarm robotics includes [10, 11, 12, 13]. These studies approached the problem from the perspective of a single robot [2]. In other words, the classifiers used to detect errors in these studies only operated on data from a single robot. In addition, the aspect of adaptivity to a dynamic environment has been largely ignored. This is rather restrictive as the ability to adapt to the environment has become more relevant for the transition of SRS to the real world.

Alternatively, a potentially more suitable approach would be to address the

error detection problem from the perspective of a collective. A collective in this context would be a collection of interacting robots over some interaction time or space, for example, over a period of time or within a wireless communication range. In a robot swarm, in which the robots are mobile, the size of this collective is dynamic. Therefore, the classifier for this approach would be required to be able to work on data of varying sizes. For this reason, statistical classifiers can be a good candidate. On the other hand, bio-inspired algorithms could also be potentially applicable.

Bio-inspired algorithms have been commonly applied to complex problems including those that deal with adaptivity to dynamic environments [14]. Within the domain of bio-inspired algorithms, Artificial Immune Systems (AIS) have been actively explored for error detection in swarm robotics. AIS algorithms are algorithms inspired by the processes and mechanisms of the vertebrate immune system which exhibit the properties of fault tolerance, learning, memory, self-organisation and robustness [15]. For example, the work in Canham et al. [10] and Mokhtar et al. [13] used AIS algorithms to detect errors of the controller and the sensors. In 2009, a new AIS algorithm called the Receptor Density Algorithm (RDA) was presented. This algorithm was analytically demonstrated to be capable of dynamic anomaly detection and shown to be connected to kernel density estimation, which is a powerful non-parametric statistical technique [16].

1.2 Research Question

This aim of this thesis is to investigate the problem of error detection in swarm robotics with a focus on adaptivity to the dynamic environment. Rather than addressing this problem from the perspective of a single robot, this research will approach it from the perspective of a collective. Therefore, the research problem is defined as follows:

> *Can data-driven error detection from the perspective of a collective be used to provide adaptive detection for robot swarms deployed in time-varying environments?*

1.3 Research Objectives

As stated in Section 1.1.2, this research will investigate the research question in the context of statistical classifiers and an immune-inspired classifier, in particular the RDA classifier.

Therefore, the research objectives of the experimental work in this thesis are:

- **RO1:** To establish an experimental testbed to investigate the nature of errors and the aspect of adaptive error detection in swarm robotics.
- **RO2:** To investigate the feasibility of addressing error detection in swarm robotics from the perspective of a collective, and to propose a corresponding detection scheme to incorporate the aspect of adaptivity to dynamic environments.
- **RO3:** To examine the application of statistical classifiers in the context of the proposed scheme for adaptive error detection.
- **RO4:** To examine the application of an immune-inspired algorithm in the context of the proposed scheme for adaptive error detection.

1.4 Thesis Structure

The remainder of the thesis is organised as follows:

Chapter 2 provides a brief historical account of swarm robotics within the swarm intelligence paradigm, from the biological foundations of the natural swarm systems to the inspirations that have led to the development of artificial swarm systems. To provide a better overview of swarm robotics, the common research topics and challenges are also included in this chapter.

One of the challenges in swarm robotics is to provide fault tolerance in robot swarms. Therefore, Chapter 3 reviews related work on fault tolerance in swarm robotics. It covers the general approaches to fault tolerance in systems and whether these approaches are relevant in swarm robotics. Recent work on error detection in swarm robotics is reviewed from the perspective of the detection target, that is, whether an observer is able to detect an error on itself (endogenous), or errors on other robots (exogenous).

Chapter 4 presents the work for **RO1**. It covers the experimental framework including the identification of robot foraging as the testbed, the design of the controller for the robots, the fault models of the wheels, the models of the time-varying environment, and the simulation platform.

As part of the work for **RO1**, Chapter 5 presents experimental analyses of the robot foraging to provide evidence to support assumptions made regarding the state of the robot swarm as well as the plausibility of inferring the existence of a fault from the operational data.

In Chapter 6, work on **RO2** and **RO3** is presented. For the first part of this chapter, illustrative examples are given of the difficulty in achieving adaptive error detection from the perspective of a single robot. Therefore, a new detection scheme is proposed to address the same problem but from the perspective of a

collective. Then, the proposed scheme is tested with the use of four conventional statistical classifiers in the second part of the chapter.

Chapter 7 presents experimental work for **RO4** with the use of an immune-inspired algorithm called the Receptor Density Algorithm (RDA) [16]. In this chapter, the RDA classifier is introduced and an example is given on applying this classifier in the context of the CoDe scheme. The RDA classifier was tested with the same data as the conventional statistical techniques described in Chapter 6, and the performance compared. With the RDA classifier, this chapter also further examines the potential of reducing the number of false positives by increasing the detection window as introduced in Christensen et al. [12].

In Chapter 8, two communication strategies are proposed in the context of the scheme proposed in Chapter 6. The aim of these strategies is to reduce the overhead of communication involved without a significant impact on the performance of detection.

Chapter 9 concludes the thesis. A summary of work presented in this thesis and its limitations, as well as suggestions for future work, are presented.

Part I

Background and Related Work

CHAPTER

TWO

From Swarm Intelligence to Swarm Robotics

This chapter introduces the concept of swarm intelligence with a focus on swarm robotics. In Section 2.1, an historical account of the swarm intelligence paradigm is given together with examples of natural swarm systems and the relevant characteristics that have been inspiring work on artificial swarm systems. Then, Section 2.2 presents a more focused discussion on an instance of artificial swarm systems namely swarm robotics. It covers the characteristics of a swarm robotic system, common research topics in swarm robotics, and related challenges for the realisation of swarm robotic systems in the real world.

2.1 SI: Swarm Intelligence

The expression 'swarm intelligence' was initially used by Beni and Wang [17] to describe their work on cellular robotic systems, in which a collection of simple and autonomous robots operated in a n-dimensional space to produce ordered patterns through nearest neighbour (local) interactions. The term 'swarm' was chosen to reflect the fact that the group of cellular robots that they were dealing with was not just a typical 'group'. Instead it was a group with special characteristics such as those found in swarms of insects, for example, decentralised control, lack of synchronicity, simpleness, and identical members [18]. Beni and Wang [17]'s work laid the foundations for what later became a scientific discipline with scientists from various disciplines (e.g. biologists, roboticists, and computer

scientists) working together on swarm intelligence systems. With continuously growing work in both natural and artificial swarm intelligence systems, swarm intelligence (SI) has developed into an established discipline.

Recently, Dorigo and Birattari [19] defined SI as:

> *... the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralised control and self-organisation. In particular, the discipline focuses on the collective behaviours that result from the local interactions of the individuals with each other and with their environment.*

This definition of SI encapsulates both natural and artificial swarm systems. In natural swarm systems, SI research aims to further develop the understanding of the underlying mechanisms in natural swarm systems that result in complex and interesting collective behaviours. Research into artificial SI then exploits the knowledge gained from the natural SI research and applies that knowledge to solving analogous engineering problems. The work in artificial SI research encompasses the modelling of the collective behaviour of natural swarms to identify, abstract and exploit the underlying computational principles, and the application of those mechanisms to artificial swarm systems ranging from optimisation to multi-robot systems [20].

Dorigo and Birattari [19] further pointed out that a typical SI system has the following properties:

- ***many individuals***: there are many individuals in the swarm. In Şahin [3], a group size of 10 - 20 is considered acceptable. However, as pointed out in Bayindir and Şahin [5], the exact number is not important because scalability in size is one of the desired characteristics of these systems;
- ***relative homogeneity***: the individuals in a swarm are physically the same (identical) or there are only a few types of individuals;
- ***local interactions***: individuals in a swarm only interact with others in a close proximity utilising only information available locally. The interactions can be direct or indirect (stigmergy); and
- ***self-organisation***: the overall behaviour emerges in response to the local environment and to local interactions.

2.1.1 Natural SI Systems

Initially, social insects such as ant and bee colonies have been the predominant SI systems in nature which have inspired work in artificial SI systems. For instance,

in Beni and Wang [17], as later explained by Beni [18], their cellular robotic systems were designed to encapsulate the characteristics found in swarms of insects. Social insect colonies, ranging from a few to millions of individuals, display the ability to continue to carry out tasks efficiently even when faced with lost individuals and environmental challenges. These tasks include the building of structures such as nests, the finding and retrieving of food (foraging), and navigating around obstacles.

For example, the army ant *Eciton burchelli* can carry out large scale foraging (up to 200 000 foragers) in a short period of time (from dawn to dusk) by forming and following traffic lanes between the food sources and the nest [21]. The ants lay pheromone on the ground when they are on their way back to nest with prey. Other ants, sensing the pheromone, will follow the pheromone trails and at the same time also lay their own pheromone. Over time, traffic lanes are formed. Whilst following the trails, the ants have directional preference to turn to the left when in a course of collision with other ants. This preference, in time, results in the emergence of three-lane trails with returning ants occupying the central lane and outbound ants occupy the periphery lanes [21].

In the construction of a nest, the termites *Macrotermes bellicosus* can build mounds up to 6m in height and 30m in diameter [22]. It has been shown that the building of nest is achieved by responding to the cues from the local environment such as shape of the local configuration and current progress. For example, in building columns and arches, pheromones are laid on the balls of mud used for construction. New mud balls are placed near other mud balls with pheromones. As the construction proceeds, the columns gets taller and the pheromones near the bottom evaporate. If two columns are built near to each other, as the columns getting taller and closer to each other the concentration of pheromones at the top of the columns will cause the two columns to be joined together to form an arch [22].

Figure 2.1(a) shows an example of Weaver ants pulling leaves together to build their nests by forming bridges of workers. The ants form a chain (bridge) with their own bodies to allow other ants to move between the leaves. Then the leaves are pulled together. The leaves are glued together using the silk produced by the ants. The resulting collective behaviours of these social insects are interesting because they do not appear to be coordinated by a specific group of leaders or queens but rather are self-organised in response to interactions between the individuals and the environment [23].

Other natural SI systems includes flocks of birds [24, 25] (see Figure 2.1(b)), schools of fish [26, 27], herds of sheep [28] and bacterial colonies [29, 30]. Birds

(a) (b)

Figure 2.1: (a) Weaver ants collectively form a bridge[a]; (b) Flock of birds[b].
[a]www.physorg.com/news11060.html, last accessed: 14/03/2012. Image reprinted with permission from PhysOrg.com.
[b]http://en.wikipedia.org/wiki/File:Auklet_flock_Shumagins_1986.jpg, last accessed: 14/03/2012. Image from Wikimedia Commons free for reuse.

in a group (up to thousands) often fly together and are able to move freely as a collective and *execute abrupt manoeuvre with precise coordination* [25]. It is suggested that the manoeuvres can be initiated by any birds but the resulting coordination in movement is achieved through visual communication [25]. The parameters of flight in the birds' coordination that may contribute to the flocking behaviour includes the turning angle, spacing between birds, velocity of flight, direction of flight, time to takeoff and landing [24]. In addition, birds of different types also produce different flight formations. For example, column formation (one after another along a flight path) can be seen in the *Brown Pelicans* following a shore line whereas V formations (approximately the same number of birds in the left and right in V shape, following a leader) are often seen in *waterfowl* [24]. The benefits offered by flocking in birds include protection against predators, mating, locating food sources, warmth [25].

Fish schools, for example hundreds of *silversides*, that initially move along a straight course suddenly disperse as predators (such as *barracuda*) approach and regroup moments later [26]. There is no leader in the group and each fish adjusts its speed and heading relative to its neighbours, with the nearest neighbours having a greater influence [26]. The eyes and lateral lines on fish are sensitive to displacement of water. By utilising these two organs, each fish can match the speed and direction of the neighbours. How does each fish respond to its sensory inputs, positions of neighbours and switching of behaviour with different sizes of a group to achieve the observed collective motion are of great interest to researchers [27].

In bacterial colonies, patterns emerge as a respond of the colonies to the environmental conditions. For example, the bacteria *Paenibacillus dendritiformis* produces different patterns under different conditions of nutrients, surface, or both [29, 30]. The geometrical patterns are observed to be inheritable and can be produced by different bacteria strains.

It has been observed that these natural SI systems share many common characteristics. In relation to the collective behaviours that they exhibit, these natural SI systems typically consist of a population of 'simple' agents which interact locally with each other and the environment. Here, the word 'simple' is used loosely to mean that these agents carry out their tasks by following a relatively simple set of rules. From these local interactions, which are observed to be decentralised, collective behaviours can emerge. Underlying this 'organisation without an organiser', researchers have identified several core mechanisms that contribute to the observed collective behaviours [31].

2.1.2 Principal Concepts in SI

Garnier et al. [31] pointed out that there are several hidden mechanisms that allow insect societies to cope with uncertain environments and to find solutions to complex problems. These mechanisms are major concepts in SI to *explain* the interactions in insect societies that lead to these complex collective behaviours. They are *decentralisation*, *stigmergy*, *self-organisation*, *emergence*, *positive and negative feedback*, *fluctuations*, and *bifurcations*.

Early research into understanding the underlying mechanisms for collective behaviours in social insects hypothesised that individual insects possess a representation of the global structure for appropriate decision making or that the queen supervises the work done by the workers, that is, centralised decision making. However, later research has shown it to be otherwise. Individual insects do not need any representation or knowledge of the global structure which they produce and there is no supervisor in the colonies [32]. Rather, a social insect colony is decentralised with autonomous units distributed in the environment which respond directly to stimuli from the local environment [33]. Because of this decentralisation and direct response to stimuli from the local environment, the insects are able to respond to external challenges such as environmental changes, lost individual and the addition of individuals [31].

The notion of stigmergy was introduced by Grassé [22] to explain the building activities carried out by termites of the *Macrotermes bellicosus*. He showed that coordination over the nest reconstruction amongst the termites is achieved through

indirect interactions (stigmergy) and mainly attributable to the condition of the local environment, such as the shape of the nest and the progress of the work. In other words, the activity of individuals is guided by the information perceived from the local environment and not through direct interactions between the individuals. Every time a worker performs a building action, the shape of the nest changes and this in turn influences the actions of other workers. This process leads to the coordination of the collective work which can make it seems that the colony is following a well-defined plan [31].

Self-organisation has been identified as a major component of a wide range of collective behaviours in social insects from the thermo-regulation of bee swarms to the construction of nests by ants [34]. It is a set of mechanisms that can result in the appearance of structures at the global level. This observation emerges due to interactions among its lower-level components instead of being explicitly coded at the individual level. Self-organisation emerges from the interplay of positive feedback, negative feedback, random behaviour (fluctuations), and multiple direct or stigmergic interactions [23, 31].

Positive feedback promotes the creation of structures and amplifies the fluctuations in the system. For example, the recruitment of ants to a food source by means of pheromone trails is a kind of positive feedback because the reinforced pheromone trail creates conditions which lead to the emergence of a trail network at the global level.

Conversely, negative feedback counterbalances positive feedback and this interaction leads to a stable collective state. In trail recruitment in ants, negative feedback on the formation of a trail can come from the limited number of foragers, exhaustion of a food source, evaporation of the pheromones creating the trail, or competition between trails to attract foragers [31].

Social insects are observed to perform actions that can be described as stochastic (random fluctuations) [31]. However, such randomness plays an important role in exploration and enables the colony to discover new paths and food sources.

Finally, self-organisation allows systems to reach different stable states (bifurcation) when some of a system's parameters change depending on the initial conditions and the random fluctuations [31].

2.1.3 From Natural to Artificial SI Systems

From a better understanding of the underlying mechanisms and collective behaviours in natural swarm systems, scientists have been inspired to develop and apply swarm-inspired solutions to solve analogous engineering problems. For

example, understanding the foraging behaviour of ant colonies in finding shortest paths from their nests to food sources inspired the development of Ant Colony Optimisation algorithms [35, 36]. Dorigo et al. [35] introduced the first ACO algorithm - the Ant System. A population of artificial ants was created and the ants traverse along a construction graph with a set of vertices and a set of edges. The ants move from vertex to vertex along the edges (with different lengths) of the graph and lays artificial pheromone trails on the edges. The probability of choosing a particular edge is influenced by the concentration of the pheromone trails which evaporate as time progresses. Edges with a higher pheromone concentration represent shorter paths. In Dorigo et al. [35], the Ant System was successfully applied to a Travelling Salesman Problem (TSP) with the vertices of the graph representing the cities and the edges representing the distances between the cities. The task in TSP is to find the shortest journey that allows a salesman to visit each city only once.

Since then, many similar ant-inspired algorithms have been introduced and with the increasing volume of research, the body of ant algorithms for optimisation problems has subsequently been referred to as Ant Colony Optimisation (ACO) algorithms [36]. ACO has been successfully applied to a variety of problems including scheduling, vehicle routing, assignment problems, set problems, data mining and information retrieval. A website with information related to ACO is at http://www.aco-metaheuristic.org/.

Particle Swarm Optimisation (PSO) is another notable algorithm inspired by bird flocking and fish schooling [37]. PSO was developed by Kennedy and Eberhart [37] through their work on simulating a large number of birds flocking synchronously in search of a food source. In bird flocking, the movement of the flock appears to be synchronous and the synchrony is influenced by the distance between each bird and its neighbours. When searching for food, individual members can benefit from the discoveries and previous experience of all other members. This same behaviour was also observed in fish schooling. This social sharing of information is fundamental to the development of PSO [37]. In PSO, a population (a swarm) of candidate solutions (particles) is modelled in a search space and each particle has a position and a velocity. The movements of the particles are based on their own best position as well as the swarm's best-known position. Since its introduction, PSO has been applied to a variety of tasks including antenna design [38], transmission network planning, and the network reconfiguration and expansion problem [39].

Reynolds [40] created a well-known simulation called Boids which was inspired by the flocking behaviour in birds (see Figure 2.1(b)). In bird flocking,

each bird seems to be flying independently without any knowledge of the global shape of the flocks as a whole. However, to the observer the entire shape of the flock, it seems to appear otherwise. This interesting flocking behaviour in birds is also seen in fish [27] and sheep [28]. In Boids [40], the flocking behaviour has been simulated using three simple rules of *separation* (steer to avoid crowding local flockmates), *alignment* (steer towards the average heading of local flockmates), and *cohesion* (steer to move toward the average position of local flockmates). The same flocking behaviour has also inspired the work of coordinated movement in multi-robot systems.

Another form of artificial SI system is the multi-robot systems which can be considered as an artificial instantiation of natural SI systems, albeit much simpler and for a different purpose. For instance, in Beni and Wang [17], the cellular robotic systems were designed to encapsulate the characteristics found in swarms of insects such as decentralised control, simplicity and identical members. In multi-robot systems, physical or logical robots in simulation were developed to carry out tasks and simulate behaviours that are analogous with natural SI systems such as foraging, cooperative transportation and aggregation. The motivation for multi-robot systems lies in the potential for real-world applications such as search-and-rescue missions, environmental monitoring and space exploration [3].

Under the umbrella term of multi-robot systems, swarm robotics is a relatively recent research topic with growing research interests, and it is the platform on which the work in this thesis is based. Thus, the next section will provide an overview of swarm robotics and the motivations for the work.

2.2 Swarm Robotics - An Instance of Artificial SI Systems

2.2.1 An Overview

Over the recent years, various expressions have been used to describe a group of simple physical robots such as cellular robotics [17], collective robotics [41], and distributed robotics. Due to the strong biological background, the term 'swarm robotics' has also recently been used. As pointed out by Şahin [3], these terms are often vague and have overlapping meanings. Therefore, to ground the discussion on swarm robotics to a single definition, this work adopts the definition of Şahin [3] who defined swarm robotics as:

> *the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment.*

This definition encapsulates the key properties of a typical SI system as pointed out by Dorigo and Birattari [19], and also applies to other multi-robot systems. However, Şahin [3] further emphasised a set of criteria which can be used to evaluate the degree to which swarm robotics might apply and how it might be different from other multi-robot systems (as there is some overlap):

- ***autonomous robots***: to be called autonomous robots, the robots should have a physical embodiment, be it in a physical world or in a simulated world, in order for physical interactions with the environment to occur;
- ***large numbers of robots***: although it is hard to justify a lower bound for a group to be called a swarm, it is common to accept group sizes of 10-20 robots [3]. However, studies which are carried out with smaller group sizes must be able to be scaled up to a larger group when required;
- ***homogeneity***: robots should be identical, at least at the level of interactions. Heterogeneous multi-robot systems fall outside the swarm robotics approach and these systems are often referred to as collective robotics;
- ***simplicity***: robots should be simple in capabilities relative to the task, not necessary hardware or software complexity; and
- ***local interaction***: individuals should have local sensing and communication capabilities, either direct or indirect.

These criteria influence on the work discussed in later chapters and justify the reasoning behind the approaches taken, and will be reflected as the need arises. Note that these criteria should be treated as a reference and not as a *checklist* [6].

One of the main motivations for the work in artificial SI systems, in this case swarm robotics, is the potential for it to be applied to solve complex and difficult engineering problems that are analogous to their biological counterparts. Based on the properties of the engineering tasks, swarm robotics would be most applicable for a number of task domains such as tasks with a large coverage area, tasks that are too dangerous for human operators, tasks with dynamic scales, and tasks that require redundancy [3].

Although SRS can be potentially applied for the application domains presented in Şahin [3], at present, most of the SRS are still laboratory-based [1]. As pointed out by Winfield and Nembrini [1], one of the reasons for this is that there is still a lot of work to be done to ensure that the physical realisation of the SRS

are safe and dependable. This is one of the challenges in swarm robotics and will be discussed in greater detail in Section 2.2.3.

From the definition of swarm robotics and its potential applications, it might be apparent which research topics swarm robotics researchers are generally involved in. From a more recent survey by Bayindir and Şahin [5], swarm robotics research covers many aspects from hardware design, software design, the modelling of collective behaviour (macroscopic modelling), individual behaviour (microscopic modelling) and communication to swarm tasks. The next section will review some of the main research topics in Bayindir and Şahin [5].

2.2.2 Swarm Robotics Research

To give a general overview on the research topics in swarm robotics, the following subsections briefly review three of the research topics identified in Bayindir and Şahin [5]: modelling, behaviour design, and swarm task.

Modelling

Modelling is important in swarm robotics research. Modelling allows experimentation in simulation which reduces the risks of physical damage to the robots as well as is not constrained by the limited power of the robots. For experiments which involve a large number of robots, modelling can be carried out easily and cheaply in simulations. At present, the number of physical robots used is generally around tens of robots due to the cost of individual robots. Experiments using simulations (e.g. computational modelling) allow the testing of concepts and hypotheses, large numbers of repeated runs, explorations, and optimisation of parameters which can be carried out cheaply and in a shorter time frame [42, 43]. Also, simulations allow the study of systems prior to construction.

Despite the advantages of modelling, there will always be differences between results from simulations and those from actual physical experiments. However, insights gained from the simulations allow a more principled design and hence reduce the risk associated with premature deployment in real robots [5].

Depending on the approach and the components to be modelled, the modelling approaches for swarm robotics can be divided into sensor-based, microscopic and macroscopic modelling. Sensor-based modelling places models of sensors, actuators and objects as the main components. Then, interactions between the robots and the environment are added. There are two main approaches for sensor-based modelling: non-physical simulations and physical simulations [5]. In non-physical simulations, the physical properties of the robots and the ob-

jects in the environment such as mass and motor force are ignored. However, the interactions are preserved by adding logic to cater for collisions between objects. In physical simulations, off-the-shelf physics engines are integrated into the simulation. For example, in Trianni et al. [44], the robots (called s-bots) are modelled in simulation using off-the-shelf physics engine Vortex™ from CMLabs[1] to reproduce the dynamics, friction and collisions between physical bodies. This adds much more complexity but is more realistic in relation to real-world scenarios.

On the other hand, microscopic and macroscopic modelling uses mathematical models to represent the interactions in the system without explicit modelling of robotic components. Therefore, they are much easier to implement and faster to simulate. The behaviour of robots is defined as states and the transition between states is determined probabilistically. For example, Martinoli et al. [45, 46, 47] used microscopic probabilistic models to study collective clustering. The activities of robots were represented as a sequence of probabilistic events that are triggered according to geometrical considerations (e.g. size of arena, robots, objects; robot-robot and robot-environment interactions) and sensory capacity of the robots. The work was later applied to study collaborative stick-pulling in Ijspeert et al. [48]. In Liu [42] macroscopic probabilistic models in the form of differential equations were used to study the effect of various parameters for a more efficient adaptation algorithm in foraging.

The difference between macroscopic and microscopic modelling is the granularity of the models developed. Whilst macroscopic approaches model the behaviour of the system as a whole, microscopic approaches model the behaviour of each robot [5, 42]. Therefore, the use of macroscopic or microscopic models is dependent on the objective and focus of the experiment, whether on the overall behaviour of the system or the behaviour of individuals in the system.

Behavioural Design for Swarm Robots

The second category of swarm robotics research is the behavioural design of the robot controllers. The design of robot controllers can be traced back to earlier work on robotics in general. However, the design of robot controllers in swarm robotics is more geared towards the accomplishment of tasks collectively by a group of individual robots. Two of the common approaches to designing robot controllers in swarm robotics are the subsumption architecture [49] and artificial neural networks (ANN).

The subsumption architecture [49] is one of the best-known controller design

[1] www.vxsim.com/en/software/index.php

architectures in behaviour-based robotics. This architecture involves the incremental development of behaviours by considering each behaviour as a separate module. The coordination of behaviours is achieved through a suppression mechanism between the behaviours. Subsumption-based controllers are manually programmed.

Robot controllers designed with ANN are often referred to as neural controllers. A neural controller is often a simple multi-layer perceptron in which the input from the sensors is connected to the input layer and the output layer is connected to the actuators of a robot. One of the challenges in the design of a neural controller lies in the setting of the connection weights between neurons of different layers. There are many approaches in which the weights can be set: hard-coded, trained with back-propagation, or evolved using genetic algorithms (GA). The problem with manual hard-coded connection is that the weights are brittle and may not be exploring aspects of the environment that a learning system might exploit. That is, they may not deal with a dynamic time-varying environment or an environment for which there is no prior knowledge. Another way to obtain the connection weights is to use GA to evolve them. For example, in [50], an ANN with connection weights evolved using a specific version of GA was used to achieve cooperative transport by two robots. The ANN in [50] consisted of an input layer of five neurons connected to various sensors, a hidden layer, and a output layer with three neurons connected to the wheels and grippers.

The choice of which approach to use in designing the robot controller is strictly a matter of preference. For a simple task such as non-cooperative foraging in which the underlying mechanisms are known and well-understood, the controller can be easily designed and implemented using the subsumption architecture. However, for more complex tasks involving cooperation between many individuals and in which the underlying mechanisms are not properly understood, evolving the neural controller allows the exploration of novel strategies.

Swarm Task

Finally, the general problems or tasks addressed in swarm robotics are particularly important because they help to establish the practical value of swarm robotics and help to divide the real problems into manageable sub-problems [5]. Examples of tasks generally studied in swarm robotics include pattern formation, aggregation, coordinated movement or flocking, cooperative transport, and foraging.

Pattern formation can be defined as the emergence of global patterns, which are usually predefined, from local interactions between the agents [5]. It is useful for many applications such as the formation of structures to navigate through obstacles as well as obstructing other objects from passing through. The main challenge in this problem is the coordination of a group of robots to form the desired global patterns with only local interactions.

Fredslund and Matarić [51] developed an algorithm for the formation of various geometric shapes including diamond, column and line. Each robot in the system has a unique ID which is broadcast regularly as a heart-beat message and other robots can detect this ID. To form the predefined shapes, each robot positions itself relative to one designated neighbour and maintains its place in the formation by keeping that neighbour in the centre of the sensor's field of view. One robot is designated as the conductor which decides the type of shape to form. For example, for a centred formation, the conductor broadcasts the information regarding the type of centrered formation together with its ID. Other robots move into positions by locating their designated neighbour and maintaining that position.

Aggregation is a crucial task in many biological systems to help organisms to avoid predators, resist hostile environmental condition and find mates [52, 44]. The self-organised aggregation that occurs in natural swarm systems is achieved through the interplay of positive and negative feedback [44]. In swarm robotics, the aggregation problem is generally defined as the task of aggregating randomly placed robots in an environment. This is an important task as it enables the creation of functional groups of individuals which is the basis of the emergence of various forms of cooperation [44].

Trianni et al. [44] evolved a neural controller for aggregation behaviour using a GA. They observed two types of controllers for aggregation behaviours (see Figure 2.2): static and dynamic clustering. In static clustering, the resulting cluster is very compact and stable and the robots within it do not change their relative positions. On the other hand, the dynamic clustering behaviour creates a group which is loose but allows the group to move together and is observed to be more scalable when more robots are added to the system.

In swarm robotics, the coordinated movement problem requires the robots to move as a group whilst maintaining a global pattern [5]. Strategies to achieve coordinated movements in swarm robotics have been inspired by the analogous behaviour in bird flocking and fish schooling. Coordinated movement is important in swarm robotics as it supports behaviours such as collective navigation through obstacles [53] and swarm taxis [54, 55].

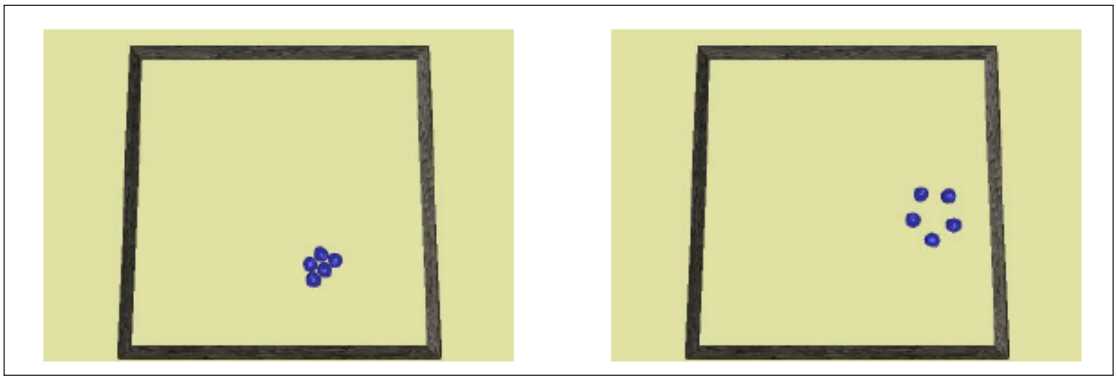


Figure 2.2: Two evolved controllers for swarm aggregation in [44]. The left-hand figure shows static clustering in which the robots form a compact cluster and do not change their relative positions. In the right-hand figure, the cluster is looser but the robots can still move together as a group. Reprinted with kind permission from Springer Copyright Clearance Center: Lecture Notes in Artificial Intelligence [44], copyright (2003).

Nembrini et al. [54] present a distributed algorithm that successfully coordinate the movement of a group of robots towards a beacon (swarm taxis) using only range-limited communication capabilities (Figure 2.3). In their work, each robot maintains a list of radio-connected neighbours. If a robot lost a connection to another robot R_L, it will query its neighbours to determine how many neighbours are still connected to R_L. If the number of neighbours still connected to R_L is less than or equal to a predefined threshold, the robot makes a turn in the opposite direction to its current heading (a 180° turn). If the number of neighbours is greater than the threshold, the robot chooses a new heading at random. This work was later extended in Bjerknes et al. [55] in which the robots do not need to have a representation of the connectivity of their neighbours but are still able to achieve swarm taxis. Note that this is a communication-free approach to achieve coordination of movements.

Cooperative transport involves the coordination of a group of robots to move a heavy object that cannot be accomplished by a single robot alone. It forms the basis for collective foraging that is commonly observed in ants. A box pushing task is a benchmark problem commonly used to study collective transport. In general, a fixed group of tightly coupled robots is involved and the coordination to accomplish the task is achieved either by explicit communication or follows a predefined formation. For example, Matarić et al. [56] used two robots, one on the left and the other on the right, to push a box towards a goal position. The coordination of the pushing behaviour was carried out using a message token between the robots. Kube and Zhang [41, 57] and Kube and Bonabeau [58] have

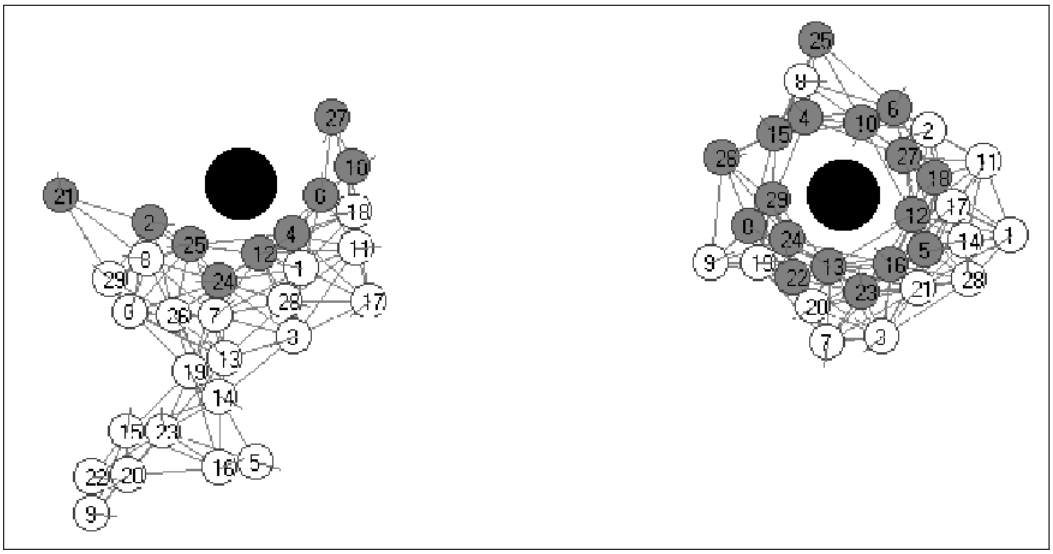


Figure 2.3: Swarm taxis in [54]. The black circle in the centre is the beacon. Numbered circles are robots and the lines are the connections between the robots. Grey robots are illuminated by the beacon whilst white robots are not. In the left-hand figure, the swarm is in the process of encapsulating the beacon. In the right-hand figure, the beacon is completely surrounded. Reprinted with kind permission from ACM: ICSAB [54], copyright (2002).

shown that without explicit communication between robots, cooperative transport can also be achieved. In Kube and Zhang [41], five robots were used for a box-pushing task with no communication involved. Instead, the robots move by following two rules: *avoid interfering other robots*, and *work towards a common task whilst observing the first rule*. This work is later extended in Kube and Zhang [57], Kube and Bonabeau [58] to utilising *cues* such as whether *a box is detected by the sensors or whether a robot is in contact with a box* (generated by concatenating input from the sensors) to achieve cooperative transport.

Recent work on cooperative transport by Groß ([50, 59, 60]) within the swarm-bots project [61] has demonstrated a group of robots that can not only act cooperatively to transport heavy objects of various shapes and sizes but also are able to self-assemble into a structure to carry out the task. The work in [50] involved cooperation between two robots (called s-bots) to self-assemble and then move an object as far away as possible from an initial position. It was then extended to involve up to sixteen robots to pull objects of varying shapes and sizes towards a target location. The neural controller was synthesised using GA and the results demonstrate that although the controller was evolved for a relatively small group, it can also be applied to larger groups.

Foraging is another of the most widely-studied tasks in swarm robotics be-

cause of its strong biological basis, reasonably well-understood underlying mechanisms, and potential for real-world applications such as cleaning, harvesting, search-and-rescue, land-mine clearance or planetary exploration [62, 63, 64]. In general, robot foraging involves the task of locating target objects, collecting the objects and transporting them back to a predefined location called the base. To achieve foraging, the robots must be equipped with the ability to recognise the target objects, avoid obstacles and navigate back to the base. The object can be handled by a single robot, given enough time. However, having more robots allows the task to be accomplished much more quickly [65, 62, 63, 64]. Also, for some task, introducing communication either explicit or implicit can significantly improve the performance of foraging [62, 64]. For objects that cannot be handled by a single robot, cooperation is required and this is generally addressed as a cooperative transport problem.

Work in robot foraging is related to many of the topics mentioned in previous paragraphs from modelling, behavioural design, up to cooperative transport. Winfield [63] presented a robot foraging taxonomy on various topics addressed in swarm robotics research dealing with foraging behaviour.

On top of the work in modelling, behavioural design and swarm tasks, there is another important aspect of swarm robotics research: the development of the physical robots for swarm robotics research. These robots vary in terms of size, number and type. The type of physical robots includes both homogenous robots [61, 66, 67] and heterogeneous robots [68, 69]. Similarly, the size of the robots varies from tens of centimetres [61, 69] to a few millimetres [68, 67]. In addition, the target number of robots involved to form robotic systems ranges from a few to tens of robots [61, 69] to hundreds of robots [68, 66].

Although the physical characteristics of these robots are different, the research topics which they address are similar. Generally, the aims are to equip the robots with functionalities to self-assemble into a structure to perform tasks beyond the capability of a single robot, and to adapt to dynamic environments. These robot swarms were developed with real-world applications in mind. The potential applications for swarm-bots include semi-automatic space exploration, and search-and-rescue. I-Swarm [68] is a development more targeted for future medical applications and micro-scale assembly whilst the self-assembly organism in the Symbrion project [69] can be deployed for monitoring in hard-to-reach locations, rough terrains and potentially hazardous environments.

2.2.3 Immediate Challenges

There are a number of challenges in swarm robotics related to the physical realisation of swarm robotic systems in the real world in addition to those in Section 2.2.2. These challenges include the issue of battery life for long-term deployment, restricted mobility for real world terrain, and the transition from laboratory work to real-world implementation.

Energy Autonomy

One of the most common constraints for physical robots is battery power. For battery-powered robots, there is always a concern about how long the battery can last. This is important because it influences the tasks that can be carried out by the robots and the need to ensure that SRS are only assigned tasks that can be completed before the batteries are exhausted.

With current technologies, most of the batteries used (generally Lithium-ION batteries) for robots in swarm robotics research last for around a few hours. For example, for continuous operation, a battery can last for around two hours for s-bots [70], and three hours[2] for e-puck robots [71, 72]

To address this problem, there have been suggestions for alternative power sources such as solar power in the I-Swarm project [68], wireless transfer, electromagnetic wave transfer heat-powering and kinetic energy conversion. However, for these technologies there are also associated problems such as interference from the environment. For example, for wireless transfer there are issues with the range of transmission, how much power can be transferred and also interference from the robot's own electronics.

Alternatively, batteries with a larger capacity can be used. However, having batteries with a larger capacity also means that more space is required to house the batteries and this results in a heavier and bigger robot. This in turn will require more power to move the robot. Clearly, a compromise is needed between the expected battery life and the size or weight of the robot.

The limitation in battery power also constraint the types of tasks that can be carried out by the robots. Therefore, much of the research on these tasks is conducted in simulations.

Alternatively, there is on-going work on power sharing as the robots self-assemble to form a structure. For example, in the joined Symbrion and Replicator project [69] power management for energy sharing is one of the topics addressed.

[2]from personal communication with Jenny Owen who works with e-puck robots in York's Robotics Lab

There are three types of robots (Scout, Backbone, and Active wheels) as shown in Figure 2.4(b) which can be joined together to form a robot organism with a central nervous system, common energy resources and homoeostasis to adapt to dynamic and open environments. Figure 2.4(a) is an exemplar organism. The target robot organisms are expected to be able to self-configure, self-heal, self-optimise, and self-protect.

(a) (b)

Figure 2.4: (a) An exemplary organism in a simulation. (b) Three types of robot in the Symbrion project. Those in the top left corner are the Scout robots, the white cube shaped robots across the middle are the Backbone robots, and the two at the bottom right corner are called Active wheels
[a]Image from http://symbrion.org/tiki-index.php?page=Simulator. Reused with permission under the copyright of Symbrion project [69].
[b]Image courtesy of Lachlan Murray from Electronics Department, University of York.

The Scout robots are equipped with long-range sensors and specialise in fast and flexible locomotion for inspecting the environment and for gathering robots for assembly. The Backbone robots serve as the backbone of the organism and enable the lifting of docked robots to perform 3D motion. Finally, the Active wheels serve as a tool module for extra support for lifting and for omni-directional movement, and as an additional energy source [73].

Mobility and Sensory Activity

The mobility of robots is another issue which can affect the implementation of SRS in the real world. At present, most of the robots in swarm robotics research are equipped with tracks, wheels or some variation of legs to enable them to move around. For example, for s-bots, mobility is achieved by treels (a combination of tracks and wheels) on both sides [70]. For e-puck robots, there are two wheels, one on each side. Although these mobility devices allow the robots to navigate in many terrains and in particular those in the laboratory environment, they are

far from sufficient for the landscapes in the real world. This in turn constrains the particular terrain types in which the robots can navigate freely and the tasks that a robot swarm can execute.

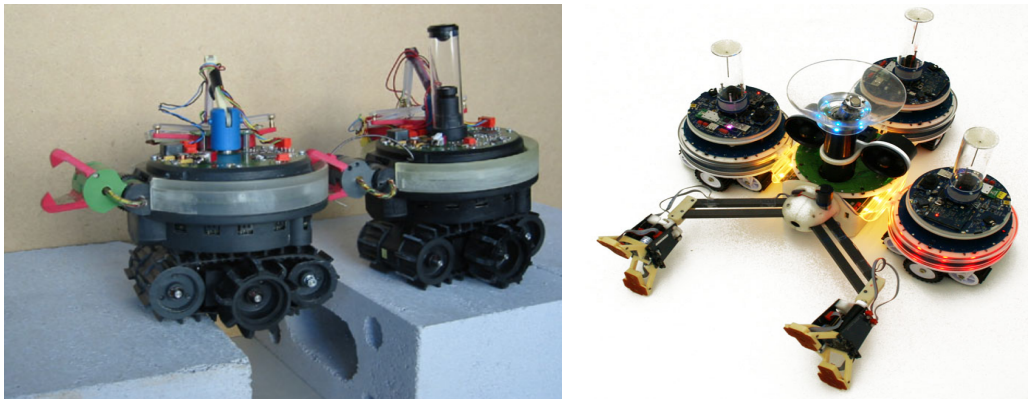
In nature, there is a wide range of mobility devices evolved specifically to suit the particular environment in which the organisms live. The range of movements that are possible in nature include slithering, crawling, walking, running, jumping and climbing. However, some of these movements and mobility devices cannot be adapted to robots at this point in time because of technological constraints. However, there is an interesting new locomotion technique for robots which is worth mentioning. It is called *jumpgliding* [74], which is a hybrid of jumping and gliding with rigid or folding wings. This new locomotion technique was developed by researchers from EPFL.

To address some of the mobility issues as part of the project objectives, projects such as the Swarm-bots [70, 61], Swarmanoid [75] and Symbrion [69] involve robots that can be assembled together for greater flexibility in mobility. For example, Figure 2.5(a) shows an example of two s-bots in the Swarm-bots [70, 61] project which are joined together to move past a large gap which is too large for a single s-bot to negotiate. Figure 2.5(b) shows foot-bots which can be joined together with hand-bots for additional mobility in the Swarmanoid project [75]. The foot-bots are capable of moving on rough terrains and transporting other robots. Hand-bots on the other hand are capable of vertical climbing but not horizontal motion on terrains. When a hand-bot is joined to foot-bots, the whole unit can move on terrains as well as reach higher grounds. Similarly, in the Symbrion project [69] the assembled artificial organisms can perform 3D motions.

In nature, the individuals in SI systems often have a wide range of sensory systems, allowing for an effective perception of the physical world. However, analogous sensory systems in the robots in swarm robotics research are still below the levels of those in nature. To provide robots with the same range of sensory systems as natural SI systems would mean weighing the robot down with heavy equipment. Until sensory systems become more accurate, lightweight and portable robots are restricted to limited sensory capabilities.

Transition from Laboratory Research to Real-World Applications

With the state of current swarm robotics research and the advances achieved so far, as evident from the work carried out in large-scale swarm projects, it is only a matter of time before swarm robotic systems are implemented for real-world applications. However, to translate the existing work from the laboratory



(a) (b)

Figure 2.5: (a) Two s-bots joined together to form a swarm-bot configuration to pass a large gap; (b) A hand-bot in the middle is joined to three foot-bots.
[a]Image from www.swarm-bots.org/index.php?main=3&sub=31&conpage=sbot. Reused with permission under the copyright of the Swarm-bots project [61]
[b]Image from www.swarmanoid.org/swarmanoid_hardware.php. Reused with permission under the copyright of the Swarmanoid project [75]

to the physical world is not as straightforward as it seems. In laboratory settings, many of the environmental conditions can be controlled. However, in the physical world, the environment is dynamic, often time-varying, and vulnerable to an abundance of interferences which are not present in laboratory settings. This difference between experiments in simulation and in the real world is typically referred to as *The Reality Gap* [76] and it has always been a target for roboticists to narrow this *gap*.

As a step forward towards the transition of swarm robotics research to real-world implementation, Winfield et al. [77] introduced the concept of the 'dependable swarm' and 'swarm engineering'. Swarm engineering is a fusion of dependable systems engineering and swarm intelligence to explore the development of engineering systems based on the swarm intelligence paradigm through a rigorous process of design, analysis, and be testing for dependability. Therefore, a dependable swarm *is a robotic swarm engineered to high standards of design, analysis and test, and is able to exhibit high levels of safety and reliability* [1]. Although some might argue that the swarm intelligence paradigm is intrinsically unsuitable for use in engineered systems that require a high level of integrity, Winfield et al. [77] believe that the validations for these systems, in principle, are not greatly different from conventional complex systems.

As part of the road maps of the work that needs to be done towards creating dependable swarms, Winfield and Nembrini [1] looked at the aspect of fault tolerance in robot swarms through Failure Mode and Effect Analysis (FMEA) and

reliability analysis. They analysed the possible internal and external faults of a robot swarm involved in a swarm containment task using the FMEA methodology. The results from the analyses showed that the robot swarm exhibited a high level of robustness due to redundancy of robots and lack of single point of failure because the robots are distributed.

Another important result from the analyses in Winfield and Nembrini [1] is that whilst redundancy of robots does promote robustness to failure, there are exceptions. In particular, when a robot faces a partial failure in which only some components are faulty whilst others are still working perfectly. For example, in their swarm containment task, a fault to the wheels while the communications devices are still working causes physical anchoring of the swarm and thus the incompletion of the task. This observation is significant because it demonstrates that instead of a graceful degradation of performance in the event of failure to individuals in the robot swarm, in some cases, the failed individuals *may* and *can* themselves cause interference that leads to catastrophic failure. This observation is also demonstrated previously in Groß et al. [7] for a group transport task in which a partial failure to one robot in a group of six robots makes the transport of a 3kg object impossible. Hence, it is important to ensure that the SRS are also tolerant to this kind of failure.

In summary, to transition swarm robotics research from laboratory to real world implementation, important issues such as the dependability of the robot swarm - not exhibiting undesirable behaviour, adaptivity to uncertain time-varying real world environment, and fault tolerance to partial failure needs to be addressed.

2.3 Summary

This chapter has presented a historical account of swarm robotics from the early work in swarm intelligence to relevant inspirations from natural systems which laid the foundations of swarm robotics. In Section 2.1, a brief background of the swarm intelligence was presented from the first use of the term 'swarm intelligence' in the late 1980s to describe a group of cellular robots to the current discipline encapsulating work in both natural and artificial swarm intelligent systems. Since artificial swarm intelligent systems were inspired by their natural counterparts, the relevant natural swarm systems and their biological principals were also presented and discussed. To provide a clearer picture of the mapping from natural swarm systems to artificial swarm systems, two examples (ACO and PSO) were provided from the biological inspirations to the development of

artificial swarm algorithms.

Swarm robotics is an instance of artificial swarm systems inspired by natural swarm systems, in particular by ant colonies. Section 2.2 presented the characteristics of swarm robotic systems according to Şahin [3] that distinguish it from other multi-robot systems. As the natural counterpart, swarm robotic systems are complex and research work on them has covered a wide range of topics from the modelling of a single robot, to various swarm tasks that may or may not require cooperative effort.

Because most of the current research in swarm robotics has been implemented in the laboratory, one of the challenges is to make sure that swarm robotic systems behave the same in the real-world environment. Thus, Section 2.2.3 reviewed some of the challenges related to the transfer of swarm robotic systems from the laboratory to real-world implementation. Traditionally, it has been assumed that in swarm robotics, fault tolerance is implicit as provided by the large number of robots in the system. The redundancy of robots allows for a graceful degradation of the performance in completing a given task. However, the work by Winfield and Nembrini [1] and Groß et al. [7] have revealed that there are exceptions. It was demonstrated that robots with partial failure to some components *can* and *will* interfere with the completion of a given task and thus have a detrimental effect on the swarm. Therefore, additional mechanisms need to be put in place to handle this type of situation. In recognition of this, the next chapter will review common fault tolerance approaches and in particular those which are applicable to swarm robotics.

CHAPTER

THREE

Fault Tolerance in Swarm Robotics

Fault tolerance is essential in swarm robotic systems. Generally, fault tolerance is implicitly present in swarm robotic systems due to the redundancy of robots in the swarm. However, there are exceptions [1]. In many tasks, faults must be handled explicitly for the system to be fault tolerant [12]. In Section 3.1, the logical differences between a fault, an error and a failure, which is relevant to explain the context of the *error detection* as part of a three-stage error detection and recovery approach is presented. Then, Section 3.2 reviews the common fault tolerance approaches which can be classified into redundancy-based, and explicit error detection and recovery. This is followed by a general overview of some of the challenges in addressing fault tolerance in swarm robotics in Section 3.3. Then, related work on error detection in swarm robotics is presented in Section 3.4. Lastly, Section 3.5 introduces the potential of artificial immune systems for error detection by reviewing related immunological background and the Receptor Density Algorithm (RDA) which is a particularly appealing AIS algorithm for error detection.

3.1 Logical Difference Between a Fault, an Error, and a Failure

Any system, even with a reasonably well-defined operational environment, may still experience undesirable behaviour due to a variety of reasons. The undesirable behaviour may be caused by flaws in the development stage or outage due to the operational environment, which is particularly relevant for SRS deployed

in real-world. These undesirable behaviours can be viewed at three logical levels: faults, errors, and failures [78].

A fault is a defect that occurs in some parts of a system [78]. A fault that occurs during one development stage may not be manifested immediately but rather become apparent at some later stage. For example, a programmer's mistake is a fault; a short-circuit in an integrated circuit is a fault; an electromagnetic perturbation is a fault; and an inappropriate man-machine interaction during system operation is also a fault [79, 78]. In robot swarms, faults that can occur to a robot can be component faults such as faulty wheels, sensors, or the circuitry.

In general, faults to systems can be further classified into various categories such as development faults, physical faults, and interaction faults. Avižienis et al. [78] present a taxonomy of faults and a comprehensive description of each category. To be aware of the many possible causes and nature of faults is important. It reflects the amount of work involved and aspects to be taken into consideration for the realisation of real-world SRS.

A fault may not be directly observable, but its effects, referred to as errors, can be observed [80]. In other words, an error is the manifestation of a fault in the system. For example, faulty wheels on a robot might not be directly observable (unless they have physically fallen apart), but its effect on the behaviour of the robot (i.e. the operational data) can be observed. By analysing the changes in data, it is possible to be aware that something has occurred. The approach to infer the presence of a fault based on the operational data is referred to as data-driven error detection [9].

If an error is activated during the operation of a system and not rectified, it can lead to a failure of the system [79, 81]. In swarm robotics, the failure can be in the form of uncompleted tasks. In Winfield and Nembrini [1], faulty wheels of a few robots eventually leads to the anchoring of the whole swarm and the swarm is unable to move to the target location.

In short, an error is the manifestation of a fault in the system, and a failure is the manifestation of an error on the service. Thus, a failed system is the one which cannot deliver its intended service. A failure, if not handled properly, can lead to further faults and errors. This chain of events is depicted in Figure 3.1. In the figure, the arrows express the causality relationship between the three impairments. A fault, once activated, leads to errors. These errors propagate and eventually leads to a failure. A failure that is not rectified may cause other faults and thus initiate a chain of events.

Therefore, the ability to detect errors and rectify faults is crucial to prevent failures to systems. This is to ensure that the system is fault tolerant, to be able of

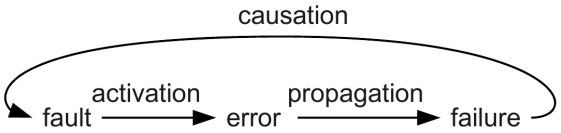


Figure 3.1: Chain of fault-error-failure.

continued operation in the event of faults in some its parts. In general, there are a number of approaches to address fault tolerance in systems. The next section present related work on error detection in swarm robotics.

3.2 Fault Tolerance Approaches

Fault tolerance is the ability of a system to operate continuously even in the presence of faults [79]. From the literature, fault tolerance approaches can be generally grouped into two categories: tolerance through redundancy, and tolerance through explicit error detection and recovery (EDR). Redundancy is the most common fault tolerance approach and it can be achieved either with hardware redundancy, software redundancy, or a combination of both. However, for swarm robotics, software redundancy is often not considered for the detection of component faults. Software redundancy involves the usage of additional codes, or routines to check the correctness or the consistency of the results produced by a given software. Although the controller can be faulty as well, it is generally assumed to be working to focus on the component faults. EDR on the other hand involves a three-stage process of error detection, fault diagnosis, and fault recovery.

3.2.1 Hardware Redundancy

Hardware redundancy involves the inclusion of additional hardware that carry out concurrent computations and the final output is based upon the result of some voting mechanism. Alternatively, duplicated hardware can be used as a backup and will be turned on automatically to replace failed components [81].

The most common hardware redundancy technique is the N-Modular Redundancy (NMR) [81]. With NMR, a system is equipped with N redundant hardware processing the same input. In case of any discrepancies, the output is determined with the use of a majority voter.

The most basic form of NMR is the Triple Modular Redundancy (TMR) [81]. Figure 3.2 shows a simple arrangement for a TMR. In this case, there are three

identical units performing the same computation concurrently and their outputs are fed to a majority voter. As long as at most only one unit produces incorrect output, the system will produce a correct output. If more than one unit is expected to be faulty, then more units need to be included. An example is in Sklaroff [82] in which five redundant computers are used in an expanded TMR voting scheme on the space shuttle. A general rule on the number of redundant hardware to include is $2k$+1 to tolerate k faulty units.

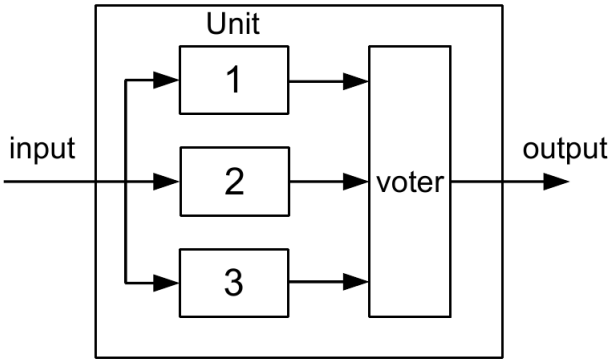


Figure 3.2: Triple Modular Redundancy. The same input is fed to three identical units and the output of the system is determined by majority voting carried out by a voter.

Hardware redundancy is simple but expensive to implement due to the extra units and interconnections between components.

In swarm robotics, hardware redundancy is implicitly present at the system level due to the number of robots in the SRSs. At the individual-level, hardware redundancy can be achieved with additional components or sensors on the robot. However, hardware redundancy at the individual robot level is typically not a viable option as the additional hardware increases the cost of implementing the SRS and the power consumption for a robot, which is relatively limited. In addition, any additional hardware added is also subject to faults itself.

3.2.2 EDR: Explicit Error Detection and Recovery

Fault tolerance through EDR (explicit error detection and recovery) is another common approach and particularly common in engineering disciplines. EDR involves a three-stage process: error detection, fault diagnosis, and fault recovery [8] (see Figure 3.3). Error detection identifies erroneous states while fault diagnosis determines the causes of an error including the nature of the fault and the exact location of the fault. When the causes and location of a fault have been

identified, recovery measures can then be carried out to prevent the fault from re-occurring. This can be done by either disabling the faulty components from being invoked again or repair the faulty unit. If a failure still persists after a recovery measure has been carried out, that information may be used as a form of feedback to the error detection and fault diagnosis mechanism for tuning and maintenance purposes.

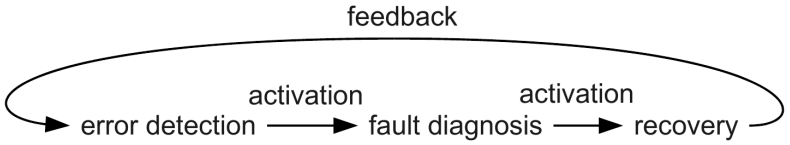


Figure 3.3: Three stages in EDR: error detection, fault diagnosis, and recovery. The activation of recovery is dependent on the output from the fault diagnosis, which in turn depends on the error detection.

The function of the error detection stage is to determine whether a system is behaving within its boundaries, i.e. whether it is behaving normally. Generally, there are only two states in which a system can be in, normal or abnormal. In other words, error detection in the EDR approach can be treated as an equivalent to a two class anomaly detection problem.

In robotics, there are two approaches in which error detection are generally carried out: model-based and data-driven approaches. In model-based approach, a model of how the system should behave is constructed and actual observations during operation are then compared against this model. A deviation from the model is interpreted as being a symptom of a fault. The problems with this approach is that constructing the models are difficult and often infeasible for systems deployed in the real-world [9]. To address this problem, an alternative approach referred to as a data-driven approach infers the presence of a fault solely based on the data during the operation [9]. The error detection is still based on the deviations but no models are constructed prior to deployment. Rather, the deviations are calculated based on observations during the system's operation.

In fault diagnosis, the output from error detection is used to locate the cause of the fault. This often involves the use of a priori knowledge in mapping between the observations and the failures [83]. The mapping may be explicitly specified in the form of lookup table or inferred from domain knowledge. The challenge in an effective fault diagnosis is acquiring the domain knowledge and mapping it to obtain an accurate diagnosis of the specific fault. There are a number of approaches proposed over the years on obtaining the domain knowledge. A 3-part companion paper [83, 84, 85] provide details on these approaches.

After a fault is identified, recovery mechanisms can be carried out accordingly. A simple recovery measure can be to shut down a faulty system or faulty units permanently, or restarts after a period of time. Other more advanced recovery measures include containment of faulty units, self-healing [86], reverting to previously error-free state (backward recovery), finding a new stable state (forward recovery), or compensation with redundancy [79].

Amongst the three stages in EDR, error detection is probably the most important stage because it is the first stage in the process and the activation of subsequent stages depends on the error detection mechanism to function properly.

In swarm robotics, applying EDR for fault tolerance on top of the redundancy of robots is the most promising approach compared to hardware redundancy. Although it is still relatively new, some existing work on error detection [9, 10, 13], fault diagnosis [87], and recovery [86] can be found. However, for the existing work on error detection, the aspect on the effects of uncertain time-varying environment on the error-detection ability has been largely ignored. Because the environmental condition in the real-world is often dynamic and judging from the potential applications of swarm robotics [3], the issue of adaptivity to the environment is important. The next section will review existing work on error detection in swarm robotics.

3.3 Fault Tolerance Challenges in Swarm Robotics

The results from Winfield and Nembrini [1] demonstrate that explicitly determining faulty robots for fault tolerance is an important topic towards realisation of swarm robotics research for real-world application. The reason is that some faults on individual robots may cause the whole system to fail even though there are still many fault-free robots in the system. Therefore, future research need to consider the consequence of partial robot failures and design measures to counter the effect of such partial failure [1]. Although the observations in [1] were specific for swarm taxis, it is used as a general motivation for current work.

In current swarm robotics research, there is little work on the explicit fault tolerance to deal with partial component failure. One notable work is by Christensen [9] within the Swarm-bot project [61] in which the author looked at the detection of internal hardware faults (endogenous detection), and faults that occur on other robots (exogenenous detection) with s-bot robots. Another work is by Mokhtar et al. [13] within the Symbrion project [69] for the detection of faulty sensors.

There are many challenges towards fault tolerance in swarm robotics with

explicit detection. First of all, since there is no centralised control and global information, how to identify which robot or which component is faulty? Secondly, for SRS in real-world, there are abundance of interferences and the environment is uncertain and often changes as time progresses (time-varying). Thirdly, as discussed in Section 2.2.3, individual robots are often limited in battery and computing power. Therefore, it would be an added advantage if the detection mechanism is lightweight in resource usage.

To address some of these challenges, inspiration from other disciplines may provide some valuable insights. For example, studies of self-isolation in social insects [88] may provide a good source of inspiration on the detection of faulty robots. Also, concepts from Social Comparison Theory [89] can provide important ideas for decision making in uncertain time-varying environments. Chapter 4 will explore these ideas and propose the idea of self-detection for the identification of faulty robots and coping with time-varying environments.

Before a more detailed discussion on error detection in swarm robotics, it is worth clarifying the terminologies and conventional approaches to address fault tolerance in engineering disciplines.

3.4 Related Work on Error Detection in Multi-Robot Systems

The error detection in the EDR is responsible for inferring the presence of faulty components. As the construction of accurate models on how the robots should behave (i.e. model-driven) in dynamic environment is difficult and not feasible, data-driven approaches to error detection are preferable. Based on the detection target, Christensen et al. [11] classify data-driven error detection approaches in multi-robot systems into endogenous and exogenous detection. Note that they refer their work as fault detection instead of error detection. However, based on the definition in Section 3.1 it will be referred to as error detection in this thesis.

3.4.1 Endogenous Error Detection

An endogenous error detection refers to the detection of errors that occur internally by an observer robot [9]. Therefore, each individual is responsible to determine whether itself is fault-free. If an observer robot is responsible for the detection of errors on other physically separated robots, the detection is referred to as an exogenous detection.

Artificial Neural Networks (ANN) is a popular approach to error detection in robotics. In Terra and Tinós [90], an ANN was used in conjunction with radial basis function (RBF) for the detection and isolation of faults on the joints in robotic manipulators. A multilayer perception (MLP) with back-propagation algorithm was used to produce the dynamics of fault-free system before it was tested. The outputs of the ANN are then compared with actual position and velocity measurements. The difference, the residual vector, is then analysed with a RBF network to detect and isolate the faults.

In Skoundrianos and Tzafestas [91], local model networks (LMNNs) were used to detect faults in the wheels of a mobile robot. The LMNNs were used to capture the input-output relationship between the voltages applied to the motor driving the wheels and the speeds of the wheels. In operation, the predicted speeds by the LMNNs are compared to the actual speeds. The difference, the residual, are computed using change-detection algorithm. The source of faults can then be found by analysing the values of the residuals.

Christensen et al. [11] employed a variant of ANN called Time-Delay Neural Network (TDNN) to detect hardware faults on s-bot robots through the examination of sensory inputs. TDNNs [92] are modified ANN with the addition of delays as shown in Figure 3.4. In the figure, delays D_1 through D_N are added to the basic unit of an ANN. Each input is multiplied by several weights, one for each delay and one for the undelayed input. For instance, a TDNN with $N = 2$ (two delays) and $J = 16$ (16 inputs) has 48 weights (3 x 16 = 48). The learning of a TDNN is through a backpropagation process [93]. Backpropagation involves two passes through the ANN: forward pass and backward pass. In forward pass, an input is applied to the network with its current connection weights (initially set either randomly or predefined). Calculated output is then compared with desired output and its error is calculated. During the backward pass, the derivative of this error is propagated backwards through the network, adjusting weights to minimise the error. This process is repeated until the network converges to produce the desired output.

Whilst the error detection with TDNNs produces positive results for the experiments in Christensen et al. [11], the authors pointed out that TDNNs consume huge resources and can be problematic for resource limited SRS. Due to the number of inputs, the number of connection weights for the TDNNs was very large, up to thousands. If more inputs were used, the number of weights would increase at a rate equals to the number of inputs times the number of neurons in the hidden layer [9]. This affects the scalability of this method. As with other ANN, they reiterate that the amount of training data required is important. Thus,

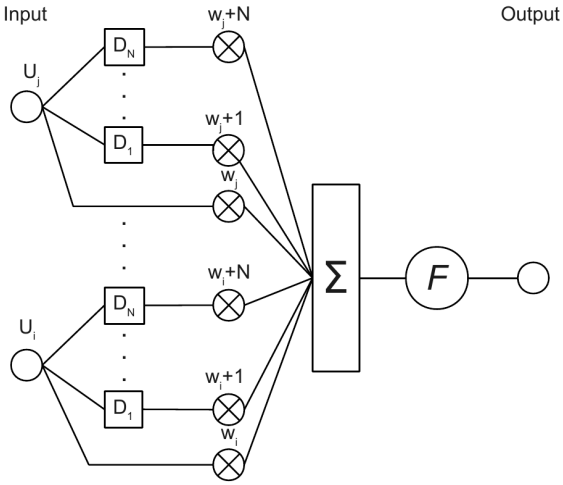


Figure 3.4: A Time-Delay Neural Network unit [92]. For each unit $u \in U$, N delay units are added. Reprinted with kind permission from IEEE: ASSP [92], copyright (1989).

a *sufficient* amount of training data needs to be collected. As noted in Christensen [9], any changes in the control program or faults also require retraining of the networks.

Endogenous error detection utilising multiple fault models on how the system should behave is another interesting approach. In this approach, faults are estimated based on how close the current state of the system is to the fault models. In Roumeliotis et al. [94], four Kalman filters (the models) are used to detect sensor faults. Each filter produces a residual by comparing the actual sensor values and the predicted values. Then, the residuals are fed into the detection and identification modules. The faults are then identified based on hypothesis testing on the residuals. This work was later extended in Goel et al. [95] to employ an ANN in the fault identification module. A potential problem with Kalman filters, in its basic form, is that it is assumed that the modelled system can be approximated. However, robotics systems are nonlinear, in particular when deployed in real-world environment [9].

Recently, artificial immune systems (AIS) has also been applied to error detection in multi-robot systems. In Canham et al. [10], an error detection based on the negative selection process in the vertebrate immune system was implemented on a Khepera robot, and a BAE System Rascal^{TM} robot. In the immune system, the negative selection process is observed in the thymus. In the thymus, a group of immune cells called the T-cells undergo a maturity process called negative selection before being released into the body. In this process, the T-cells that bind to the body's local molecules (self protein) are removed. T-cells that survived

this process are released from the thymus and distributed throughout the body to monitor non-self protein. From this observation, Forrest et al. [96] devised the Negative Selection Algorithm (NSA) involving the definition of self, generation of detectors in the complementary space of self and applying the detectors to classify new data as either self or non-self. This is similar to the model-based approach in which the models of expected behaviour are predefined. It was reported that the implemented NSA-based detectors managed to detect all errors that are greater than a predefined threshold and correctly identified unseen data as erroneous.

Based on the NSA, if the behaviour of a system can be expressed as a function that maps a single input to a single output in a deterministic manner, then data that described a system state during all normal behaviour is treated as self [10]. Deviation from this data is considered to be non-self and said to be erroneous.

Error detection in Canham et al. [10] is a two-phase process: the learning phase and the monitoring phase. The learning phase generates self detectors that represent normal or the steady state of the system. The data is the set of infrared sensor values and corresponding motor speeds. Detectors are initially created blank and then training data applied to the detectors. If data falls within a detector, a window is generated spanning the limits of the data within that detector. When more data falls within a detector's windows, the height of the detector increases to cover the data. If a detector's height is greater than a specific range (or threshold), it is split into two new blank detectors and previous data is re-applied to them. This process is repeated until the specific threshold is maintained. Using this approach, the robot learns a priori the states that are considered as self (fault-free) and non-self (faulty). In the monitoring phase, data that does not match the self detectors are treated as errors. No further learning of self is done in the monitoring phase.

A drawback of the error detection in Canham et al. [10] for SRS in the real-world environment is that the errors are predefined and no adaptations made to dynamic changes. If the operational environment of the SRS is dynamic, a high level of false positives might occur because the new fault-free states have not been encountered before. Similarly, if new errors were introduced to the system, this method has no capability to detect such new errors. This can results in a high level of false negatives.

Mokhtar et al. [13] implemented an error detection system employing a modified version of the Dendritic Cell Algorithm (DCA) in a resource limited microcontroller as part of an integrated homeostatic system in the Symbrion project [69]. The DCA [97] was developed based on the abstraction of the functionalities

of the dendritic cells in the immune system. Dendritic cells are responsible for some of the initial pathogenic recognition process, sampling the local environment for antigens, and differentiating between the semi-mature or mature state depending on the concentration of signals, or perceived misbehaviour, in the host tissue cells [97].

Mokhtar et al. [13] adapted the initial DCA for an online implementation with an emphasis on the resource limitation on the chosen micro controller. Their implementation is referred to as the mDCA. The antigens to the mDCA are the sensor values and signals are derived based on deviations among sensor values during system operation. The mDCA was able to detect two faults: *stuck-at-value*, and *random sensor values*. Results obtained showed that the mDCA is capable of immediate online detection of errors.

The work in Mokhtar et al. [13] as well as Christensen et al. [11] and Canham et al. [10] was demonstrated for an endogenous error detection utilising only data from a single individual. This could be an under utilisation of the advantage provided by the many interacting robots in SRS that can be harnessed for a possibly more efficient detection. Whilst the endogenous error detection with data from a single individual is suitable for SRS in non-dynamic environment or in situations in which the data are not influenced by the changes in the environment, it might not be in uncertain time-varying environments. Utilising data from all robots within a local neighbourhood can be potentially useful when dealing with dynamic environment as an observer robot can use these data to cross-reference it's own behaviour with other robots.

3.4.2 Exogenous Error Detection

Exogenous error detection refers to the detection of errors on other physically separated robots by an observing robot.

Exogenous error detection often requires relatively complex robots to track progress of the tasks, and thus (in one way or another) some form of global knowledge. This form of error detection is often incorporated as part of the control mechanisms. In Parker [98], an architecture for cooperation among teams of heterogeneous mobile robots called ALLIANCE was introduced. Based on the ALLIANCE architecture, each robot in a team has a number of task-achieving behaviours. These behaviours are grouped into behaviour set so that competing actions cannot be pursued in parallel. The activation of the behaviour set is controlled by a mathematically modelled variable called *motivation*. When a robot fails to achieve satisfactory progress on its current task (maybe due to faults or

failures), its motivation for performing the current task decreases and eventually it abandons the task. This robot can then try to perform other available tasks that it may still be able to perform. Alternatively, if another robot discovers that there is no progress in the task undertaken by the failed robot, the motivation of the robot to perform the same task increases and eventually takes over. When a robot is performing a task, it will broadcast a message to other robots informing them about its current activities (hence, decreasing the motivation of other robots for the same task). The ALLIANCE has been demonstrated to be able to produce high degree of fault tolerance.

Gerkey and Matarić [99] used MURDOCH, a general-purpose task allocation system to implement a distributed pusher-watcher system for box-pushing task. The robots used are heterogeneous robots with different capabilities. The pusher can see the box to be pushed (cannot see the goal) while the watcher can see the destination where the box should be pushed. The watcher (coordinator) needs to guide the pushers to push to box to the goal. In MURDOCH, when a task is to be assigned, the task is put up for auction by broadcasting the task announcement and capable robots bid for the task. The bid is time-limited contract with the best-suited robot wins the auction and consequently executes the task. In the experiment, the watcher monitors the progress of task completion by pushers. If a pusher robot fails to complete the task within a given time duration, it is considered ineligible for further tasks and the task will be re-assignment to another robot.

In Dias et al. [100], a TraderBots approach to coordinate multi-robot system was presented. Similar to MURDOCH, TraderBots is also inspired by the contract net protocol by Smith [101] for explicit communication and control. In their approach, strategies such as frequent auctioning and bidding to reallocate tasks among robots, monitoring communication connectivity to robots with subcontracted tasks, and continuous scheduling of assigned tasks for execution as tasks are completed are employed. Partial robot malfunctions are detected by monitoring the resources available to the robot. In this case, unforeseen depletion of resources is assumed to be a partial robot failure. Once an error is detected, the robot will reallocate all the tasks it cannot complete to other robots. Robot death is detected by monitoring periodic signals from robots. Once a dead robot is discovered, all tasks assigned to that robot will be reassigned to other robots. This approach assumes global communication capacity and every robot is able to sense other robots in the environment.

The approaches mentioned above are for multi-robot systems consists of relatively complex robots with high-level coordination and sophisticated reasoning

[9]. The robots often allocate task and track the progress of the mission by communicating over a network.

In swarm robotic systems, simpler robots are employed and the behaviour of a robot is based on the local sensing and communications. In Christensen et al. [12], an exogenous error detection on a *follow the leader* task in which a pre-assigned *leader* robot tries to detect errors on a pre-assigned *follower* robot. The *leader* moves around in the environment and the *follower* tails the *leader* and tries to stay at a distance of 35 cm. If the *follower* falls behind, the *leader* waits. During a training phase, the sensory data from the *leader* robot is collected whilst faults were injected to the *follower* robot. The data was correlated with the fault-state of the *follower* robot and a TDNN was trained to detect the errors. Results show that an exogenous error detection is possible. However, only two robots were involved in the experiment and they were pre-assigned. As noted by the authors, they are other issues to be considered such as the scalability to more robots and how to correlate the fault-state with more than one robot that may or may not be within a sensory range.

In Christensen et al. [102], an exogenous error detection inspired by the synchronisation phenomenon of fireflies that flash in harmony was presented. The fireflies' synchronisation is an example of coupled oscillating systems referred to as *pulse-coupled* oscillators, where one oscillator only influences other oscillators during short, periodic pulses [102]. Each robot acts as an *pulse-coupled* oscillator and when the activation of the oscillator reaches a certain threshold, the robot lights up its red LEDs and resets its oscillator. When neighbouring robots (within 50 cm) visually detect the flash, they increment their own activation and flash in synchrony after a period of time. A robot is assumed to be faulty if it does not flash its LEDs in synchrony with a neighbouring robot. An observer robot P flashes its LEDs and monitors whether an observed robot Q flashes as well after a period of time. If Q does not flashes, Q is treated as suspicious of faults. Then P flashes the second time and observes whether Q flashes. If after the second flash and Q still does not flashes its LEDs, Q is detected as faulty.

This approach works on faults that directly affecting the flashing of LEDs such as hardware input-output (I/O) fault but not on faults such as broken wheels or a toppled robot [102]. For those type of faults, it has to resort to the endogenous detection. At some level, this approach demonstrates that cross-referencing a robot's behaviour against others is plausible and in some cases sufficient for the detection of errors. The work in this thesis is related to this approach but implemented in the context of an endogenous error detection.

3.5 The Potential of Artificial Immune Systems for Error Detection

Related work in Section 3.4.1 on error detection in swarm robotics reveals that many inspirations from biology have been applied to problems in swarm robotics. In particular, algorithms inspired by the vertebrate immune system, i.e., the AIS, have been successfully implemented in Canham et al. [10] and Mokhtar et al. [13]. Due to very close analogy between components in the immune system and swarm robotics, swarm robotics has been identified as the domain in which AIS can potentially be very effective [103]. In addition, the similarities between AIS and SI provides a good indication that both paradigms can be used together as complementary tools to solve complex engineering problems [20].

3.5.1 AIS: Artificial Immune Systems

Like many new computing paradigms, there is a number of definitions for AIS over the years. However, the central theme is the same in that the source of inspiration is from the vertebrate immune system. To ground the discussion on AIS to a single definition, this work adopts the definition in de Castro and Timmis [15] that defines AIS as:

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

From an engineering perspective, the definition of AIS in de Castro and Timmis [15] covers two important aspects. First, AIS is *inspired* by but *not constrained* by the biological processes of the immune system. This has the implication that the developed AIS do not have to be an exact equivalent of the immunological processes from which they are based on. Rather, it is an abstraction of relevant immunological properties that can be utilised for problem solving. To abstract the relevant properties require collaborations amongst researchers from various disciplines, as proposed in the conceptual framework [104]. Secondly, the primary motivation of developing an AIS is to solve engineering problems.

The early work on AIS can be traced back to the early 1990s by Ishida [105], Bersini and Varela [106] and Forrest et al. [96]. These early work are important because they provide a platform that attracts many researchers to the field. Many of these AIS algorithms are examples of interdisciplinary work. However, in more recent years, there are instances of AIS that have *drifted away from the more biologically-appealing models and attention to biological detail, with a focus on*

more engineering-oriented approach [104]. These AIS work are criticised to suffer from what is described as *reasoning by metaphor* [104] in which the AIS algorithms were developed directly from a naive biological model without much analytical framing of the representation's properties. Consequently, an initiative called immuno-engineering [107] which is inline with the conceptual framework [104] was proposed for the *development of biologically grounded and theoretically understood AIS*.

The AIS algorithms are inspired by immune models such as *self-nonself discrimination*, *clonal selection* [108], *immune network theory* [109], *danger theory* [110], and *tuneable activation threshold* [111]. Table 3.1 list of some of the examples of AIS algorithms and the corresponding inspirations of the immune system.

Table 3.1: Example of AIS algorithms and their corresponding immune inspirations.

Immune inspiration	**AIS algorithm**
Negative Selection	NSA [96]
Immune Network Theory	aiNET [112], RAIN [113]
Clonal selection	CLONAG [114]
Danger Theory	DCA [97]
T cell receptor signalling	RDA [115]

Over the years, AIS has been successfully applied to a number of problem domains such as classification, optimisation, control, learning, and computer security [103].

3.5.2 Immune System Overview

The immune system is typically viewed as a *defence* system in which its primary function is to protect the body and fight against harmful microorganisms. This defence mechanism involves many biological components such as immune-related cells (e.g. lymphocytes, phagocytes) and molecules (e.g. cytokines, major histocompatibility complex molecule) that are spread across many organs. The lymphoid organs, which can be functionally divided into primary and secondary organs, play a major role in the immune system. The primary lymphoid organs (e.g. bone marrow, thymus) are responsible for the production and maturation of immune cells such as lymphocytes. The secondary lymphoid organs (e.g. tonsils, adenoids, lymph nodes) are the hosts where interactions between immune components and pathogens occur [116].

With reference to the ability to recognise and respond to invading pathogens, the immune system can be broadly split into the *innate* and the *adaptive* compo-

nents. The innate immune system is passed on from parents to children through heredity (*germline* encoded) [116]. It can be considered unchanged during the lifetime of an individual. The adaptive immune system is also referred to as *acquired* immunity as the immune system also has the ability to initiate responses against a variety of novel pathogenic materials by producing proteins not encoded in an individual's genome at birth. Although the adaptive component is also passed on genetically, it can change (regions of an individual's genome) through genetic rearrangements [117].

The innate immune system can be considered as a first line of defence for the body. It is made up of immune cells such as phagocytes that can ingest and destroy foreign materials and plasma proteins that coat surfaces of foreign cells (broadly referred to as foreign antigens). Phagocytes have surface receptors (evolved over millions of years) that bind to common molecular patterns of pathogenic materials. These receptors are germline encoded. The binding of antigens to these receptors stimulate the phagocytes to engulf and ingest the antigens. In comparison to the adaptive immunity, the response time for the innate immunity is faster. In the processes of an innate immune response, active molecules such as cytokines that initiate and regulate adaptive immune response are secreted. In addition, the interactions between the two immune components are also initiated through the presentation of antigens by the *antigen presenting cells* (APCs) in the innate immune system to cells of the adaptive immune system. The APCs take (engulf) antigenic materials in the environment and process it into small segments of amino acids called *peptides*. The peptides are then loaded onto the molecules on the surface of the APCs called the *major histocompatibility complex* (MHC) [117]. The lymphocytes, e.g. the T-cell, can bind to the peptide:MHC complex (pMHC) through its receptors and initiate an immune response.

In the adaptive immune system, the molecular pattern on a cell's receptors can change during the lifetime of the individual. This allows the adaptive immune system to protect the body against pathogens not recognisable by the innate immune system. A class of cells called the *lymphocytes* is the most important cell in adaptive immunity. It is the lymphocytes to which the APCs present the antigens [117]. Lymphocytes consist of the T-cells and the B-cells. A receptor on a T-cell that reacts to the antigens presented by APCs is called a T-cell receptor (TCR). During an adaptive immune response, a lymphocyte that binds to antigen and being activated will proliferate and produce clones. These clones will react the same towards the same foreign antigen as the parent. Different from the T-cells, the progeny of a B-cell may mutate and this can cause the cell's receptors to react differently than the parent. The mutation allows the cells to recognise

variations of the same type of antigen. In addition to produce clones, the B-cells can also differentiate into long-lived memory cells that react faster upon second encounter with the same antigen [116].

3.5.3 Immunological Theories

Over the years, there are many general theories to *describe* immunology such as *self-nonself discrimination*, *danger theory* [110], and the more recent *tuneable activation threshold* [111]. However, there are exceptions to each theory which leads to the proposal of new theories to cover the observed exceptions. However, instead of viewing those theories as competing, they shall be treated as contributing towards the understanding of the immune system.

The most prominent theory of immunology is that of self-nonself discrimination which states that the immune system (via the T-cells) protects the body against foreign microorganisms through the discrimination of molecules produced in the body (self-peptides) and those of external microorganisms (foreign peptides) [117]. The production of T-cells that are capable of detecting nonself-peptides occur in the thymus through a process called *negative selection*. This theory requires that there are no T-cells outside the thymus that can interact with self-peptides [117]. However, there is evidence found in contrary to this basis such as the lack of immune reaction to foreign bacteria in the gut or food and autoimmune deceases [110, 118].

From observations on many immune responses that are exceptions to the self-nonself discrimination, Matzinger [110] proposed an alternative perspective on immunology coined the danger theory. The danger theory proposes that to initiate an immune response, more complex interactions beyond self-nonself discrimination must have been involved. It suggests that the immune system responds to damages (cellular stress or cell death) to host cells by detecting the danger signals [110, 118]. Danger signals are chemical signals released by cells under injury, stress or uncontrolled cell death. These danger signals are recognised by professional APCs called the *dendritic cells*. The dendritic cells can communicate the information to lymphocytes via immune messenger molecules called *cytokines*. The cytokines affects the probability of T-cell activation and thus provide a context to the antigen presented by the APCs [117]. Therefore, an immune response is only initiated based on the context of the current environment, and thus provide an explanation to the lack of immune reaction in the gut to foreign bacteria or food.

The tuneable activation threshold (TAT) hypothesis [111] is a recent theory

that postulate that the activation of T-cells is dependent on their interactions to recurrent signals (their recent history). For a T-cell to be activated, the excitation or stimulation of the cell when interact with an APC must exceed the *activation threshold*. However, the activation threshold changes according to recent history of interactions. For a T-cell that receives continuous interactions with the same peptides, the activation threshold will become higher. In other words, interactions of a higher magnitude are required to active the T-cell. This means that the T-cells that interact regularly with self-peptides will have a higher activation threshold [117]. Similarly, because the frequency of interacting with foreign peptides is low, T-cells will have a low activation threshold for the foreign peptides.

These immunology theories have provided inspirations to the development of AIS algorithms, some of which have been successfully implemented for error detection in swarm robotics. Amongst these AIS algorithms, the Receptor Density Algorithm (RDA) is the most recent one and it was inspired by the T-cell signalling mechanism in the TAT [111]. The RDA, being developed specifically for anomaly detection problems, is an appealing algorithm for the error detection in this thesis. Therefore, the next section will provide an overview on the biological inspiration and the development of the algorithm.

3.5.4 The RDA: Receptor Density Algorithm

The RDA was developed through the extraction of features of the generalised T-cell receptor, and mapped onto the kernel density estimation [115]. This section presents the biological background on the activation of the T-cell receptor and the mapping of the generalised receptor onto kernel density estimation.

Biological Inspiration

Of all pMHCs presented on the APCs, the TCRs on the T-cells must discriminate between pMHCs from the body (self pMHCs) and those of foreign pMHCs. The discrimination is fine grain as the self pMHCs comprises from 99.9 -99.99% of all pMHCs on a APC [119]. The binding of TCR to pMHC to initiate an immune response can be seen as regulated by at least four processes: *kinetic proofreading*, *negative feedback*, *negative feedback destruction*, and *tuning* [115, 117].

- ***Kinetic Proofreading:*** Before a TCR can generate an activation signal, the binding of a TCR-pMHC must be sufficiently strong and long enough. This involves energy consumption steps that are reversed when the binding dissociate;

- ***Negative Feedback and Base State:*** As the kinetic proofreading steps progresses, a signal that progresses in the opposite direction (i.e. reverses the steps) builds up over time. A negative feedback signal is only generated when the kinetic proofreading process is equal to or greater than the base negative feedback;
- ***Negative Feedback Destruction:*** If the kinetic proofreading of a TCR is complete, an activation signal is generated. This signal undergoes amplification (which can be viewed as a positive feedback) and is able to negate the effects of negative feedback;
- ***Tuning:*** On the surface of the T-cells, there are *co-receptors*. The density of these co-receptors influence the probability of activation of the T-cells. A small increase in the density of the co-receptors has the effect of increasing the probability of activation. However, if the increase is large, it has a negative effect and decreases the probability of activation.

The signals generated during a TCR-pMHC binding can spread to surrounding area and affecting the binding of other TCRs with pMHC on the T-cell. First, when a TCR-pMHC binding dissociates, the pMHC may rebind to a nearly TCR on the T-cell. Second, the negative feedback generated on a TCR spreads and dampens nearby TCRs. Third, the negative feedback destruction signal spreads and protects nearby TCRs from the negative feedback [120]. This signal spreading is a key concept for the development of the RDA.

The Generalised Receptor

The modelling work on the tunability of early T-cell signalling events by Owens et al. [120] inspired the author to abstract features of the T-cell's receptors for computation. The definition of a receptor r, taken from [117], is as follows:

Definition. *A receptor r is a tuple (p, n, β, ℓ, c), with p, n, β, $\ell \in \mathbb{R}$:*

- $p \in [0, \ell]$, *the receptor position;*
- $n \geq 0$, *the generated negative feedback;*
- $\beta > 0$, *the base negative feedback barrier;*
- $\ell \in (0, \infty), \ell > \beta$, *the length of the receptor;*
- $c = \{0,1\}$, *the receptor output. c = 1 if* $p \geq \ell$.

This definition of a receptor is *an abstraction of the internal component of the TCR* in which p represents the kinetic proofreading state, n represents the generation of negative feedback, β is the base negative feedback, ℓ is the final kinetic proofreading state for T-cell activation, and c is whether the T-cell is activated [117].

A receptor behaves as follows [117]:

- a receptor receives a sequence of input $\{u_t\}$ at discrete time t;
- the receptor position p_t and negative feedback n_t are updated accordingly. If p_t is greater or equals to base negative barrier β, then a negative feedback is generated. The update of p_{t+1} and n_{t+1} is according to Eq. 3.1 and Eq. 3.2.

$$p_{t+1} = bp_t + u_t - an_t, \tag{3.1}$$

$$n_{t+1} = \begin{cases} dn_t, & \text{if } p_t < \beta \\ dn_t + g, & \text{if } p_t \geq \beta \end{cases} \tag{3.2}$$

  with $b > 0$ is the receptor position decay rate, $d < 1$ is the negative feedback decay rate, $b < d, a > 0$ controls the influence of negative feedback, $g > 0$ is the negative feedback growth rate.
- If p_t is greater than or equals to ℓ, then a receptor activation occurs and c is set to 1.

Mapping The Generalised Receptor Onto Kernel Density Estimation

The target AIS algorithm from the modelling work on the tunability of the TCR is anomaly classification [117]. Formally, given a sequence of values (e.g. training data) $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n \in \mathbf{x}$, the task is to classify a subsequent value $\mathbf{v}$ as normal (class $\mathcal{C}_1$) or anomalous (class $\mathcal{C}_2$).

Assuming all training data belong to class $\mathcal{C}_1$, the new data $\mathbf{v}$ may belong to class $\mathcal{C}_1$ with probability $P(\mathcal{C}_1)$ and class $\mathcal{C}_2$ with probability $P(\mathcal{C}_2)$. The new data $\mathbf{v}$ is assigned to $\mathcal{C}_1$ if $P(\mathcal{C}_1|\mathbf{v}) > P(\mathcal{C}_2|\mathbf{v})$. Based on Bayes theorem, given the data on the training data $\mathcal{C}_1$, then $\mathbf{v}$ is assigned to $\mathcal{C}_1$ when

$$p(\mathbf{v}|\mathcal{C}_1)P(\mathcal{C}_1) > p(\mathbf{v}|\mathcal{C}_2)P(\mathcal{C}_2). \tag{3.3}$$

Kernel density estimation (KDE) can be used on the training data to model the distribution of $\mathcal{C}_1$. KDE is a non-parametric method for estimating the probability density function of a given set of data $\mathbf{x}$. The probability estimation $\hat{p}(\mathbf{x})$ is given by Eq 3.4:

$$\hat{p}(\mathbf{x}) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x}_i}{h}); \quad K(\mathbf{x}) \geq 0; \quad \int_{-\infty}^{\infty} K(\mathbf{x})dx = 1 \tag{3.4}$$

$K(.)$ is the kernel function with width h, n is the number of training data. For the

kernel, the Gaussian kernel $K(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-x_1)^2}{2h^2}}$ is commonly used.

Since only training data is given, the distribution of anomalous data is unknown. Therefore, it is assumed that the distribution of the anomalous data is uniform with a probability α [120]. This assumption recasts Eq. 3.3 as equivalent to applying a threshold on the estimated probability $\hat{p}(\mathbf{x})$ in Eq. 3.4. Then, the classification of $\mathbf{v}$ is

$$\text{Classification}(\mathbf{v}) = \begin{cases} \mathcal{C}_1, \text{if } \alpha \leq \frac{1}{nh} \sum_{i=1}^{n} K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) \\ \mathcal{C}_2, \text{otherwise.} \end{cases} \tag{3.5}$$

The abstracted features of the TCR in the computational context allow *exact* reconstruction of the KDE-based Bayesian anomaly classification [120]. In the case of univariate data, the receptor position and negative feedback are defined at every point $\mathbf{x}$ in $\mathbb{R}$. Let the receptor position at $\mathbf{x} \in \mathbb{R}$ as $r_p(\mathbf{x})$ and the corresponding negative feedback as $r_n(\mathbf{x})$. Based on the concept of signal spreading in previous section, the spreading of stimulation from input data $\mathbf{x}_i$ can be modelled via a kernel. Thus, the stimulation $S(.)$ from an input data $\mathbf{x}_i$ given by a stimulation kernel K_s:

$$S(\mathbf{x}, \mathbf{x}_i) = \frac{1}{nh} K_s(\frac{\mathbf{x} - \mathbf{x}_i}{h}) \tag{3.6}$$

To apply the mapping for anomaly classification, it is split into a training and a testing phase [120]. In the training phase, the receptor position $r_p(\mathbf{x})$ and the corresponding negative feedback $r_n(\mathbf{x})$ are established with the training data. In the testing phase, the new data $\mathbf{v}$ are tested to see whether $r_p(\mathbf{x}) \geq \ell$.

The RDA works as follow. The spectrum of input value is divided into s discretised location and a *receptor* $\mathbf{x}$ is placed at each of these locations. A receptor has a length $\ell = \frac{1}{nh(\sqrt{2\pi})}$, a position $r_p(\mathbf{x}) \in [0, \ell]$, a negative feedback $r_n(\mathbf{x}) \geq 0$, a negative feedback barrier $\beta \in (0, \ell)$.

During training, the stimulation and negative feedback of each input data $\mathbf{x}_i$ on each receptor $r_p(\mathbf{x})$ is calculated (Eq. 3.7). If the resulting $r_p(\mathbf{x}) \geq \beta$ then a negative feedback $r_n(\mathbf{x})$ is generated which acts to reverse the progression of $r_p(\mathbf{x})$. If $r_p(\mathbf{x}) < \beta$, no negative feedback will be generated, $r_n(\mathbf{x}) = 0$.

$$r_p(\mathbf{x}) = \sum_{i=1}^{n} \frac{1}{nh} K(\frac{\mathbf{x} - \mathbf{x}_i}{h}); \quad r_n(\mathbf{x}) = \begin{cases} r_p(\mathbf{x}) - \beta, & \text{if } r_p(\mathbf{x}) \geq \beta \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

The receptor position and negative feedback decay over time. During testing, for a new data instance $\mathbf{v}$ recalculate the receptor position $r_p^t(\mathbf{x})$ (Eq. 3.8). If $r_p^t(\mathbf{x}) > \ell$, then the receptor generates an anomaly classification c_t = 1 (Eq. 3.9).

$$r_p^t(\mathbf{x}) = b \times r_p^{t-1}(\mathbf{x}) + gb \times K(\frac{\mathbf{x}-\mathbf{v}}{h}) - a \times r_n^{t-1}(\mathbf{x}) \tag{3.8}$$

where $b \in \mathbb{R}^+$ is receptor position's decay rate,
$gb \in \mathbb{R}^+$ is current input stimulation rate,
$a \in \mathbb{R}^+$ is negative feedback's decay rate.

$$c(\mathbf{v}) = \begin{cases} 1, & \text{if } r_p^t(\mathbf{x}) \geq \ell \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

3.5.5 The Application of AIS to Robotics

AIS has been applied for a variety of areas in robotics from robot control, learning, to anomaly detection. In Ishiguro et al. [121], an immune network-based AIS has been applied as a behaviour arbitration mechanism for the selection of actions given current state of a robot in a garbage collection problem. Given the distance, energy level, and direction from the base, a robot has to decide whether it has enough energy to carry out the garbage collection or to return to base for a recharge. In Singh and Thayer [122], inspirations from innate and adaptive immunity were incorporated for multi-robot control for a minefield clearing task. In Krautmacher and Dilger [123], immune network-based AIS has been proposed and demonstrated to be feasible for robot navigation in unstructured and unknown environments in a rescue scenario. Subsequent work in Whitbrook et al. [124] improves on Krautmacher and Dilger [123] with reinforcement learning for maze navigation in detecting and tracking door markers.

Neal et al. [125] commented on the lack of appreciation for AIS utilising innate immune principles in robotics. To demonstrate the potential of innate immune principles in AIS, they implemented a mechanism of self-monitoring low-level responses and subsequently feedback to the high-level behaviour control through artificial inflammation. The states during the lifetime of a robot are represented with Kohenen's Self-Organising Map (SOM)[126]. Responses from various internal sensors serve as input to the SOM and sum of these responses are used to update the inflammation level. The system then passes the responses, which correspond to the winning node within the SOM, to the higher level controller. Results from experiments support the described principles. However, as pointed out by the authors, manual assignment of fault conditions and associated actions and the overhead of maintaining a system-wide SOM of robot's state might be problematic.

In Jakimovski and Maehle [127], the authors developed a robot anomaly detection engine called RADE targeted for the realisation of self-healing autonomous robots. RADE is based on the clonal selection principles and information was represented with triangular fuzzy logic membership sets. The work has been tested successfully with a normal robot walking, obstacles collisions, a robot with disconnected servo joint motor and a robot with a falling screw on servo joint.

3.6 Summary

This chapter reviewed common approaches to achieve fault tolerance in systems, in particular those that are relevant to swarm robotics. From the literature on error detection in robotics, a related terminology that is often used is 'fault detection'. Often, both terminologies referred to the same problem. To clarify the differences between a fault, an error, and a failure, Section 3.1 reviewed the work in Avižienis et al. [78] and Lyons et al. [80] which provide an explanation on the logical differences between the terms. Based on the definitions, work in this thesis is referred to as *error detection* instead of fault detection. An important distinction between the three terms is that an error is a manisfestation of a fault, and an error generally comes before a failure. Therefore error detection may enable prevention of a failure.

In Section 3.2, it was identified that fault tolerance approaches can be generally classified into redundancy-based, and those involving explicit error detection and recovery (EDR). Although fault tolerance through redundancy is implicit in swarm robotics, there are exceptions. For these exceptions, an explicit EDR approach is more applicable when compared to hardware redundancy (i.e. redundancy of robotic components). However, there are a number of challenges to implementing EDR in swarm robotics and they are described in Section 3.3.

Error detection is an integral component in an EDR, together with fault diagnosis and fault recovery. As part of the ongoing work on EDR in swarm robotics, this thesis focuses on the error detection component. Therefore, Section 3.4 reviewed the current state-of-the-art on error detection in robotics. From the review, it was identified that the aspect of deploying the SRS in dynamic operational environments has been largely ignored. In addition, most of the error detection techniques only utilise the data from a single robot's perspective. This is potentially an under utilisation of existing data that are provided by the many interacting robots. Therefore, it is believed these data can be harnessed for an adaptive error detection in dynamic environments.

Bio-inspired algorithms have been commonly applied for complex problems

including those that deal with adaptivity to dynamic environments. Within the domain of bio-inspired algorithms, AIS have been actively explored for error detection in swarm robotics. To provide some background on AIS, Section 3.5 presented some general information regarding AIS and previous work that uses AIS in the domain of robotics. One AIS algorithm called the Receptor Density Algorithm (RDA), being specifically developed for anomaly detection problem, is particularly appealing. Therefore, relevant biological inspiration and algorithm development are also presented in Section 3.5.

To continue the investigation of whether an adaptive error detection can be achieved by utilising the data from the interacting robots within a local neighbourhood (i.e. communication range), the next chapter will present the platform and context in which the experiments will be carried out.

Part II

Error Detection in Swarm Robotics: Developing an Experimental Framework

CHAPTER

FOUR

Experimental Testbed: A Foraging Robot Swarm

This chapter presents the experimental testbed for the work in this thesis. In Section 4.1, the taxonomy of robot foraging as compiled by Winfield [63] is presented to provide an overview on the various aspects of robot foraging. Then, a description on the simulator, Player/Stage, in which the experiments will be carry out is in Section 4.2. Section 4.3 presents the robot's specification, behaviour modules for the robot controller, and arena layout. The last two sections of this chapter present the specific fault models of the wheels (Section 4.4) for which the error detection mechanism has to detect, and the time-varying environmental conditions (Section 4.5) in which the robot swarm operates.

4.1 Taxonomy of Robot Foraging

Foraging is a canonical task in robotics, and particularly in multi-robot systems. It is a complex behaviour that requires careful consideration of various aspects of foraging such as behavioural modelling, cooperative transport, software control, and communication (refer Section 2.2.2). To roboticists, foraging is a useful benchmark for a number of reasons [63]:

- foraging in social insects, which *recently becoming well understood*, provides system-level models and inspirations for various artificial swarm systems. In other words, the understanding of the underlying mechanisms in foraging allows an easier, faster, and a more realistic implementation in swarm

robotics;

- foraging involves the coordination of multiple subtasks that are equally complex and challenging such as exploration, collection, transportation, homing and deposition of objects in the nest. By investigating different strategies for each subtask, more efficient solutions may be devised for targeted real-world applications; and
- effective foraging requires cooperation between individuals to communicate the location of food sources, to cooperate with the transportation of objects that are too large for a single individual, or both. Again, foraging provides inspiration for many real-world problems such as resource allocation, finding the shortest path and other analogous cooperative transport problems.

The basic foraging behaviour, as observed in ant colonies, involves the act of some agents *searching* for attractors be it food sources or a particular type of object, collecting or *grabbing*, and transporting the attractor back (*homing*) and *depositing* it at the nest. These basic actions (behaviour) expressed with a finite state machine are shown in Figure 4.1. In the model, an agent is always in one of the four states as depicted with rectangular boxes. This model presents a foraging process that is continuous, implying the presence of more than one object in the environment. Also, there is a central collection point (the base).

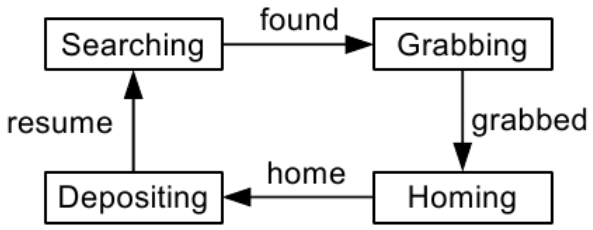


Figure 4.1: Finite state machine for basic foraging [63].

From this top-level abstract model, variations can be produced depending on specific research interests and desired level of details. It may be simplified or complexities added. For instance, foraging for a finite number of objects does not need to continue indefinitely. Thus, the process ends with the deposit of the last object. Similarly, more complex foraging tasks may require cooperation between a certain number of robots to carry an object that is too large or too heavy for an individual. The variation in robot foraging experiments can be seen from the taxonomy of robot foraging by Winfield [63] (reproduced as Table 4.1).

Table 4.1: A taxonomy of robot foraging. Adapted from [63].

Main Category	**Sub-category**	**Value**
Environment	search space	unbounded, constrained
	source areas	single limited, single unlimited, multiple
	sinks/bases/homes	single, multiple
	object types	single static, multiple static, single active
	object placement	fixed known locations, clustered, uniform distributions
	object placement rate†	constant, time-varying
Robot(s)	number	single, multiple
	type	homogeneous, heterogeneous
	object sensing	limited, unlimited
	localisation	none, relative, absolute
	communications	none, near, infinite
	power	limited, unlimited, consume object
Performance	time	fixed, minimum, unlimited
	energy	fixed, minimum, unlimited
Strategy	search	random wander, geometrical pattern, follow trail, follow other robots, in teams
	grabbing	single, cooperative
	transport	single, cooperative
	homing	self-navigation, home on beacon, follow trail, follow other robots†
	recruitment	none, direct, indirect
	coordination	none, self-organised, master slave, central control

† Added to the taxonomy.

Referring to the robot foraging taxonomy in Table 4.1, Winfield [63] classifies robot foraging research into four main categories according to the operational environment of the SRS (Environment), the characteristics of the robots (Robot(s)), the measurement on the performance of foraging (Performance), and the behavioural strategies to carry out the foraging (Strategy). Each category in turn has several minor sub-categories with several options of value that it can take. The meaning of each sub-category is self-explanatory but some will be further described as appropriate.

The Environment category concerns about the characteristics of the arena (space) in which a robot swarm operates and the objects of interest. Perhaps what is missing in this category, that is related to option of unlimited objects in the arena, is the rate in which new objects are added into the arena. This is referred to as the object placement rate in this thesis. In a controlled environment, the rate is often constant. However, in natural world, the rate is often dynamic (time-varying) and can be affected by various factors. Thus, this sub-category is added to the

existing taxonomy to reflect this fact.

In terms of the characteristics of the forager robots (i.e. the Robot(s) category), robot foraging experiments differ from one to another in terms of the number of robots involved, the topology of robots involved, the availability of positional information such as Global Positioning System's data, communication and sensing capability.

The success of the robot foraging task (i.e. the Performance), can be measured with respect to time and energy. In other words, the aim can be either to minimise time to forage or to minimise the energy used.

The last main category is the various strategies to carry out the foraging task itself. From the taxonomy, it is apparent that there are a number of strategies to carry out the basic foraging activities in Figure 4.1. In terms of the homing strategies, it is commonly known that in ants, the navigation to food sources and the nest is based on pheromone trails. However, what is less known is that there also exists species of ants that do not lay pheromone trails. An example is the desert ant *Cataglyphis fortis*. The reason is that in deserts, the pheromones will be quickly evaporated due to the extreme temperature [128]. For these ants, they rely on visual landmarks as cues to locate their nest. In the context of robot foraging, the landmark can be the home beacon, another robot that is also on the same way back to the base, or a trail [129]. Since the robot following behaviour is a strategy in searching for objects, it also can be a strategy for homing. Thus, this strategy is also added to the existing taxonomy.

From the taxonomy of robot foraging in Table 4.1, it is apparent that robot foraging is a complex task and the selection of a particular value for each of the variables is difficult and often hard to justify. Therefore, to focus on the specific problem, this work builds on top of an existing and established work by Liu [42], and extends it in the context of current research.

To investigate the research question of this thesis from the perspective of data-driven error detection, the following conditions need to be established:

- **E1:** A fault to a robot has an observable effect on the behaviour the robot as indicated by changes in the operational data. For a robot foraging task, an example of the operational data can be the number of objects collected within a time interval. Thus, from the perspective of data-driven error detection the presence of a fault can be inferred from the data;
- **E2:** The environment in which the robot swarm operates has a direct influence on the behaviour of the robots in the swarm. That is, when the environment changes the behaviour of the robots changes accordingly to reflect such changes; and

- **E3:** The robots are functionally homogeneous and act relatively independently from each other and no cooperation is involved to carry out the task. This is assuming that each robot should experience the same effects as the environment changes.

The robot foraging in Liu [42] is extended for the work in this thesis for a number of reasons:

- the model has been implemented and tested in simulation and validated with the macroscopic model. Therefore, there is greater confidence that when a fault is injected to a robot the observed changes to the behaviour of the robot is a direct consequence of the fault. Similarly, it also supports the same premise when time-varying changes that affect the behaviour of the robot swarm are introduced into the environment. This establishes the conditions of **E1** and **E2** to facilitate the investigation of the research problem from the perspective of data-driven error detection;
- it is a non-cooperative foraging in which the objects can be handled by a single robot. However, with a swarm of robots more objects can be collected in a shorter time. This establishes **E3**; and
- it involves a group of homogeneous robots in which all robots employ the same controller, and are equipped with the same number and type of sensors. This also establishes **E3**.

The conditions of **E1**, **E2** are investigated in Chapter 5 and the results from the experiments provide evidence to support them. The condition **E3** is inherently part of the robot foraging in [42].

4.2 The Simulation Platform: Player/Stage

Given the amount of published work in the robotics literature, the *Player/Stage* [130] is probably the most widely used robotic simulation tool. Naturally, it would be beneficial to implement the foraging SRS in Player/Stage so that the work can be replicated and shared more effectively.

The Player/Stage[1] comprises a network server called Player which communicates with hardware through the source code over a TCP socket and a plug-in called Stage which receives instructions from Player and moves robots in a simulated world and passes data to the Player.

[1]Latest version of Player/Stage can be found on http://playerstage.sourceforge.net/. For this thesis, Player 2.1 and customised Stage 2.1.0 from www.brl.uwe.ac.uk/projects/swarm/index.html are used.

Released under the GNU Public License, Player/Stage is a free open source software and runs on Linux, Solaris, some versions of BSD and Mac OSX (Darwin). All code from the Player Project is free to use, distribute and modify.

Player is designed to be language and platform independent. The client program can run on any machine that has a network connection to the robot, and it can be written in any language that supports TCP sockets. Client libraries in C, C++, Python, and Ruby are officially supported, while Java, Ada, Octave, and others are supported by third parties. Player supports a wide variety of robots and hardware, with new hardware easily supported by Player's modular architecture and its open interfaces.

A simulation with Player/Stage is composed of three parts: the main program, Player, and Stage. The Player takes codes from the main program to control the robots and redirect data from sensors to the main program. The Stage interfaces with the Player to receive instructions to move robots in the simulated world and redirects sensor data to the Player.

There are three basic files in Player/Stage: a *.world* file, a *.cfg* file, and a *.inc* file [131]. The .world file lists information regarding the arena including the robot, any other items and layout of the arena. These items can be defined using the built-in *model*s. For example, a robot can be modelled with a collection of models including a *Position* for the robot's odometry, a *gripper* model for the grippers, a *blobfinder* model to simulate camera, and *ranger* models to simulate infrared sensors. In each model, there is a list of associated properties that can be set accordingly. For full list of supported models in Player/Stage, readers are directed to the official documentation on Player/Stage website [132]. The .inc file follows the same syntax as the .world file. It stores information regarding the configuration (dimension) of the robot, and any other items to be placed in the arena. For convenience of changing a robot's description when necessary, instead of putting the description in a .world file, it can be put in a .inc and be referenced to in a .world file. Finally, the .cfg file is used by the Player to include all necessary drivers to interact with the robot. If a real robot is used, the (common) drivers are built-in to the Player. Alternatively, for simulation, the driver is always Stage [131]. A helpful tutorial on installing and creating simulations with Player/Stage is provided by [131].

All experiments in this thesis have been conducted in simulation (in 2D) using the Stage plug-in. The reason being that through simulation, it allows data to be collected easily in a controlled manner. The support for programs written in C/C++ in Player/Stage has the benefit that the robot foraging in [42] can be easily adapted to inject the necessary faults and to introduce the time-varying

changes to the environment in order to establish **E1** and **E2**. The ease to generate required logs for data analysis means that the same data can be shared with other researchers for comparison purposes.

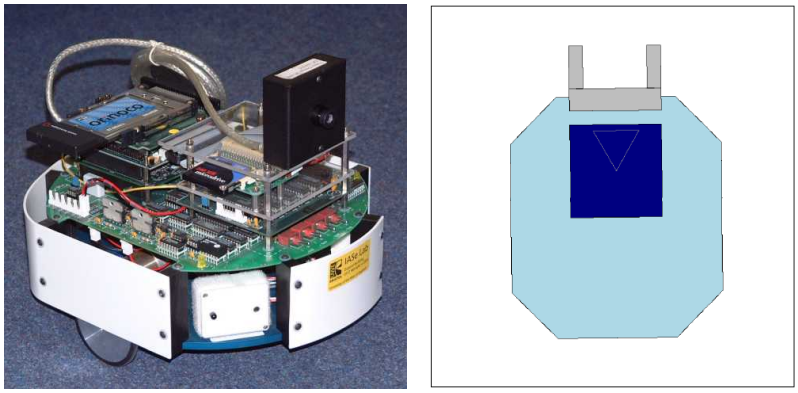
Another reason why this work is simulation-based is that during the course of the work, the physical robots suitable for the experiments were not available. Of course, ultimately any system developed should be deployed in an actual robotic system, but experience gained from this simulation process provides insights into the design, constraints, and limitations in the work when dealing with real robots in the future.

4.3 Simulation Setting

This section provide the details on the physical specification of the robot and the robot control designed using the behaviour-based subsumption architecture [49].

4.3.1 Robot's Specification

The robot's specification in Liu [42] is based on the Linuxbot developed at Bristol Robotics Laboratory[2]. The robot is octagonal-shaped with a dimension of 0.23m x 0.26m, and a mass of approximately 3.0kg (see Figure 4.2(a)). The robot is equipped with two wheels, a pair of front-mounted grippers, and an array of sensors. The robot has two degree of freedom (DOF) for the differential wheels allowing it to move forward, backward, to the left and to the right.



(a) (b)

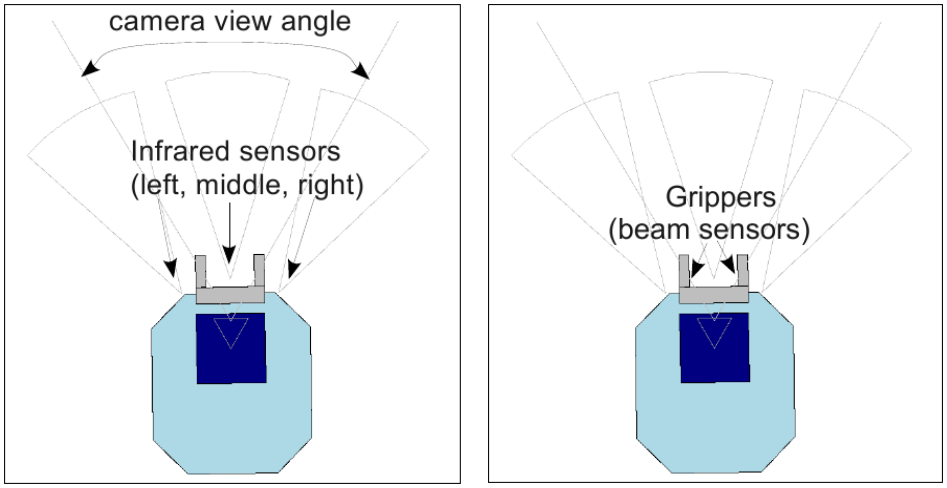
Figure 4.2: (a) Linuxbot without grippers and camera attached; image courtesy of A. Winfield, Bristol Robotics Laboratory, University of West England. (b) Linuxbot in simulation with front-mounted grippers and camera located at the top of the robot.

[2]http://www.ias.uwe.ac.uk/People Pages/a-winfie/linuxbot/linuxbot.htm

The robot is equipped with the following sensors:

- 3 front-mounted infrared (IR) sensors: on the left, middle, and right. The range of the IR sensors is 0.2m;
- 3 front-mounted light intensity sensors: on the left, middle, and right. The left and right sensors are 60° away from the middle sensor;
- a colour sensor located a the bottom of the robot;
- one camera located on the top of the robot. The camera has a viewing angle of 60° and a view distance of 2m;
- 2 beam sensors located on the grippers, one on each side.

Figure 4.2(b) shows the LinuxBot in simulation with attached grippers whilst Figure 4.3 shows the locations of the various sensors. In the simulation, the IR sensor is simulated with the `ranger` model, camera with the `blobfinder` model, light sensor with the `light` model, grippers with beams sensors with the `gripper` model, and the LinuxBot's odometry with the `position` model.



(a) (b)

Figure 4.3: Sensors on a robot. (a) A camera mounted on the top of the robot and three IR sensors in front of the robot. (b) Beam sensors at each sides of the grippers.

Table 4.2 is the specification of the LinuxBot in Stage. The `size` parameter in the `position` model specifies the dimension of the LinuxBot. The `mass` is the weight of the LinuxBot, and the `drive` specifies how the robot is driven. This value "diff" means that the robot can be controlled by changing the speeds of the left and right wheels independently. The `sview` parameter in the `ranger` model specifies the minimum, and maximum distances that can be sensed, and also the field of view (fov) in degrees (i.e.[min max fov)). In the `blobfinder` model, the `channel_count` parameter specifies the number of colours that can be detected

by the `blobfinder`, the `channels` parameter specifies the colours to be detected, and the `range` parameter specifies the maximum range the `blobfinder` model can sense in metres. For the `gripper` model, the `size` parameter specifies the width and length of the grippers.

Table 4.2: The specification of LinuxBot in Stage.

Robot Component	Model	Configuration
LinuxBot	`position`	`size` [0.26 0.23], `mass` 3.0, `drive` "diff"
IR sensor	`ranger`	`sview` [0 0.3 35]
Light sensor	`light`	
Camera	`blobfinder`	`channel_count` 4, `range` 2.0 `channels` ["blue" "green" "red" "yellow"]
Grippers	`gripper`	`size` [0.07 0.10]

4.3.2 Behaviour Modules

The robot controller for the foraging in Liu [42] was designed using the behaviour-based subsumption architecture [49]. Subsumption architecture presents a complex behaviour as consists of separate layers of simpler behavioural modules. Each layer works on individual goal independently. A higher layer has a higher priority and take precedence (subsumes) over lower layers.

The subsumption-based controller for the robot foraging is shown in Figure 4.4. Upon activation, each behaviour of the upper layers suppresses the lower layers to take control of the actuators. For instance, the *Depositing* behaviour has a higher precedence than the *Homing* behaviour and thus a robot is in the base with an object, it will proceed to deposit that object instead of still locating the base.

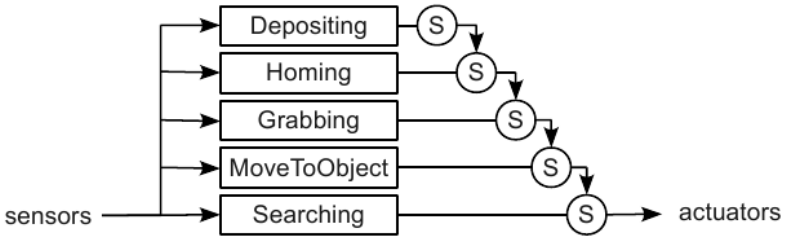


Figure 4.4: Subsumption control architecture for robot foraging. Note that there is another module *Avoidance* - not shown in the figure - that is triggered whenever obstacles are detected.

The behaviour modules are explained as follows:

- *Searching*: robot moves forward at a fixed speed, turning left and right at random intervals. It uses the camera to search for objects of interest in the arena;
- *MovingToObject*: when an object of interest is detected by the camera, the robot moves to the direction of the object. The robot estimates the angle between its heading and the direction of the object based on the information from the camera. If there are more than one objects, the robot chooses the closest one by comparing the pixels of the objects sensed by the camera: a closer object has more pixels;
- *Grabbing*: when a robot reaches the object, as indicated by the beam sensors on the grippers, it closes the grippers to grab the object;
- *Homing*: if an object is successfully grabbed, the robot moves to the base according to the information from the light sensors;
- *Depositing*: when the robot arrived in the base, it deposits the object in the base;
- *Avoidance*: at any time except during *Grabbing*, the robot avoids other robots and obstacles by examining the information from the IR sensors.

4.3.3 Arena Layout

The arena in which the robot swarm operates in the simulation is an octagonal shaped area of 10m $\times$ 10m with a circular base of 2m in diameter located at the centre (see Figure 4.5). The base is coloured green as a visual indicator of the area. A light post is located at the centre of the base. Objects can be placed anywhere in the arena but must be 1.5m from the centre of the base and 0.5m from the edges of the arena. Each object is an red-coloured square box of 0.05m in length.

4.3.4 Foraging Operation

To show the transition from one behaviour to another for foraging in simulation, Figure 4.6 is a graphical representation, using a state diagram, of a robot foraging. Each of the decomposed low-level behaviours in foraging is represented as a state in the diagram. Transitions between states are triggered by events perceived by a robot's sensors as well as initiated by the main program.

When the simulation starts, all robots depart from the base and the heading for each robot is calculated accordingly. With n robots, the direction for robot $\mathtt{R}i = i \times \frac{2\times\pi}{n}$. All fault-free robots move with a default speed of 0.15m.s^{-1}. The movement of a robot is controlled by two variables: the *turnrate* and the *speed*. A

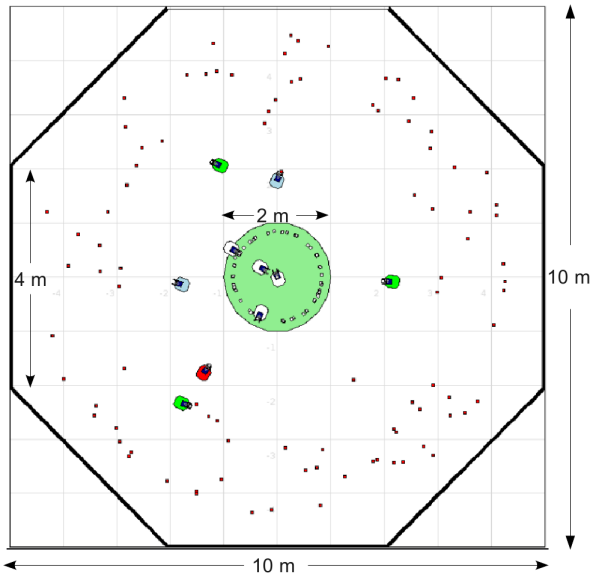


Figure 4.5: The dimension of the arena in which a foraging SRS operates is 10m × 10m octagonal shaped with a base of 2m in diameter located at the centre of the arena.

positive value for the turnrate causes the robot to turn to the left whilst a negative value causes the robot to turn to the right. Similarly, a positive value for the speed causes the robot to move forward whilst a negative value causes the robot to reverse.

A robot starts in the *Searching* state and randomly turns to left or right (up to 10°) to search for objects of interest. For this, a random number is generated at each time step between -10 and 10. If objects are perceived by the `blobfinder`, the robot transitions to *MovingToObject* state and moves toward the closest object. This is achieved by comparing the size of the objects detected by the `blobfinder` model. The closer is an object, the bigger is the size. The robot then moves towards the position of the object, as sensed by the `blobfinder`. The speed of the robot is slowed down according to the relative distance of the robot and the object.

The robot then transitions from the *MovingToObject* to the *Grabbing* state when it reaches the object. This is determined by whether an object is detected by both beam sensors. To grab the object, the robot's speed is slowed down to 0.01m.s^{-1} and the robot then closes its grippers. If the object is successfully grabbed, a transition to the *Homing* occurs and the robot navigates back to the base by following the intensity of a light source located at the base. If no light is detected by the light sensors, the robot makes a random turn to the left or right by 20°. If the highest light intensity is detected on the left, the robot moves to the left by 20°. If

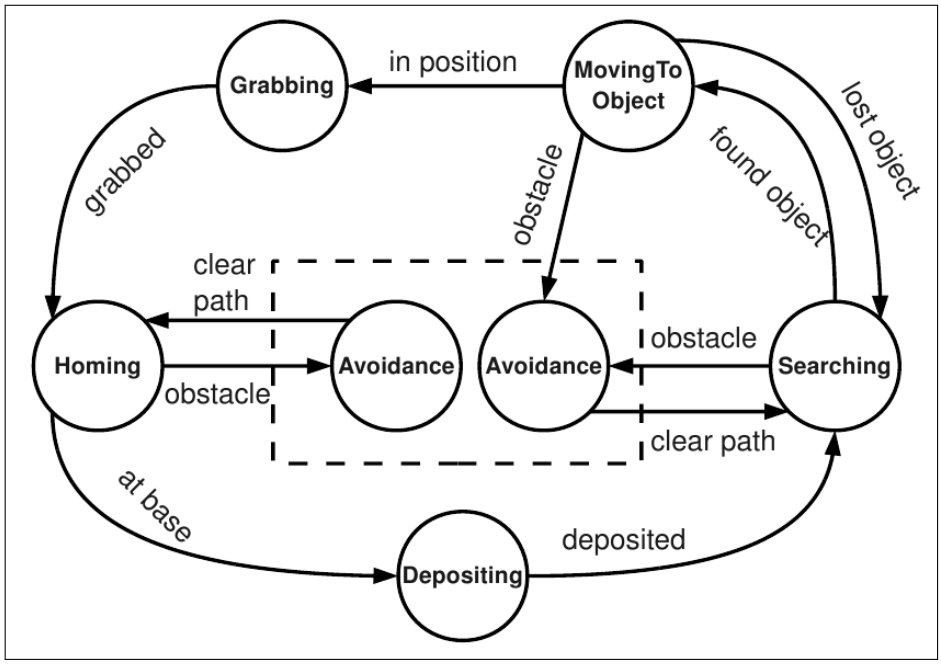


Figure 4.6: State diagram for robot foraging adapted from [42]. Note that before transitioning into the *Avoidance* state, the information regarding the current state is stored and will revert to it after obstacles have been avoided.

the highest light intensity is detected on the right, the robot moves to the right by 20°. In these three cases, the robot moves with a random speed between 0.00m.s^{-1} to 0.15m.s^{-1}. Otherwise, the robot moves forward with normal speed.

When a robot is in the base, by checking whether the position (the x and y coordinates returned by the *position* model) of the robot is within the base, a transition to the state *Depositing* is triggered. The robot opens its grippers and drops the object in the base. It then transitions to *Searching* state and resumes the foraging.

At any time outside the base, if a robot encounters obstacles (including other robots), a transition to the *Avoidance* state occurs. Before transition to the *Avoidance* state, the current state is saved so that the robot can resume its operation after avoiding the obstacles. The *Avoidance* state is triggered by the three `ranger` sensors. If an obstacle is detected on the right, the robot turns to the left by 35°. If an obstacle is detected on the left, the robot turns to the right by 35°. For these two cases, the speed of the robot is reduced to 0.005m.s^{-1}. If an obstacle is detected in the middle, the robot reverses by setting the speed to -0.01m.s^{-1} and randomly turns either left or right by 50°. For the cases where obstacles are detected in the middle and left or right, the robot reverses by setting the speed to -0.01m.s^{-1} and turns to left or right by 45°. If obstacles are detected in all directions, the robot stops (i.e. speed of 0.0 m.s^{-1}) and makes a random left or right 90° turn. Once the

obstacles have been avoided, the robot reverts back to the previous state.

It will be noted that the foraging SRS described here is a stripped down version of the one in Liu [42]. In Liu [42], the work is concern with the design and optimisation of interaction rules for energy efficiency. For the purpose of this work, significant alterations have been made to the original implementation by removing all interaction rules and irrelevant components to end up with a stripped down version with only basic foraging behaviours. The full source code and data can be found online [133].

Another alteration is the assignment of colours to robots for different states: light blue when searching for objects, light green when there is an object in grippers, white when in the base, and red when faulty. These colours are assigned as a visual indication of a robot's current state. In Figure 4.7, there are three fault-free robots with an object in their grippers, four robots are at the base, two fault-free robots are still searching for objects, and a faulty robot.

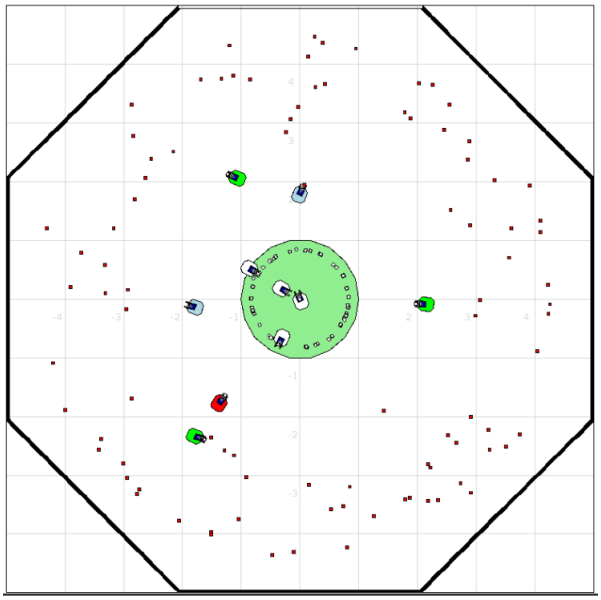


Figure 4.7: A snapshot of a running simulation with 10 robots. Based on the colour assignment: a robot with an object in the grippers is in light green, a robot at base is in white, a robot searching for an object is in light blue, and a faulty robot is in red.

The execution of each behaviour in each robot consumes a certain amount of energy. The energy consumption for each behaviour per second is based on those in Liu [42] and presented in Table 4.3. These values were estimated based on the relative energy usage of sensors and actuators used for each behaviour [42].

The distance travelled by each robot is calculated by simply multiplying the moving speed by time, which is every second. For example, if a robot moves with

Table 4.3: Energy usage for each behaviour.

Behaviour	Energy usage (per second)
Avoidance	0.9 unit
MovingToObject	0.8 unit
Grabbing	1.2 unit
Searching	0.8 unit
Homing	1.2 unit
Depositing	1.2 unit

normal speed of 0.15m.s^{-1} for 3s, then the distance travelled is 0.45m.

The robots will continuously carry out foraging until the end of the simulation. For all experiments in this thesis, each simulation lasts 20,000s. Periodically at every 250s, relevant data from each robot are extracted and output as csv files. The period of 250 is referred to as a *control cycle*, a term used by Christensen [9]. The extracted data (files) are tested on classifiers to detect errors and the performance of each classifier is evaluated.

For the experiments with fault injection, only one robot is injected with a fault to the wheels, as in the previous work by Canham et al. [10], Christensen et al. [11], Mokhtar et al. [13]. Faults to a single robot in dynamic environments, although a simple one, is important because if the problems is sufficiently complex that current state of the art approaches are shown to not work well, then they are unlikely to deal with more complex ones. In addition, by only considering a single robot failure, it is possible to provide a set of results more easily comprehensible to allow a more accurate understanding of the problem.

4.4 Fault Models of The Robot Wheels

Components on a robot may fail for a variety of reasons including wear-and-tear, power loss, and damaged circuitry connections. The failure of these components affect the ability of a robot to continue its task and thus may affect the overall swarm. Some failures have greater implications on the completion of task than others. In Winfield and Nembrini [1], a fault to the motors (wheels) on robots has the most significant effect on the ability of the robot swarm to complete a beacon taxis. Since a robot foraging relies on the ability of the robots to move to locate, collect, and transport objects to the base, fault to the wheels is suitable to establish the condition **E1** in which the effect a fault can be observed from the ability of the robot to forage.

Depending on whether a fault on the wheels of a robot occurs instantaneously

or gradually, and the extend (severity) of failure to the wheels, the fault can be of three types: $\texttt{P}_{\texttt{CP}}$ resulting in complete failure, $\texttt{P}_{\texttt{PT}}$ resulting in partial failure, and $\texttt{P}_{\texttt{GR}}$ resulting in gradual failure. Faults of the type $\texttt{P}_{\texttt{CP}}$ and $\texttt{P}_{\texttt{PT}}$ occur instantaneously such as those due to power loss and damaged circuitry. On the other hand, faults of the type $\texttt{P}_{\texttt{GR}}$ are faults that accumulate over time such as the gradual wear-and-tear of components.

The wheels on each robot might malfunction at any time. Faults of the type $\texttt{P}_{\texttt{CP}}$ are simulated in this work by setting the turnrate of the robot to 10° per time step whilst maintaining the normal speed causing the robot to turn left and move in circle. This fault results in complete failure as a robot with this fault is not able to continue collecting any more objects from the arena. This fault is the most damaging to the ability to forage compared to the other types of faults. Having said that, depending on the environment in which the SRS is deployed, inferring the presence of this fault might not be as easy. This can happen in a scenario in which the availability of objects in the arena is limited and approaching zero. In this scenario, a faulty robot cannot collect any objects because it is unable to do so. However, fault-free robots also might not collect any objects because there are limited objects in the arena. The $\texttt{P}_{\texttt{CP}}$ in this thesis is analogous to the motor failure in Winfield and Nembrini [1]. In their work, the authors refer to it as a partial failure of the wheels from a system-level perspective. Here, the fault is addressed from a component-level perspective. In other words, with $\texttt{P}_{\texttt{CP}}$ the component fails completely and permanently.

With $\texttt{P}_{\texttt{PT}}$, the fault of the wheels also occurs instantly as with $\texttt{P}_{\texttt{CP}}$. However, the fault does not result in a permanent failure but rather a partial failure which is less severe. In this work, this is presented as a case in which the wheels operate less efficiently. In simulation, this is achieved by an instantaneous reduction in the moving speed of wheels to a constant value 0.045m.s^{-1} from a normal speed of 0.15 m.s^{-1}. This fault is analogous to the *stuck-at-value* fault investigated in Mokhtar et al. [13].

Faults to the wheels can also occur gradually such as the case with gradual wear-and-tear of physical components. Gradual fault to the wheels $\texttt{P}_{\texttt{GR}}$ is simulated in this work by gradually reducing the moving speed of the wheels by 100×10^{-5} m.s^{-2} from a normal speed of 0.15 m.s^{-1}, and it will eventually stop moving.

For the experiments in this thesis, these faults are injected independently to a single robot at control cycle 20 and persist until the end of a simulation. Thus, the motor speed of a robot S(t) at a time instance t is influence by the fault $F(f)$ (refer Eq. 4.1 and Eq. 4.2). Note that the reason why only the permanent faults (i.e.

faults that persist till the end of simulation after initial injection) are considered is that transient faults (i.e. faults that are on and off for a comparatively short intervals) do not appear to significantly affect the completion of a given task and can generally be tolerated by the redundancy of robots as demonstrated in Winfield and Nembrini [1]. Another reason for experimenting with permanent fault is to test the proposed detection scheme over many control cycles to ensure consistent positive detection is achieved. Note that in practise, after an error is detected the subsequent fault diagnosis and recovery *should* rectify the fault and thus the fault will not be permanent.

$$S(t) = \begin{cases} F(f), & \text{if } t > ts, f \in \{\texttt{P}_{\texttt{CP}}, \texttt{P}_{\texttt{PT}}, \texttt{P}_{\texttt{GR}}, \text{fault-free}\} \\ 0.15 \text{ m.s}^{-1}, & \text{if } t \leq ts \end{cases} \tag{4.1}$$

$$F(f) = \begin{cases} 0.15 \text{ m.s}^{-1}, & \text{if } f = \{\texttt{P}_{\texttt{CP}}, \text{fault-free}\}, \\ 0.045 \text{ m.s}^{-1}, & \text{if } f = \texttt{P}_{\texttt{PT}}, \\ S(t-1) - 100 \times 10^{-5} \text{ m.s}^{-1}, & \text{if } f = \texttt{P}_{\texttt{GR}}. \end{cases} \tag{4.2}$$

A summary of the fault models to the wheels to establish **E1** is tabulated in Table 4.4.

Table 4.4: Summary of the fault models of the wheels.

Fault	Fault occurrence	Effect on the wheels
$\texttt{P}_{\texttt{CP}}$	instantaneous	turnrate equals to 10°, robot moves in circle
$\texttt{P}_{\texttt{PT}}$	instantaneous	speed is slowed instantly to 0.045 m.s^{-1}
$\texttt{P}_{\texttt{GR}}$	gradual	speed is slowed gradually by 100 $\times 10^{-5}$ m.s^{-2} until it stops completely (i.e. speed = 0.00 m.s^{-1})

4.5 Models of Uncertain Time-Varying Environments

In the arena, objects are placed at random locations but outside of the base at a rate referred to as the object replenishing rate (`OPR`). The random number generator used is `gsl_rng_mt19937` in GSL-GNU Scientific Library [134] initialised with a random seed. The default value for `OPR` is 0.10. An `OPR` of 0.10 means that the probability of adding an object at every second is 0.10.

The performance of individual robots in the foraging task can be evaluated based on the number of objects collected at each control cycle as well as during their lifetime (duration of a simulation). Therefore, under a normal operating condition with a constant `OPR`, it is assumed that a fault-free robot should collect

roughly the same amounts of objects as other fault-free robots. If the number of objects by one robot is significantly different from other robots, it may be caused by the presence of fault. There are other conditions that may also influence this performance metric.

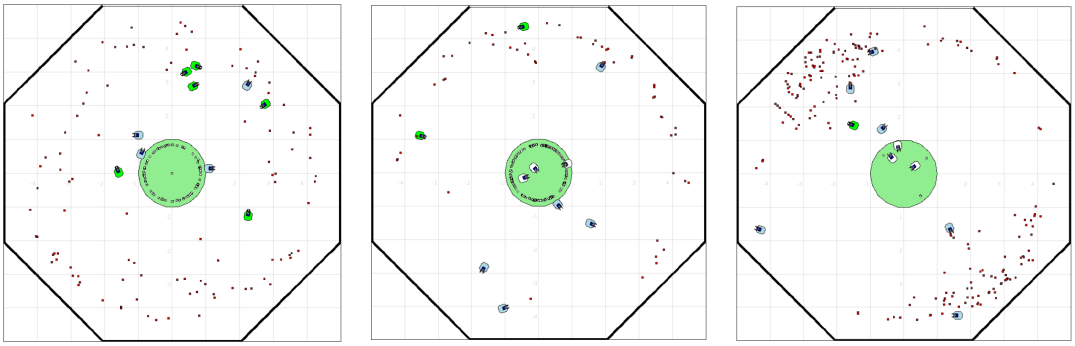
In this work, two of such conditions are considered. They are the concentration (or the availability) and the physical placement of objects. These two conditions are selected because both are concerned with the locations of target objects and thus influence the movement and navigation of robots (wheels) to those locations. By altering these two factors over different periods of time creates time-varying conditions that causes the performance of foraging for the robots change accordingly and thus enable the condition **E2** to be established.

From these two factors, it is possible to simulate three different scenarios in which the robot swarm operates: constant OPR (CST), varying OPR (V_{OPR}), and varying object distribution (V_{ODS}). In the V_{OPR} and V_{ODS} scenarios, the environments are time-varying (dynamic) whilst in the CST scenario it is non-dynamic (with reference to the OPR and unbiased object distribution).

As well as the availability and placement of target objects, there are other conditions that can also affect the swarm foraging such as the smoothness of terrains, the consistency of air that can affect a robot's vision, the size of target objects that may be too big for a robot's grippers, and so on. However, these conditions are not considered in this thesis because it is nearly impossible to consider every environmental factor that affects the swarm foraging and a choice was made to focus on availability and placements of objects. In addition, some of these conditions (such as the air quality) are very difficult, if not impossible, to simulate. In theory, if the proposed approach works on the selected conditions, it should also work for other environmental conditions that satisfy **E2**.

In the CST scenario, the OPR is fixed at 0.10 with one hundred initial objects. Then, objects are inserted randomly at every time step as shown in Figure 4.8(a). In the figure, objects are inserted randomly at 0.5m outside the perimeter of the base and 0.5m away from the walls. This CST scenario can be used as a baseline for comparison of the error-detection ability of various classifiers with the other two scenarios.

In the V_{OPR} scenario, the number of objects in the environment changes according to the OPR. By changing the OPR, it affects the number of objects put back into the arena and thus the number of objects than can be collected by each robot in a control cycle. The V_{OPR} scenario is simulated by changing the OPR alternately between 0.10 and 0.025. Figure 4.8(b) is a snapshot of the distribution of object with a OPR of 0.025. Comparing this with the CST scenario in Figure 4.8(a), the



(a) CST, OPR=0.10. At control cycle t=3.

(b) V_{OPR}, OPR=0.025. At control cycle t = 30.

(c) V_{ODS}, new objects biased to top left and bottom right

Figure 4.8: Three types of operational environment for the SRS.

amount of objects in the arena is significantly less.

Recall in Section 4.4 that with P_{CP}, a faulty robot cannot continue with foraging and thus no objects can be collected within each control cycle. In the CST scenario, this detection of the error is straightforward as the fault-free robots would have collected significantly more than zero object. However, in the V_{OPR} scenario, if the amount of objects in the arena is very small and approaching zero, the fault-free robots would have also collected a number of objects that is close to zero. In this situation, it is nearly impossible to infer the presence of the P_{CP} based on the number of objects collected. For this reason, other data (such as distance travelled and energy used) are also used in conjunction with the number of objects collected to infer the presence of a fault.

Finally, in the V_{ODS} scenario, the physical placement of a new object in the arena is biased to particular regions in the arena. The effect of this scenario is that the concentration of objects at particular region forces a robot to a longer time to reach the region if it was at another region and eventually results in less objects being collected. However, the influence on the object collection should be less when compared to the one in the V_{OPR} scenario. In simulation, the V_{ODS} scenario is simulated by biasing the distribution of new objects in the arena between top left and bottom right regions as shown in Figure 4.8(c). Note that the OPR in this scenario is the default one with a value of 0.10.

For the V_{OPR} and V_{ODS} scenarios in this thesis, the changes in the environment are activated at different time intervals. For example, if there are two cycles of environmental changes in the V_{OPR} scenario, then simulation can be logically divided into four time slots and the changes are activated alternately between each slot as depicted in Figure 4.9.

A summary of the time-varying environments to establish **E2** is tabulated in Table 4.5.

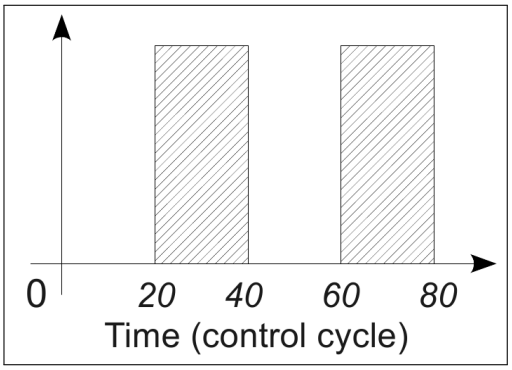


Figure 4.9: Time slots to simulate time-varying changes in the environment. At the time slots in grey, the V_{OPR} and V_{ODS} scenarios are simulated.

Table 4.5: Summary of scenarios with time-varying availability and physical distribution of new objects in the arena.

Scenario	OPR	Distribution of objects
CST	constant at 0.10	randomly distributed
V_{OPR}	alternatively between 0.10, 0.025	randomly distributed
V_{ODS}	constant at 0.10	between top left, bottom right

4.6 Summary

This chapter presented the specific context and experimental testbed in which the research question in this thesis will be investigated. In Section 4.1, a taxonomy of robot foraging as compiled in Winfield [63] is presented to give a general overview on the various research topics involved. Also included in this section is the motivation to extend the robot foraging in Liu [42] as the experimental testbed.

In Section 4.2 the simulator used for the experiments namely the Player/Stage [130] is presented. Then, Section 4.3 describes the physical specifications of the forager robot, the behaviour modules for the robot controller, and the arena layout.

In Section 4.4, three fault models were introduced to established the first condition required to investigate the research question namely **E1** in which a fault has an observable effect on the foraging behaviour. This is followed by Section 4.5 that introduces the models of time-varying environment that also affect the foraging behaviour of robots to establish condition **E2**.

The experimental setup in this chapter specified the conditions necessary so that the research question can be investigated in an unbiased and fair manner. To be able to support this claim, the next chapter will investigate through experi-

mental analysis to determine whether **E1** and **E2** hold.

CHAPTER

FIVE

Experimental Analysis of Robot Foraging

In the context of the experimental testbed, this chapter presents experimental analysis to support the conditions (**E1** and **E2**) specified in the previous chapter. In Section 5.1, an analysis is presented on the effects of swarm size on the foraging performance as measured with the number of objects collected by the whole swarm. This is to ensure that the right number of robots are placed in the 10m $\times$ 10m arena without overcrowding. Section 5.2 then proceeds to investigate the consistency of data amongst the robots within the swarm, according to the percentage relative standard deviation, for four different control cycles. Section 5.3 then examines **E1** by analysing the manisfestation of the faults on the foraging performance. Finally, Section 5.4 is the analysis of the effects of the different environmental conditions on the foraging performance to examine **E2**.

5.1 The Effects of Swarm Size on the Foraging Performance

Generally, it is assumed that in swarm robotics an increase in the size of a robot swarm will also lead to a proportional increase in the performance [42]. Thus, more robots engaged in foraging will lead to more objects being collected. Whilst this may be true in an unbounded space with unlimited energy and unlimited object of interest, it is often not the case in swarm robotics research due to various physical constraints [42]. Examples of these constraints include the size of

the arena, availability of objects in the arena, physical distribution of objects in the arena, the amount of energy available, and the interference between robots as each robot avoids bumping into each other. Thus, in many cases, simply increasing the number of robots in the system may not even result in the completion of a given task or when it is completed it may take longer than expected. Therefore, determining a suitable swarm size in a fixed arena is important.

5.1.1 Experimental Setup

Experimental Objective: To investigate the number of robots that can be placed in the arena without (negatively) affecting the overall foraging performance of the robot swarm, to choose a suitable swarm size for subsequent experiments.

The aim of this experiment is to find out the number of robots that can be placed in the arena without causing the robots to spent most of the time avoiding each other as the arena became overcrowded. The effect of overcrowding can be inferred by analysing the foraging performance (total number of objects collected) of the robot swarm as the swarm size is increased. In principle, if a swarm size is increased from 5 to 10 robots - an increase of 100%, the performance should also increase by 100%. This is assuming that the amount of objects in the arena is significantly more than the size of the swarm and there are not other interfering factors. Therefore, if an increase in the swarm size is not in proportion to the increase on the performance, this can signify the occurence of an overcrowding situation.

Results from this experiment can be used to determine an appropriate swarm size for the rest of the work and also to demonstrate that having many redundant robots does not necessary leads to improved performance for the robot swarm in this thesis. Also, it helps to support **E1** and **E2** in that it eliminates overcrowding as a factor that affects the foraging performance.

In this experiment, a series of simulations were carried out for a robot swarm with 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 75, and 100 robots. All robots are fault-free in the `CST` scenario (refer Section 4.5 for details of this scenario), and data (foraging performance) are analysed at every 250s.

The rate of adding a new object per second, the `OPR`, is dependent on the number of robots in the swarm (`OPR` = $\frac{number\ of\ robots}{10}$). For example, for 10 robots, the `OPR` is set to 0.10. For 20 robots, the `OPR` is set to 0.20. With 100 initial objects in the arena and respective `OPR`, the amount of objects in the arena is always more than the number of robots at any time. Therefore, this can eliminate the amount of objects in the arena as a factor that affects the foraging performance of the robot

swarm. Instead, if a change occurs in the foraging performance it is a result of the increase in the swarm size.

To compare the effect of swarm size N on the foraging performance of the robot swarm, results are plotted in graphs.

5.1.2 Results

For each swarm size, twenty repeated runs were performed and the median of the foraging performance at each control cycle calculated.

Figure 5.1 plots the medians of the foraging performance (total number of objects collected by the robot swarm during the control cycle) during the simulations. In the figure, a swarm of 5 robots collected approximately 20 objects at each control cycle. When the swarm is increased to 10 robots, the objects collected also increases to nearly 40 units. This is nearly a 100% increase, which is a direct correlation with the increase of swarm size. From this result, it is safe to assume that the problem of overcrowding has not been observed.

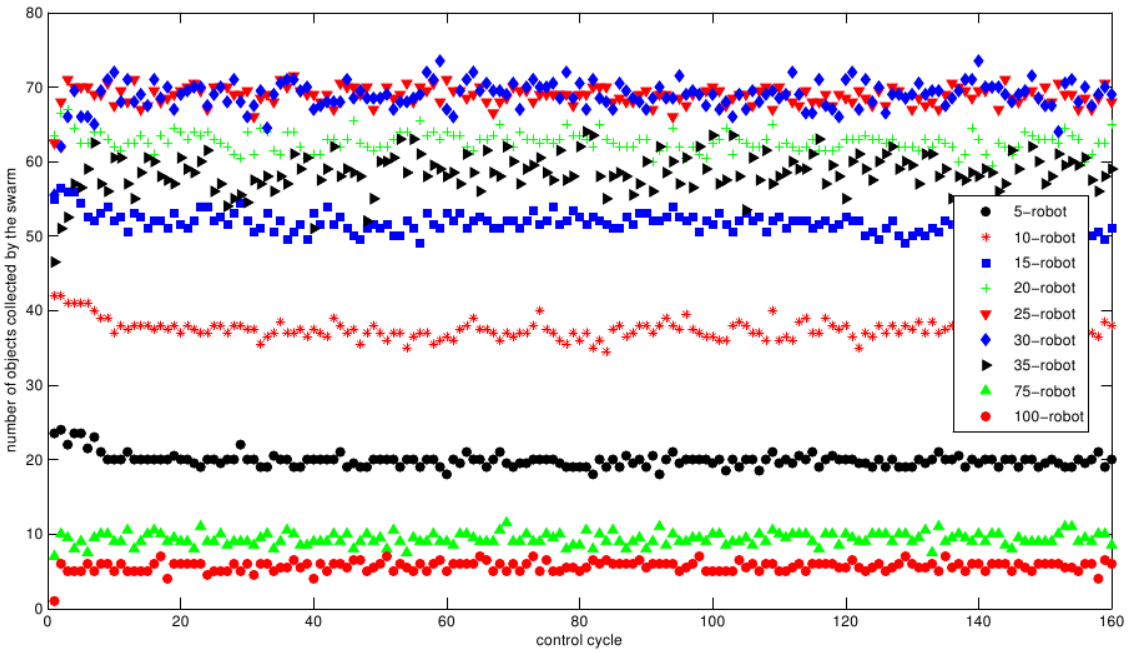


Figure 5.1: Graph showing the total number of objects collected by a robot swarm of different sizes. A steady increase on the number of objects collected is observed as the swarm size is increased from 5 to 10, 15, robots. However, an adverse effect is observed with a swarm of 35, 75 and 100 robots.

Under the same assumption, when the robot swarm is increased to 15 robots, approximately 60 units is expected to be collected. However, such a result has not been observed in the figure. Instead, the number is just slightly above 50 units.

When the swarm is further increased to 20 robots, the number of objects collected is only increased by 10 units. When the swarm size is 25 and 30 robots, the number of objects collected are roughly the same. This is an interesting observation because it shows that a point is reached where further increasing the swarm size does not result in an increased performance. Therefore, it is reasonable to assume that the arena has become overcrowded. When the swarm size is increased to 35 robots, an adverse effect of decreasing number of object collected is observed.

To further *stress-test* the system to investigate the correlation (or lack of positive correlation) between the size of the swarm and the number of objects collected, the swarm is drastically increased to 75 and 100 robots. As expected, significantly less number of objects is collected. With a swarm size of 75 robots, the number of objects collected is decreased to approximately 10 units, and further reduced to about 5 units with a swarm of 100 robots. This result conforms to the one reported in Liu [42], albeit the performance measured there is the energy efficiency of the robot swarm.

The results in Figure 5.1 show that having more robots does not always guarantee an improvement in the foraging performance. Therefore, a qualitative decision was made to use a swarm size of 10 robots for subsequent experiments. A swarm of 10 robots is appropriate for the work in this thesis for the following reasons. Firstly, it results in a stable system in which the object collection does not fluctuate much from one control cycle to another. Secondly, because the swarm is not overcrowded it eliminates the size of the swarm as a factor that influences the foraging performance. This indirectly helps to provide evidence to support **E1** and **E2**. Thirdly, it provides flexibility in that more robots can be added, if required, for potential future work to investigate other aspects of error detection not investigated in this thesis such as the scalability of detection algorithms, and failures of multiple robots.

5.2 The Effects of the Control Cycle on the Consistency of Data

The granularity (i.e. the length) of a control cycle determines the absolute values of the data used for detection. In order for the data to be meaningful for the subsequent error detection process, it is relevant to choose a suitable control cycle. For example, in a foraging task, a robot may collect up to two objects in 50s, three in 100s, five in 200s, and six in 250s. In this case, the probability of collecting a particular number of objects differs with respect to the granularity of a control

cycle. If a fault causes a robot to stop functioning and unable to collect any object within a control cycle, the presence of a fault can be inferred with a greater confidence at 250s rather than after 50s. However, it also means 250s have elapsed before the presence of a fault can be inferred. Hence, the granularity of a control cycle can influence the ability to positively identify an error as well as the time taken to report an error.

A short control cycle may result in a faster response but it is likely to result in false positives. This is because with a short control cycle (short relative to the time to carry out a given task), a small change in the data due to the stochasticity of the system and not due to faults can trigger a false detection. Inversely, a long control cycle may result in less false positives but the response is slower. Thus, a trade-off between the number of false positives and the response time is generally involved. However, unless the desired performance (e.g. false positive rate, response time) was explicitly specified, this is a non-trivial task.

5.2.1 Experimental Setup

Experimental Objective: To investigate the consistency of data amongst the robots with control cycles of different granularity, to choose a control cycle that produce the most consistent data to facilitate the detection of errors.

To help in determining an appropriate control cycle's length, the patterns of foraging performance (the data) for each control cycle are presented as graphs of the percentage relative standard deviation (pRSD) over time. RSD (relative standard deviation), also known as *coefficient of variance*, is a useful measure to compare the variations between different measurements of varying absolute magnitude. When comparing two measurements with pRSD, the measurement with a lower pRSD is considered more precise, i.e. less variations, and thus greater consistency. Therefore, the granularity of a control cycle that provides a more consistent data can be determined by examining the pattern of data expressed with pRSD. RSD expressed as percentage, i.e. pRSD, is a measurement of standard deviation (σ) over the mean ($\bar{x}$) and multiplying by 100 (Eq. 5.1).

$$pRSD = \frac{\sigma}{\bar{x}} \times 100 \tag{5.1}$$

From the findings in Section 5.1, a robot swarm of 10 robots is placed in the arena in this experiment. All robots are fault-free in a `CST` scenario with a `OPR` of 0.10, and 100 initial objects.

In this experiment, the data to be analysed is the number of objects collected

by each robot for control cycles of 50s, 100s, 200s, and 250s. Notice that the simulation duration (20,000s) is an integer multiple of the length of each control cycle and it is also a multiple of 50s which is approximately (50s $\pm$ 10s) the average time taken for a robot to to collect an object (see Figure 5.1).

To compare the consistency of data amongst the robots with different granularities of control cycles, the results are plotted as graphs for the pRSD against different control cycles.

5.2.2 Results

Figure 5.2 plots the pRSD between all robots at each control cycle for a single simulation run. In the figure, it becomes apparent that as the length of a control cycle increases, the value of the pRSD decreases. For example, with a control cycle of 50s the pRSD fluctuates between 50% and 80%. With a control cycle of 100s, the pRSD reduces to between 30% and 50%. This trend continues for control cycles of 200s and 250s with pRSDs around 20%. In other words, the differences between the number of objects collected amongst the robots are bigger with a control cycle of 50s compared to 250s. Therefore, it appears that the longer a control cycle, the more consistent (less deviation) is the data from all robots. In this experiment, a control cycle of 250s produces the smallest pRSDs, and thus the data is the least deviated between the robots. From this result, a control cycle of 250s is chosen for subsequent experiments.

Results from Figure 5.2 indirectly also revealed something about the nature of data. By cross checking the values for the number of objects collected by each robot and the resulting pRSD, a difference of one object at different control cycle's granularity produces different pRSD. For example, for a control cycle of 50s, a difference of one object results in a pRSD of about 100%. For a control cycle of 250s, the same difference of 1 unit is now only 20% in pRSD. Therefore, a control cycle of 250s provides a greater confidence that the foraging performance amongst the robots in a local neighbourhood is similar to those measured by the pRSD.

One might argue that for a more consistent data, a control cycle of longer than 250s can be used. However, as mentioned in the previous paragraph, a longer control cycle means a longer time to detect the presence of a fault. This has a cascading effect on the proceeding stages in an EDR to recover from a fault. Therefore, there is a tradeoff between the confidence in a positive detection and the time to detect and error. In practise, if the response time is relatively less important the a positive detection, then a control cycle of a longer period can be used.

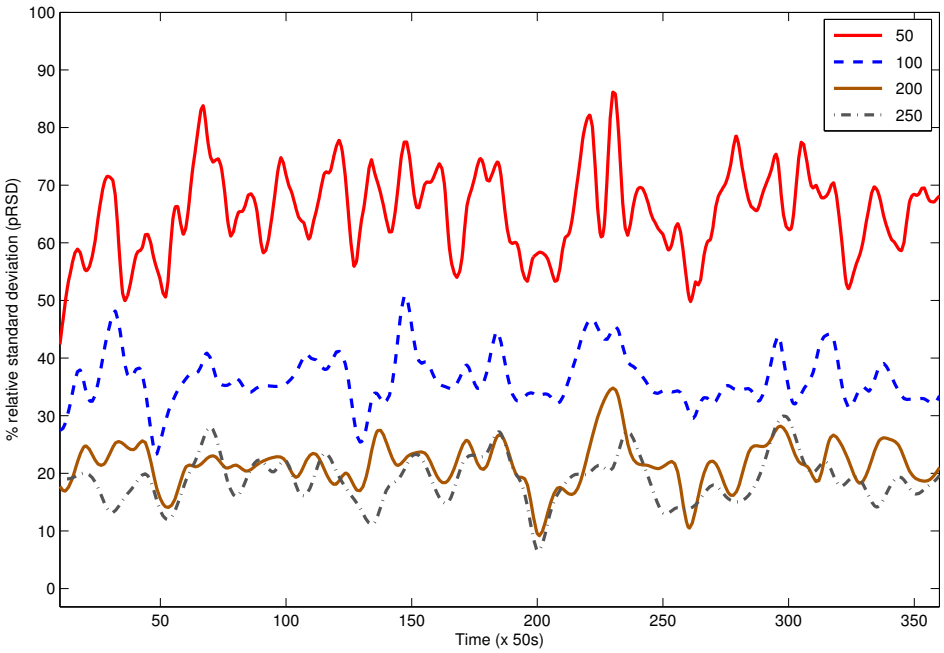


Figure 5.2: Graphs for the percentage relative standard deviation on the number of objects collected by each robot with a control cycle of 50s, 100s, 200s, and 250s. The graphs are fitted with Smoothing Spline in Matlab.

Another possible implementation with respect to the control cycle is that instead of having a fixed control cycle, the system can utilise multiple control cycles of different granularity. This is often referred to as a multi-resolution approach and could potentially avoid the problem of slower response times. For example, instead of having a fixed control cycle of 250s, data can be extracted at multiple control cycles such as at 50s, 100s, 200s, and 250s. However, having multi-resolution requires the computation and storage of data for every resolution which in turn imposes a heavier resource usage. Nevertheless, this is an interesting approach that could be further pursued in future research but will not be explored further in this thesis.

5.3 The Occurrence of Faults Through Observation

Different fault models of the wheels may have different effects on the robots. By analysing the foraging performance of the robots as indicated by the number of objects collected in a control cycle, the presence of a fault may be inferred. Although the results from the analysis may not be sufficient to identify which fault model is present, it should be sufficient to determine whether **E1** is satisfied.

5.3.1 Experimental Setup

Experimental Objective: To analyse the occurrence of faults of the wheels on a robot by observing the changes to the ability of the robot to forage.

To establish the effects of the faults described in Section 4.4 on the ability to forage, this experiment analyses the differences between the number of objects collected by robot in a fault-free and faulty conditions. Ten robots are placed in the 10m $\times$ 10m octagonal-shaped arena with a default `OPR` of 0.10. Every 250s (control cycle), the number of objects collected within that period by each robot is extracted. For this experiment, the robot swarm operates in a `CST` scenario in which the `OPR` is constant and new objects are randomly placed within the arena.

Each model of faults of the wheels in Section 4.4 is injected independently into a single robot at 5000s (control cycle 20) and the fault persists until the end of the simulation. In this experiment, the $\mathtt{P}_{\mathtt{PT}}$ is set to 45×10^{-3} m.s^{-1} and the $\mathtt{P}_{\mathtt{GR}}$ is set to 100×10^{-5} m.s^{-2}. For each fault model, twenty repeated runs were conducted.

To compare the resulting occurrence of faults, the results are plotted in graphs.

5.3.2 Results

Figure 5.3 shows the results (the median) for the number of objects collected by a robot when it is fault-free and when a fault was injected to its wheels. A vertical dotted line is drawn at control cycle 20 (5000s) to mark the fault injection time.

In general, a fault-free robot collects about 4 objects in the `CST` scenario. This can be seen from the graph labelled `fault-free` as well from the first to the twentieth control cycles. Once a fault has been injected to the wheels, a decrease in the ability of the robot to collect objects is observed.

Comparing the number of objects collected by the robot after the faults were injected in Figure 5.3, it is apparent that different fault models affect the ability of a robot to forage differently. With $\mathtt{P}_{\mathtt{CP}}$, the fault causes the robot to be unable to collect any more objects. Referring to the graph labelled $\mathtt{P}_{\mathtt{CP}}$, after control cycle 20 no more objects were collected. The effects of $\mathtt{P}_{\mathtt{CP}}$ on the ability of a robot to forage is straightforward and expected.

With $\mathtt{P}_{\mathtt{PT}}$ of 45×10^{-3} m.s^{-1}, the number of objects collected reduces to between 1 and 2 units. Again, this is expected as a slower movement means that less objects can be collected within the same amount of time. In a `CST` scenario, the number of objects (*obj*) that can be collected by a robot with $\mathtt{P}_{\mathtt{PT}}$ can be roughly

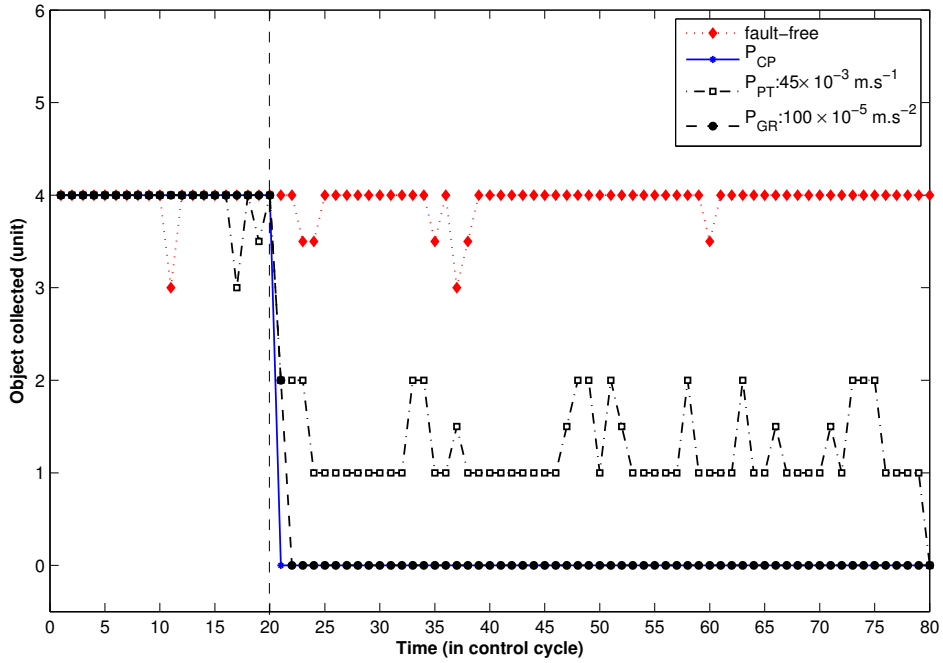


Figure 5.3: Graph for the number of objects collected by a robot in a fault-free state and when $\mathtt{P}_{\mathtt{CP}}$, $\mathtt{P}_{\mathtt{PT}}$, and $\mathtt{P}_{\mathtt{GR}}$ were injected to the wheels.

approximated according to Eq. 5.2.

$$obj \approx \left\lfloor \left(\frac{2}{(arena\ width + arena\ height)} \times P_{PT} \times 250 \right\rceil \right. \tag{5.2}$$

For example, for a robot with $\mathtt{P}_{\mathtt{PT}}$ of 45×10^{-3} m.s^{-1}, $obj \approx \lfloor (\frac{2}{10+10} \times 45 \times 10^{-3} \times 250) \rceil \approx 1$.

Referring to Figure 5.3, a $\mathtt{P}_{\mathtt{GR}}$ of 100×10^{-5} m.s^{-2} produces a similar effect on the ability of a robot to forage as a $\mathtt{P}_{\mathtt{CP}}$. A $\mathtt{P}_{\mathtt{GR}}$ also causes a robot to eventually stop collecting any more objects. However, the difference is that with a $\mathtt{P}_{\mathtt{GR}}$, a slightly longer time was taken for the robot to be unable to move to continue with foraging. In the figure, a $\mathtt{P}_{\mathtt{GR}}$ of 100×10^{-5} m.s^{-2} causes the robot to stop completely after one control cycle (250s). In fact, the robot stops after 150s.

From the results so far, it appears that it is easy to infer the presence of the faults to the wheels based on the number of objects collected in a CST scenario. A basic thresholding technique should be sufficient. For example, a linear classifier that specifies an error as any observation with the number of collected objects below 2.5 units. With this classifier, all errors due to the faults can be detected. However, as mentioned previously, such a straightforward observation is uncommon for robot swarms in dynamic environments.

From this experiment, results have shown that different faults are manifested differently on the data and thus establishes **E1**. The effects of $\mathtt{P}_{\mathtt{CP}}$ and $\mathtt{P}_{\mathtt{GR}}$ are more apparent when compared to $\mathtt{P}_{\mathtt{PT}}$. Also, the difference in the data before and after the faults are injected is apparent in a `CST` scenario. Thus, these changes can be detected relatively easily. However, if the data is also influenced by the changes in the environment, then it might not be that straightforward. Therefore, the experiment in the next section will look at how different environmental conditions affects the behaviour of robots.

5.4 The Effects of the Operational Environment on the Data

As described in Section 4.5, a robot swarm deployed in real-world often has to face a variety of challenges in particular those due to interference from a variety of sources. On top of that, the environment in which the robot swarm operates can also affect the performance of the robot swarm. For the foraging robot swarm in this thesis, the performance is measured on the number of objects collected within a control cycle. Therefore, this experiment simulates two environments in which the concentration of objects and physical placement of objects are biased. This examines how these changes will affect the performance of the robots and thus provides evidence to determine whether **E2** is satisfied.

5.4.1 Experimental Setup

Experimental Objective: To examine how different environment in which a robot swarm operates can affect the number of objects that can be collected by the robots.

In this experiment, a robot swarm of 10 fault-free robots is placed in the arena. Every 250s (control cycle), the number of objects collected by each robot are extracted. The robot swarm is independently placed in the environments as specified in Section 4.5.

The details for each scenario is as follows. In the `CST` scenario, the environment is non-dynamic with a default `OPR` of 0.10 and new objects are uniformly and randomly placed within the arena. In the $\mathtt{V}_{\mathtt{OPR}}$ scenario, the `OPR` changes alternatively between the default value of 0.10 and 0.025 at two intervals: at control cycles [20, 40), and control cycles [60, 80). This means that the `OPR` is 0.10 at [1, 20) and [40, 60), and 0.025 at [20, 40) and [60, 80). This setting creates a situation in

which the concentration of objects in the arena is time-varying. This same setting is also applied to the $\mathtt{V}_{\mathtt{ODS}}$ scenario in which at control cycles [1, 20) and [40, 60) new objects are randomly placed in the arena. At control cycles [20, 40) and [60, 80), new objects are not randomly distributed but instead placed at the top left and bottom right regions in the arena (see Figure 4.8(c)).

For each scenario, twenty repeated runs were performed. The results are plotted as graphs of the median number of objects collected by each robot throughout the simulation.

5.4.2 Results

Figure 5.4 shows the results (the median) on the number of objects collected by each robot at each control cycle under different environmental conditions. Vertical lines were drawn at various control cycles to indicate the time when the changes are activated and when they terminate.

The figure shows that for the CST scenario each robot collects about 4 objects at each control cycle throughout the simulation. However, when there are changes to the concentration and physical distribution of the objects in the arena, the number of objects collected also changes and thus providing evidence to support **E2**.

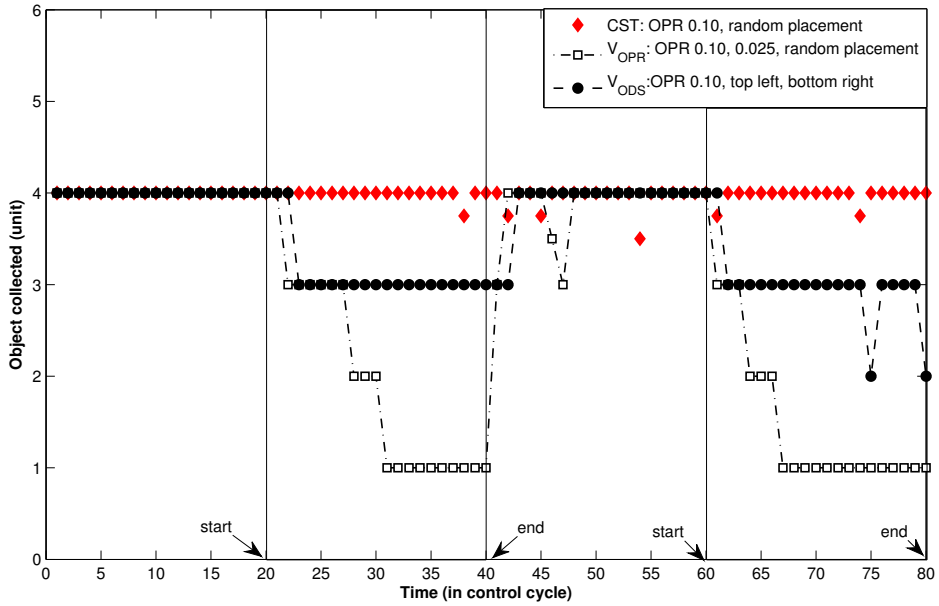


Figure 5.4: Graphs on the number of objects collected by fault-free robots in the CST, $\mathtt{V}_{\mathtt{OPR}}$, and $\mathtt{V}_{\mathtt{ODS}}$.

In a $\mathtt{V}_{\mathtt{OPR}}$ scenario, the number of objects collected by the robots starts to drop

immediately after the OPR was changed to 0.025. This is evident from the drop in the object collected of 4 units before the changes (control cycle 1 to 20) to 3 units at control cycle 22. This trend of a reduced performance (object collection) continues until control cycle 31. From control cycle 31 onwards, the value remained constant at 1 unit until the OPR switches back to 0.10 at control cycle 40. Then at control cycle 41, the objects collected increases back to 3 units and eventually back to 4 units one control cycle later. The same trend is observed when the OPR switches to 0.025 again at control cycle 60.

The effect of varying the concentration of objects in the arena, as simulated by varying the OPR, to the performance of foraging is direct and apparent. With less objects in the arena, less objects can be collected by each robot. However, if a robot is faulty when operated in the V_{OPR} environment, it is a challenging task to infer the presence of the fault based on the data. To illustrate this point, Figure 5.5 shows the overlapping in observations for the number of objects collected by fault-free robots in the V_{OPR} scenario and faulty robots in a CST scenario. From the graphs, it is apparent that the number of objects collected by fault-free robots in the V_{OPR} scenario is almost exactly the same with faulty-robots in particular at control cycles [31, 40] and [67, 80]. This observation shows that to infer the presence of a fault in the V_{OPR} scenario is difficult and thus worthy of further investigations.

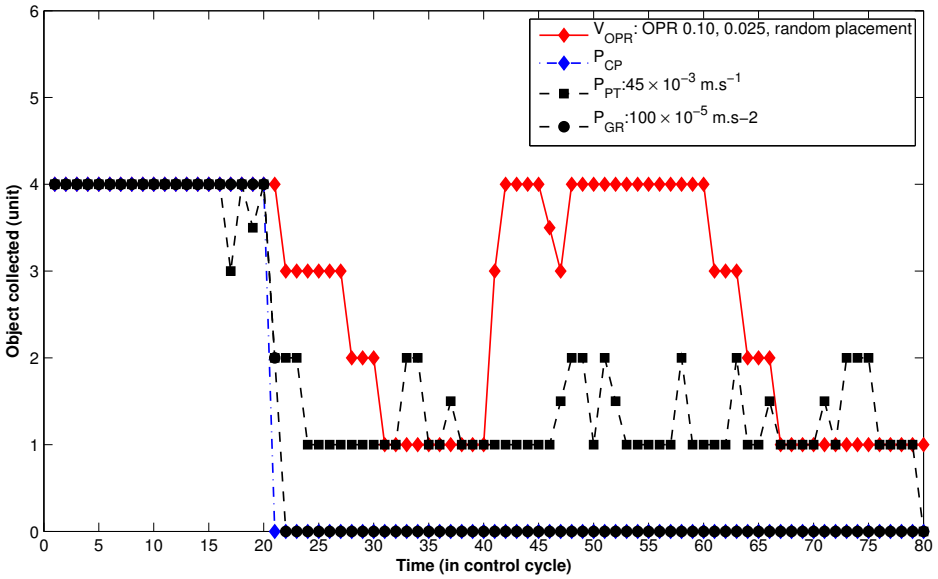


Figure 5.5: Overlapping of observations on the number of objects collected by fault-free robots in the V_{OPR} scenario and faulty robots in a CST scenario.

Referring to Figure 5.4, it is also observed that the changes to the distribution of objects in the arena as simulated by the V_{ODS} scenario have a lesser effect on the

number of objects that can be collected by the fault-free robots when compared to the $\mathtt{V}_{\mathtt{OPR}}$ scenario. When the changes in the $\mathtt{V}_{\mathtt{ODS}}$ scenario were activated for the two time intervals, the differences in the object collection by robots is only around one to two units. Therefore, to infer the presence of a fault in the $\mathtt{V}_{\mathtt{ODS}}$ scenario may be easier when compared to the $\mathtt{V}_{\mathtt{OPR}}$ scenario as illustrated in Figure 5.6.

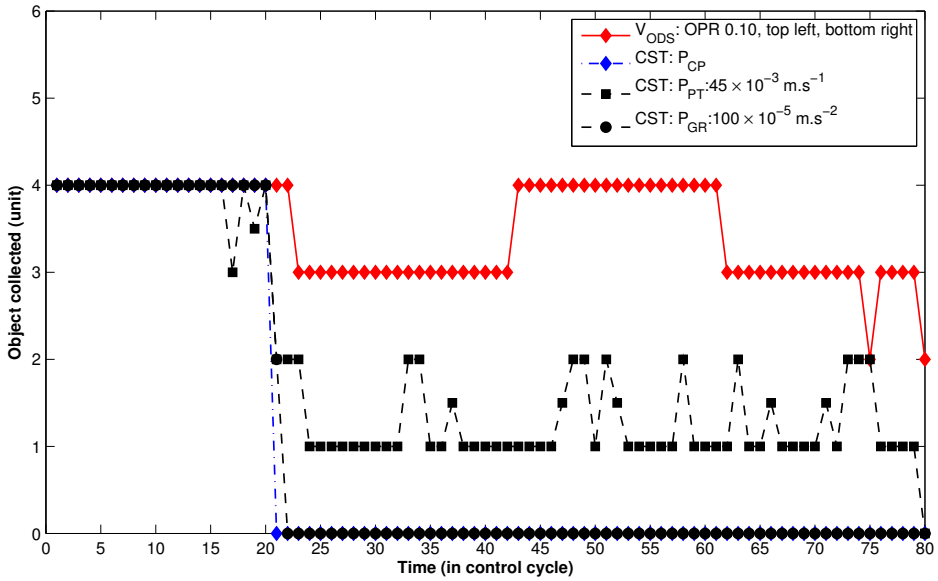


Figure 5.6: No overlapping of observations on the number of objects collected by fault-free robots in the $\mathtt{V}_{\mathtt{ODS}}$ scenario and faulty robots in a `CST` scenario.

5.5 Summary

This chapter presented analyses on the robot foraging simulations as part of establishing the experimental testbed. In Section 5.1, an analysis of the effect of swarm size on the foraging performance (the number of objects that can be collected by the whole swarm) is presented. The results from the experiment is consistent with those in Liu [42] that an increase in the number of robots in a swarm does not guarantee an increase in the performance. The results show that for an octagonal-shaped arena of 10m $\times$ 10m, a swarm size of 10 robots is suitable. With this size, the robot swarm is not overcrowded and thus eliminates it as a factor that influences the foraging performance. As such, it helps to establish **E1** and **E2** in Sections 5.3 and 4.4. For this reason, subsequent experiments and results are based on a robot swarm of 10 robots.

The experiment in Section 5.2 was carried out to investigate the effects of the length of a control cycle on the absolute values on the data and eventually the

ability to infer the presence of a fault. In this section, the consistency of data of four different control cycles. The consistency of data is important to support the assumption that data from homogenous robots in a local neighbourhood are similar and thus it is possible to use this information to infer the presence of a fault. Results from this experiment show that the longer the control cycle, the more consistent the data as indicated by a lower percentage relative standard deviation. However, having a longer control cycle also means a longer time would have elapsed before a fault can be detected. Out of the four control cycles investigated, a control cycle of 250s provides the greatest consistency of data between robots and thus subsequent experiments are based on a control cycle of 250s.

With the size of the robot swarm and the length of a control cycle for the simulation determined in Section 5.1 and Section 5.2, Section 5.3 analysed the occurrence of the faults specified in Section 4.4 by examining the foraging performance as the faults were injected to a robot in the swarm. Results shows that in a `CST` scenario, the performance of the robots before and after the faults have been injected is distinctively different and thus provide evidence to support **E1**. From this experiment, it appears that the presence of the faults may be easily inferred.

However, it is expected to be different when the robot swarm was deployed in dynamic environments. Therefore, Section 5.4 investigated the performance of the robots in time-varying environments as specified in Section 4.5. Results from this experiment show that under different operating conditions, the foraging performance is also different and thus provide evidence to support **E2**. Results also show that in the $\mathtt{V}_{\mathtt{OPR}}$ scenario, the performance of a faulty robot is very similar to fault-free robots and thus can be very difficult to infer the presence of the faults. The $\mathtt{V}_{\mathtt{OPR}}$ scenario is a representation of many analogous operational environments for which a robot swarm might operates in real-world. Therefore, to be able to infer the presence of a fault in this condition is important.

To investigate ways to infer the presence of faults in time-varying environments from the perspective of a collective, the next part of the thesis will examine this aspect. In the next chapter, a collective detection scheme will be proposed and its application will be demonstrated with the use of statistical classifiers.

Part III

Error Detection in Swarm Robotics: Approach and Algorithms

CHAPTER

SIX

Collective Self-Detection Approach and Statistical Classifiers

This chapter proposes an error detection scheme from the perspective of a collective to deal with the issue of adaptivity to dynamic environments. The scheme is tested with the use of statistical classifiers. In Section 6.1, illustrative examples are given on the inefficiency in detecting errors in the simulated time-varying environments from the perspective of a single robot. Therefore, error detection from a perspective of a collective referred to as the CoDe scheme is proposed in Section 6.2. To evaluate the performance of error detection, Section 6.3 describes the input data and the performance metrics. Section 6.4 proceeds with the implementation of two conventional parametric classifiers in the context of the CoDe scheme. This is followed by Section 6.5 with non-parametric classifiers. Also in this section, the performance between the implemented classifiers are compared. Finally in Section 6.6, an analysis is presented on the performance of the implemented statistical classifiers as the detection threshold is varied.

6.1 Error Detection - Perspective of a Single Robot

From a perspective of data-driven error detection, if the input data comes only from a single robot, the simplest model-based classifier for current problem may be a linear classifier. For example, assuming it is known that a robot swarm might be deployed in either a $\mathtt{V}_{\mathtt{OPR}}$ or $\mathtt{V}_{\mathtt{ODS}}$ scenario, and each robot in the swarm may experience faults of the wheels as presented in Section 4.4. Lets further assume that

the graphs in Figure 6.1 represent the obtained data on the foraging performance of a fault-free robot in $\mathtt{V_{OPR}}$ and $\mathtt{V_{ODS}}$ scenarios, and with faults in a CST scenario. If the exact time intervals of when the faults and the changes in the environment to occur are known, then a perfect classifier may be derived. However, outside controlled environments having such information is often not possible.

In Figure 6.1, two linear classifiers (classifier1 with a threshold at 2.6 and classifier2with a threshold at 1.7) are shown. In this example, classifier1 can detect all $\mathtt{P_{CP}}$, $\mathtt{P_{PT}}$ and $\mathtt{P_{GR}}$ errors in CST. However, in $\mathtt{V_{OPR}}$ and $\mathtt{V_{ODS}}$ scenarios there are instances, as indicated by thick purple lines in Figure 6.1, in which the errors are wrongly identified (referred to as false positives). For the $\mathtt{V_{OPR}}$ scenario, the false positive rate for classifier1 is 0.38 ($\frac{30}{80}$, from control cycles [28, 40] and [64, 80]). For the $\mathtt{V_{ODS}}$ scenario, the false positive rate is 0.03 ($\frac{2}{80}$, at control cycles 75 and 80).

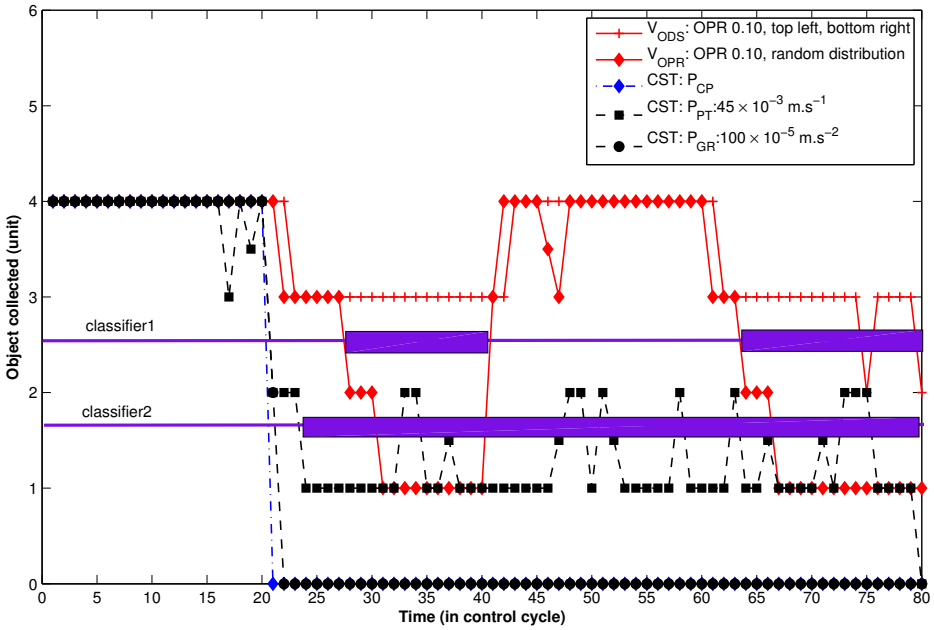


Figure 6.1: An example of the data that is available for the design of model-based classifiers.

For classifier2, it can detect all $\mathtt{P_{CP}}$ and $\mathtt{P_{GR}}$ errors in the CST scenario. However, it failed to detect many instances of $\mathtt{P_{PT}}$ errors. Failure to detect an error is a false negative. In this case, the false negative rate for $\mathtt{P_{PT}}$ errors in CST scenario is 0.22 ($\frac{13}{60}$). Also this classifier produces no false positive in a $\mathtt{V_{ODS}}$ scenario when the robot is fault-free. However, in the $\mathtt{V_{OPR}}$ scenario the false positive rate is 0.30 ($\frac{24}{80}$).

From the two examples, it is apparent that from a perspective of a single robot

using a linear classifier to detect faults of the wheels is insufficient when dealing with dynamic environments.

Besides linear classifiers, another possible approach to detect errors in the robot is to use a sliding time window on the data to calculate some descriptive values (depending on the technique used) of the sample. Figure 6.2 illustrates the usage of a sliding time window of size 4 being applied on the data $\mathbf{x}_{(i-3)}$ to $\mathbf{x}_i$ to produce a descriptive value $\mathbf{y}^i$. For example, control cycle $t = 6$, the sliding window uses data from control cycle $t = 3$ ($\mathbf{x}_3$) to control cycle $t = 6$ ($\mathbf{x}_6$) to calculate $\mathbf{y}^6$. Then, $\mathbf{y}^6$ can be compared to some predefined threshold to see whether an error has occurred.

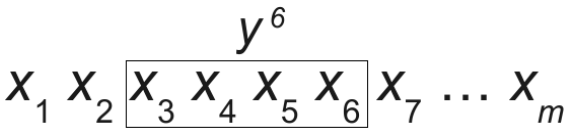


Figure 6.2: An example of a classifier based on a sliding time window.

To illustrate the usage of a sliding time window, Figure 6.3 is a sample of the data for a fault-free robot in the $\mathrm{V}_{\mathrm{OPR}}$ scenario from control cycle 18 to 38 originally from Figure 6.1. Lets assume a Q-test [135] classifier is used with the time window. A Q-test classifier calculates a Q-value of the sample by finding the difference between a current value x and the closest value to it (x^*) and then divides the difference over the range (largest value - lowest value) in the sample ($\frac{|x-x^*|}{max-min}$). This Q-value is then compared to a threshold value from the Q-table (Table A.1). Assuming the calculated Q-value is to be evaluated at 95% confidence level. From the Q-table, for a sliding window of size 4 the corresponding Q-value is 0.846. If the calculated Q-value is greater than the value from the Q-table, current observation x is classified as an error.

```
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 ..t
 4  4  4  4  3  3  3  3  3  3  2  2  2  1  1  1  1  1  1  1  1 ..obj
```

Figure 6.3: A sample data on a fault-free robot in the $\mathrm{V}_{\mathrm{OPR}}$ scenario from Figure 6.1

Applying the Q-test classifier over the sample data, the calculated Q-value is presented in Figure 6.4. Comparing the calculated Q-value to 0.829, there are three instances of error detected, at control cycle 22, 28, and 31. Notice that these instances are at the beginning of a transition in the data. After each transition, subsequent values were not classified as errors. Comparing this method with the linear classifiers presented earlier for the sample, this method produces less

false positives. There were only 3 false positives for a Q-test classifier with a time window of 4 whilst it was 11 for `classifier1`, and 8 for `classifier2`.

```
 21   22   23   24   25   26   27   28   29       t
 0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0    Q-value

 30   31    32   33   34   35   36   37   38      t
 0.0  1.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0   Q-value
```

Figure 6.4: Calculated Q-value.

From the perspective of a single robot, using a sliding time window technique is capable of producing a lower number of false positive in a dynamic environment (i.e. in a V_{OPR} scenario). However, it is still susceptible to false positives, which will become more apparent if the frequency of changes in the environment increases. For this reason, the following section will investigate the problem of adapting to dynamic environment from the perspective of a collective, a collection of robots logically grouped according to a wireless communication range.

6.2 CoDe: A Collective Self-Detection Approach

The central idea of a collective self-detection is that each robot periodically communicate its data to other robots that fall within a logically defined space. In this thesis, the space is defined by the wireless communication range of a robot. Assuming a perfect communication medium, all robots within a communication range can broadcast and receive data from each other. With acquired data from other robots, a robot determines whether itself is behaving normally with relative to other robots. Here, this approach is referred to as a **Co**llective Self-**De**tection (CoDe) Scheme. This approach is applicable for current problem because of the homogeneity of robots in a swarm (condition **E3**) first described in Section 4.1.

There are two parts to CoDe: detection from the perspective of a collective **P1** (a collection of robots within a logical communication range), and the detection of errors within an individual (self-detection, **P2**) instead of other robots in the collective. For **P1**, it is assumed that if all robots are physically and logically homogeneous, then they will behave and perform equally well in the same environment in which they are deployed. Therefore, homogeneity of robots is important and it is dictated by **E3** in this work. Under this assumption of homogeneity, it is possible to infer the presence of a fault by cross-referencing a robot's data with others within certain communication range (i.e. 2m radius in this work).

This concept of a collective decision through cross-referencing of data between agents is not new. In social science, it is referred to as social comparison [89] which is an important basis of opinion formation in social groups. In social groups, it is often impossible to immediately determined whether an opinion is correct or not, even with reference to the physical world. The same applies for an accurate assessment of one's ability by reference to the physical world. For such situations, subjective judgements of correct or incorrect opinion and assessments of one's ability depends upon how one compares with others [89]. This forms the second hypothesis for social comparison in [89] that states:

> *Hypothesis II: To the extent that objective, non-social means are not available, people evaluate their opinions and abilities by comparison respectively with the opinions and abilities of others.*

The unavailability of objective means in Festinger [89] is analogous to the problems of dynamic environments in this work. Since the foraging performance of the robots in the robot swarm is also influenced by the environment in which they operate, there is no definite value of how many objects a robot should collect. However, since the robots are homogeneous, it possible to (subjectively) determine whether a robot is faulty by comparing a robot's data with others within a collective.

The second part of the CoDe scheme, **P2**, relates to the identification of a faulty robot from the collective data. On each robot, the data from the collective can be used to determine whether the robot itself is faulty (self-detection) or to identify other faulty robots (exogenous-detection). However, this work advocates the self-detection approach for a number of reasons:

- a robot does not need to acquire and store information regarding the identity of other robots in the collective. Therefore, less data is needed to broadcast and receive information, and the energy usage and storage is minimised;
- when an error is detected, the robot can react immediately. For exogenous-detection, the observer robot has to convey the finding to the target robot which requires an additional communication. Besides, the target robot might not be reachable as it might have moved outside the communication range of the observer robot; and
- no central point of detection. Every robot is responsible to detect the errors within itself. Therefore, in principle, it is scalable to a robot swarm of any size.

Self-detection is a phenomenon that has been observed in nature. In the context of social insects, it exists in the form of self-isolation. For some social insects, when an insect is infected by parasites, (most of the time) the infected insect will leave the nest to die in isolation (self-isolation) [88]. Instead of being actively located and isolated by healthy members, the act of withdrawal to die in isolation appears to be self-motivated. This self-motivated isolation is interesting and has been argued to be part of the defence mechanism in social insects to prevent the spread of infections [88]. Therefore, together with the reasons specified previously, the self-detection approach could be potentially very important for current problem.

To demonstrate the potential of the CoDe scheme, Figure 6.5 is a sample of collective data as received by an observer robot `R1` (assuming all robots can communicate with each other). Notice that the sample data from `R1` is the same as those in Figure 6.3. Applying the same Q-test classifier as in the previous example, but from the perspective of a collective, the results for a confidence level of 95% (corresponding Q-value is 0.493) are shown in Figure 6.6. From the perspective of R1, the Q-test result is 0 ($\frac{4-4}{5-3}$) for control cycle 18, 0 ($\frac{4-4}{5-3}$) for control cycle 19, and so on. At control cycle 22 ($\frac{3-3}{5-2}$=0), 28($\frac{2-2}{4-2}$=0), and 31($\frac{|1-2|}{4-1}$=0.333), there are no false positives detected. Note that for robots R2 and R3 false positives are detected at control cycle 18.

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	..
R1	4	4	4	4	3	3	3	3	3	3	2	2	2	1	1	1	1	1	1	1	1	..
R2	5	3	4	4	4	4	1	3	4	4	3	3	4	4	4	2	1	4	3	3	1	..
R3	3	5	5	5	3	4	4	3	3	3	4	3	4	3	3	4	3	3	1	3	2	..
R4	4	3	4	4	3	4	3	2	4	1	2	1	2	3	4	2	1	3	2	3	3	..
R5	4	4	4	3	4	3	3	3	4	4	3	3	4	2	4	4	3	3	3	2	3	..
R6	4	3	4	3	4	4	4	4	3	3	4	2	3	3	3	3	3	2	4	2	4	..
R7	4	4	4	4	3	4	3	3	3	3	2	4	4	3	2	3	2	3	2	2	2	..
R8	4	5	4	4	2	5	4	3	3	3	3	3	4	3	3	3	3	3	4	4	3	..
R9	4	3	4	4	4	3	4	3	4	2	4	3	2	4	3	3	2	4	2	3	3	..
R10	4	5	3	4	5	4	2	5	3	3	3	3	0	3	3	4	2	3	3	2	3	..

Figure 6.5: Sample data from a collective of 10 robots from control cycle 18 to 38.

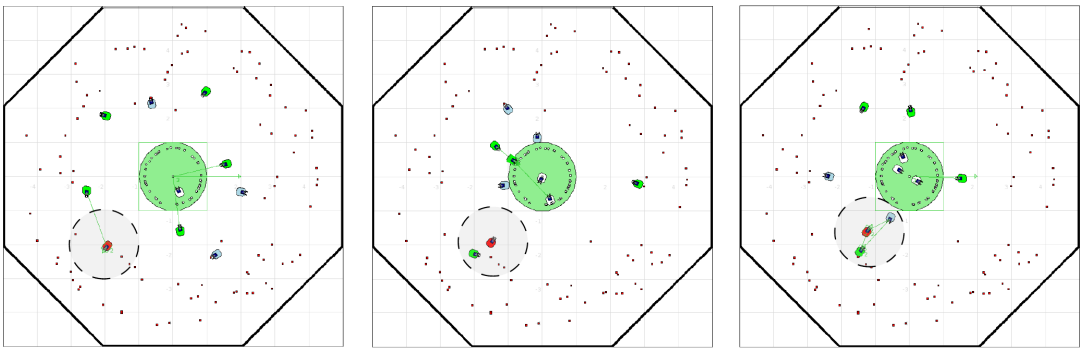
Notice that the example presented in Figure 6.5 assumes all robots can communicate with each other, which is often not the case. For example, with a wireless communication range of 2m, each robot in the robot swarm in this thesis only interact with a few other robots as illustrated in Figure 6.7. Therefore, at control cycle t in the CoDe scheme, the collection of neighbours of an observer robot is dynamic; it ranges from zero to the maximum number of robots minus one in the arena (9 in this thesis) or the number that can be spatially fit in the commu-

18	19	20	21	22	23	24	25	26	27	28	t
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Q-value

29	30	31	32	33	34	35	36	37	38	t
0.0	0.0	0.333	0.333	0.0	0.333	0.0	0.333	0.0	0.0	Q-value

Figure 6.6: Calculated Q-value from the perspective of a collective with the CoDe scheme for sample in Figure 6.5.

nication radius. This fits nicely for the Q-test classifier that requires independent samples from the original population.



(a) $t = 3$ (b) $t = 7$ (c) $t = 9$

Figure 6.7: Time-lapsed screenshots showing the different number of robots within some communication range of a robot in red.

With the CoDe scheme, there is a special case when an observer robot interacts with less than two other robots. For such a case, cross-referencing a robot's data with others is unreliable because (1) if an observer robot interacts with zero robot, there exists no data to cross-reference, (2) if an observer robot interacts with only one other robot, then it might be the case that either robot can be faulty and thus it can lead to either a false positive, or a false negative. In this thesis, this special case is handled by reverting to use the collective data from a previous control cycle ignoring the data from the other robot at current control cycle.

Having given the necessary details regarding the CoDe scheme, it is formally presented in Algorithm 1. At control cycle t, an observer robot in which the algorithm runs will have a set of data of the current collective $\mathcal{DN}$ together including data instance from the observer robot v from the observer robot. As mentioned in previous paragraph, if an observer robot interacts with less than two other robots, data from a previous control cycle will be used. To determine whether v is errorneous with respect to $\mathcal{DN}$, the data is presented to classifier $\mathcal{A}$. Taking this approach, the detection is a task of determining whether an error has occurred on the observer robot and thus a self-detection.

Algorithm 1: Collective Self-Detection Scheme (CoDe)

Input: current data instance v, neighbourhood data $\mathcal{DN}$, classifier $\mathcal{A}$
Output: Result of detection
initialisation;
foreach *control cycle* t **do**
  if *CalculateNeighbour*($\mathcal{DN}$) < 2 **then**
    err = $\mathcal{A}(v, \mathcal{DN}_{temp})$;
  else
    err = $\mathcal{A}(v, \mathcal{DN})$;
    $\mathcal{DN}_{temp}$=$\mathcal{DN}$;
  end
  if err **then**
    Report err;
  end
end

Note that the classifier $\mathcal{A}$ has to be able to cope with data of various sizes given by the number of robots an observer robot interacts with. Therefore, statistical classifiers which can be easily adapted to work with data of dynamic sizes are chosen. Section 6.4 and Section 6.5 will investigate the usage of statistical classifiers to detect errors in dynamic environments in the context of the CoDe scheme.

6.3 Data and Performance Metrics

6.3.1 Data

From the analysis in Section 5.2 as well as the illustrative examples in Section 6.1 and Section 6.2, it is apparent that to infer the presence of a fault based on only the number of objects collected ($\texttt{obj} \in \mathbb{Z}_0^+$) is insufficient. The results in Section 5.2 show that a difference of one object between the robots amounts to a pRSD of 20%. Also, the results in Sections 6.1 and 6.2 demonstrate that a difference in one object is significant between a positive detection and a false detection. For this reason, for subsequent experiments, the other data that are also influenced by both the faulty wheels and the environments are included: energy used ($\texttt{eng} \in \mathbb{R}_0^+$), and distance travelled ($\texttt{dist} \in \mathbb{R}_0^+$). Each data variable will be independently presented to the classifier and an error is considered detected if it is flagged in more than one variable.

6.3.2 Performance Metrics

The performance of error detection by a classifier is derived based on the following metrics, which are commonly used in the literature:

- True Positive (TP) - an error is correctly classified;
- False Positive (FP) - a normal instance is incorrectly classified as an error;
- True Negative (TN) - a normal instance is correctly classified as normal; and
- False Negative (FN) - an error is incorrectly classified as a normal instance.

Given the above metrics, the accuracy of the error detection or the true positive rate (TPR) is defined as the proportion of the number of correctly classified errors over the total number of errorneous instances (Eq. 6.1).

$$\mathtt{TPR} = \frac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FN}} \tag{6.1}$$

Similarly, the false alarm or the false positive rate (FPR) is defined as the proportion of the number of incorrectly classified errors over the total number of normal instances (Eq. 6.2).

$$\mathtt{FPR} = \frac{\mathtt{FP}}{\mathtt{FP} + \mathtt{TN}} \tag{6.2}$$

On top of the TPR and FPR, another performance metric called the responsiveness of detection (Latency) is also evaluated. This metric evaluate how long the time has elapsed before an error is positively identified. Given that $\mathtt{t_{pd}}$ is the fault detection time, and $\mathtt{t_{ft}}$ is the fault injection time, then

$$\mathtt{Latency} = \mathtt{t_{pd}} - \mathtt{t_{ft}} \tag{6.3}$$

As mentioned in Section 4.3.4, this thesis only considers the case of a single robot failure in the robot swarm. Thus, the TPR and the Latency is only applicable to the faulty robot. However, the FPR is calculated over all the robots in the swarm. It will be noted that if the FPR for the faulty robot is separately calculated, the FPR will likely be higher compared to those of all robots because the relative contribution per unit of false positive toward the FPR is higher. In other words, one instance of false positive per robot is equal to a FPR of 0.05 ($=\frac{1}{20}$) for the faulty robot alone whilst it is ≈ 0.01 ($\frac{10}{20+(9\times80)}$) for the whole system.

6.4 Conventional Parametric Classifiers

In statistics, the task of determining whether a data instance is errorneous is an example of the outlier detection problem. An outlier is an observation that devi-

ates so much from other observations as to arouse suspicion that it was generated by a different mechanism (Hawkins, 1980). There are three aspects to this definition: the observation, the deviation, and the mechanism. The observation is the statistical properties of the variables of interest (the data). How much deviation is considered as an outlier is governed by the statistical thresholds. The mechanism relates to the underlying assumptions on the distribution of the variables.

Depending on whether an assumption is made on the underlying data distribution, statistical outlier detection techniques can be parametric or non-parametric. With parametric techniques, it is assumed that the observed variables are generated by some named probability distribution such as a Gaussian or Normal distribution. The opposite of parametric statistics is non-parametric statistics in which the data are not assumed to belong to any particular distribution.

If the distribution of data is known, parametric techniques can be more accurate and are generally advised. However, for many real world applications, it is often impossible to infer the underlying distribution of data and thus non-parametric techniques are more applicable. However, non-parametric techniques are more powerful when a sufficiently large data set is available [136].

To illustrate the frequency distribution for the data in this thesis, histograms on the `obj`, `eng`, and `dist` in a `CST` scenario are presented in Figure 6.8. From a manual inspection on the histograms, it appears that the frequency distributions follow a Normal distribution albeit discretised. Therefore, the first experiment investigates the implementation of parametric classifiers in the context of the CoDe scheme for the current error detection problem.

6.4.1 Experimental Setup

Experimental Objective: To evaluate the performance of two conventional parametric classifiers in the context of the CoDe scheme for error detection in both non-dynamic and dynamic environments.

It will be recalled that with $\mathtt{P}_{\mathtt{CP}}$ (complete failure), the left wheel on a faulty robot was set to left turn by 10°. This causes the robot to move in a circle and unable to continue foraging. With $\mathtt{P}_{\mathtt{PT}}$ (partial failure), the robot wheels moves less efficiently by reducing the speed of the wheels to 45 $\times 10^{-3}$ m.s^{-1} causing the robot to move more slowly but otherwise still able to search and collect objects. With $\mathtt{P}_{\mathtt{GR}}$ (gradual failure), the wheels on a faulty robot move with a gradually reducing speed of 100 $\times 10^{-5}$ m.s^{-2}.

These faults were independently injected to a robot in the swarm that operate

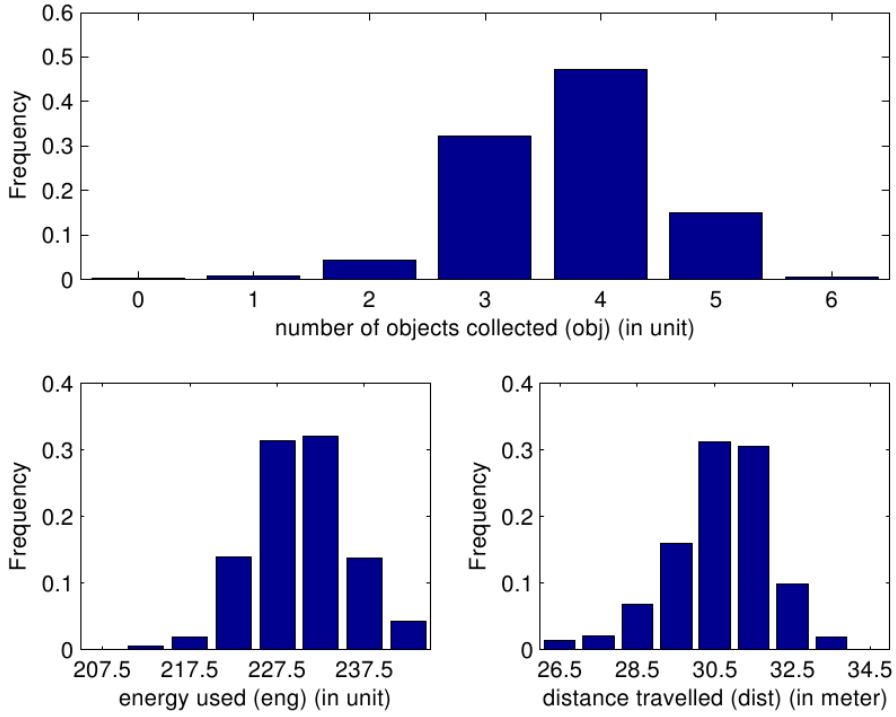


Figure 6.8: The distribution of `obj`, `eng`, and `dist` in the system for a control cycle of 250s in `CST`.

in `CST` (constant `OPR`), $\texttt{V}_{\texttt{OPR}}$ (varying `OPR`), and $\texttt{V}_{\texttt{ODS}}$ (varying object distribution) scenarios. For each combination of a fault model and a scenario, twenty repeated runs were performed. The faults were injected at control cycle 20 and persisted until the end of the simulation. The changes to the environmental condition in $\texttt{V}_{\texttt{OPR}}$ and $\texttt{V}_{\texttt{ODS}}$ scenarios occur at control cycles [20, 40), and control cycles [60, 80). Assuming the classifiers can detect all errors in non-dynamic environment. In principle, if the classifiers were not able to adapt to the environmental changes, then the `FPR` will be greater than $\frac{40}{80}$ = 0.50 (i.e. environmental changes last for 40 control cycles of a simulation of 80 control cycles). Similarly, the `TPR` will be lower than 0.50. Note that a classifier with a `TPR` and a `FPR` that equals to 0.50 is equivalent to a random classifier.

In this experiment, two parametric classifiers were tested: Extreme Studentized Deviate (ESD) and T-test classifiers. The ESD classifier or the Grubbs' method [137], calculates a Z value as the difference between the arithmetic mean μ of a sample and a data instance x over the standard deviation σ. The Z value for x is compared with a threshold k. If the calculated Z value is greater than k, x is classified as an outlier (Eq. 6.4).

$$Z = \frac{|x - \mu|}{\sigma} > k \tag{6.4}$$

The value of k in Eq. 6.4 is usually taken as 2 or 3 because if the data are normally distributed, they are expected to be within two (three, respectively) standard deviations from the mean [138].

To implement the ESD classifier in the context of the CoDe scheme, the σ and μ are calculated using the data from neighbours. Then, the data instance v from current robot is substituted into Eq. 6.4 to calculate Z. For this experiment, the default or commonly used threshold value of 2 for k is used.

A T-test is a statistical hypothesis test to assess whether the arithmetic means μ of two groups (A and B) are statistically different from each other. The T-test classifier is computationally more complex than the ESD classifier. It calculates the ratio (Eq. 6.5) of the difference between the two means over a measure of the variability or dispersion of the scores (also referred to as the standard error of the difference) [139]. After the t-value is calculated, it is compared with the standard t_s-value of p confidence interval in the T-table (Table A.2). If the t-value is greater than the t_s, group A is statistically different from group B.

$$t\text{-value} = \frac{|\mu_A - \mu_B|}{SE(\mu_A - \mu_B)} \leq T(p, df) \tag{6.5}$$

The parameter p is the statistical significant threshold (usually 0.05 for a 95% confidence level), df is the degrees of freedom, and $T(p, df)$ is the standard t_s-value that corresponds to given values of p and df in the T-table.

To implement the T-test classifier, instead of communicating a current data instance at each control cycle, each robot communicates its mean value of respective data variable calculated over five control cycles. The p value for the T-test classifier is set to the default value of 0.05.

The effectiveness of detecting injected faults is evaluated based on the performance metrics presented in Section 6.3.2. For each classifier, the `TPR`, `FPR`, and the `Latency` were calculated for each run. The results were then compiled and presented in boxplots. For each boxplot, the centre line of the box is the median whilst the upper edge of the box is the third quartile and the lower edge of the box is the first quartile. The whiskers extend to the extreme data points not considered outliers, and outliers are plotted individually.

6.4.2 Experimental Results

The results on the performance in the detection of $\texttt{P}_{\texttt{CP}}$ errors with the ESD and the T-test classifiers are presented in Figure 6.9. In the figure, the boxplots are arranged so that the first two boxplots are the results for the `CST` scenarios, followed by the next two boxplots for the $\texttt{V}_{\texttt{OPR}}$, and ends with the last two boxplots

for V_{ODS} scenarios. The label A1 stands for the ESD classifier, A2 for the T-test classifier, S1 for CST scenario, S2 for V_{OPR} scenario, and S3 is for V_{ODS} scenario. With this arrangement, it is easier to compare the results between the two classifiers.

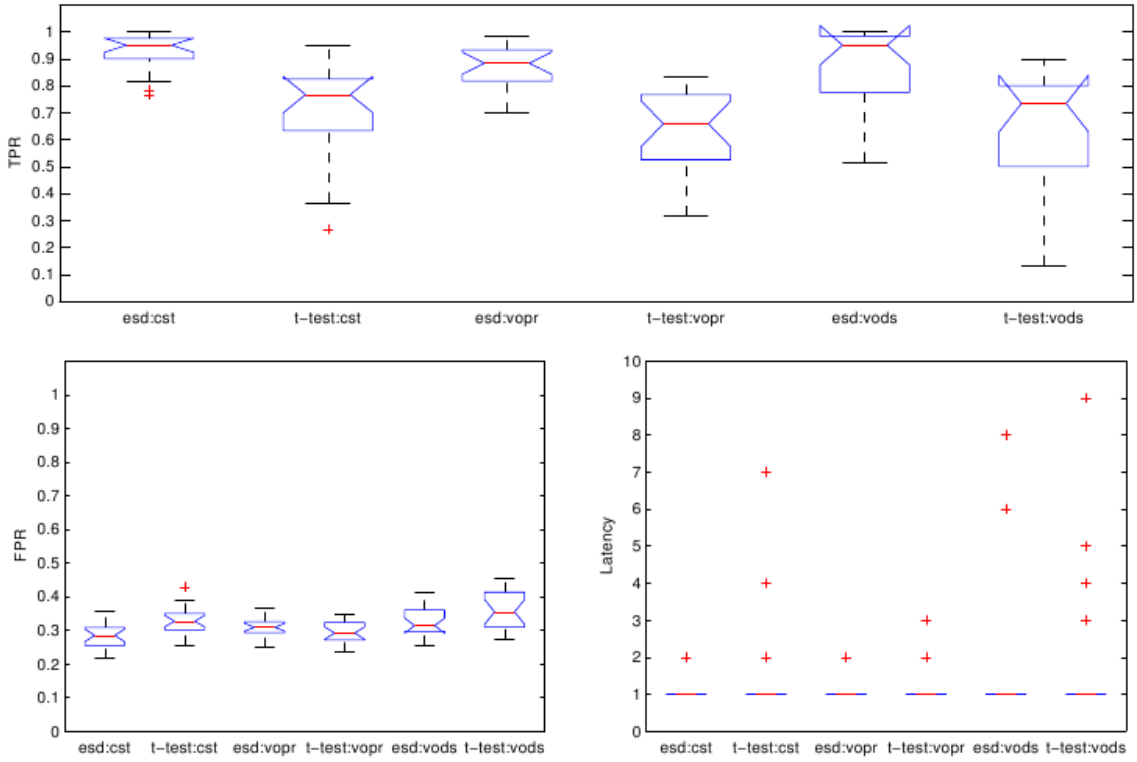


Figure 6.9: Boxlots for the TPR, FPR, and the Latency for the ESD and T-test classifiers in detecting P_{CP} errors.

The top figure in Figure 6.9 for the results of the TPR shows that the median TPR for both classifiers is above 0.60 with the ESD classifier being the one with a higher TPR in all scenarios. This results indicate that the ESD classifier may be more effective in inferring the presence of P_{CP} from the data compared to the T-test classifier. Having said that, the comparison has to consider the results for the FPR as well. A classifier may achieve a perfect detection (TPR = 1.00) by setting a low detection threshold, but this may inadvertently results in a high FPR. Nevertheless, the results on the FPR in the bottom left figure show that the FPR for the ESD classifier is lower compared to the T-test classifier in two of the three scenarios (i.e. in CST and V_{ODS} scenarios). Therefore, in this case, it seems that the ESD classifier with the current threshold value achieved a better result compared to the T-test classifier in detecting P_{CP} errors. For the Latency, both classifiers has a medium Latency of 1.0.

Examining the results over different scenarios, it is noticeable that the TPR is lower in a V_{OPR} scenario compared to the CST and V_{ODS} scenarios. This is evident

from the results for both classifiers. For the ESD classifier, the TPR is around 0.95 in the CST and V_{ODS} scenarios. However, it is slightly lower than 0.90 in a V_{OPR} scenario. Similarly, for the T-test classifier the TPR is around 0.65 in a V_{OPR} scenario whilst it is above 0.75 in other scenarios. These results suggest that the changes in the environment in a V_{OPR} scenario is more challenging compared to other scenarios in the detection of P_{CP} errors.

The challenge in detecting P_{CP} errors in a V_{OPR} scenario comes from the fact that as the quantity of objects in the arena gradually diminishes and approaches zero (as simulated by reducing the OPR from 0.10 to 0.025), many fault-free robots will not be able to collect any objects. At the same time, a faulty robot may not be able to collect any object and thus it is very hard to determine (based on operational data alone) whether a data instance is normal or anomalous. As a result, false detection (i.e. false positives and false negatives) is likely to happen. However, the results of detection may be improved if the detection is based on observations over multiple control cycles instead of every control cycle as currently implemented. This is further investigated in Section 7.4 and the results show that the FPR can be significantly reduced.

Compared to the effects of the environmental changes in a V_{OPR} scenario, the changes in a V_{ODS} scenario appear to have a lesser impact on the performance of all classifiers. Although the TPR is lower and the FPR is higher, the difference in the TPR is low. For example, for the ESD classifier the difference in the TPR between a CST scenario and a V_{OPR} scenario is about 0.07 (7%) whilst it is zero with a V_{ODS} scenario. This is likely because no matter where the objects are distributed, the robots still wander randomly in locating the objects. However, if the robots are wandering in areas with a small number of objects, less objects will be collected.

Based on the results on the TPR and the FPR for both classifiers, which are significantly better than 0.50, it suggests that the presence of P_{CP} can be inferred. The results also give evidence that the application of parametric classifiers in CoDe can be adaptive to the environmental changes in V_{OPR} and V_{ODS} scenarios.

Figure 6.10 is the result on the detection of P_{PT} errors with the ESD and the T-test classifiers. Similar to the results for P_{CP} errors, the TPR for both classifiers is greater than 0.50 in all scenarios. Again, the ESD classifier has a higher TPR and a lower FPR compared to the T-test classifier in all scenarios.

The similar results of a lower TPR and a higher FPR with P_{PT} errors in a V_{ODS} scenario further demonstrates the difficulty in detecting errors in systems deployed in dynamic environments similar to those in a V_{ODS} scenario. However, albeit with a lower performance, the P_{PT} errors can still be detected as indicated with a TPR greater than 0.50 and a FPR lower than 0.50.

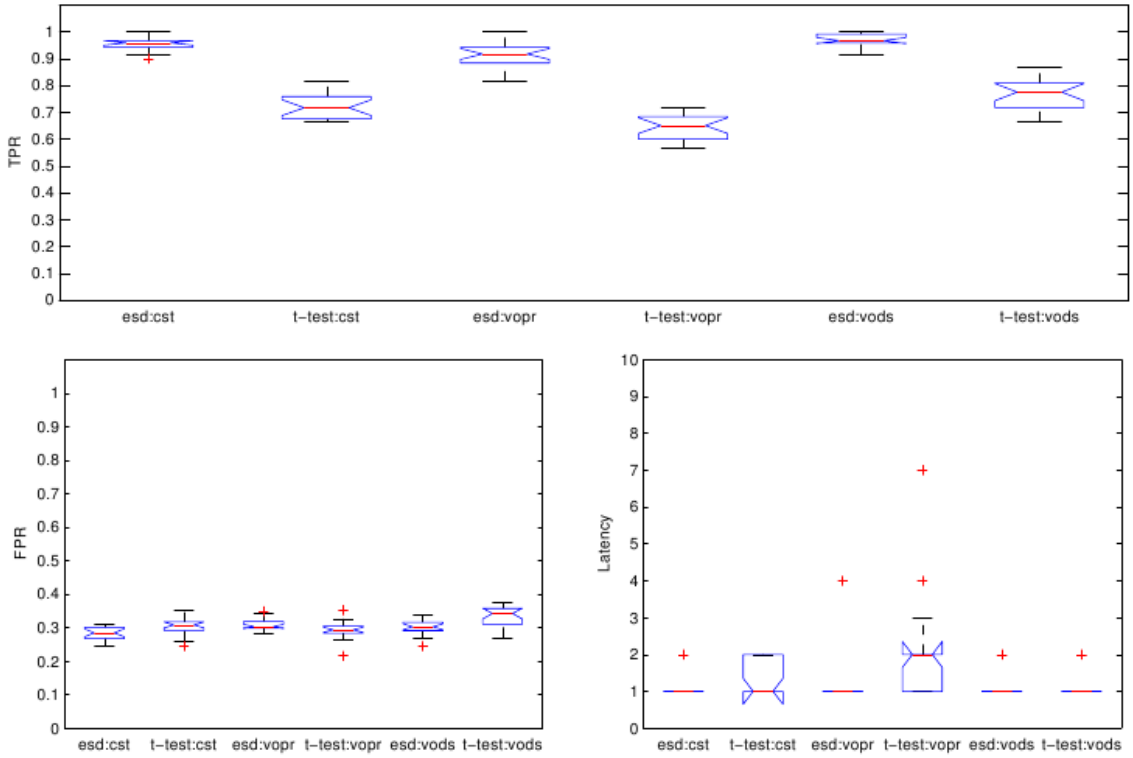


Figure 6.10: Boxlots for the TPR, FPR, and the Latency for ESD and T-test classifiers in detecting P_{PT} errors.

Finally, the results on the detection of P_{GR} errors with the two parametric classifiers are shown in Figure 6.11. The results on the TPR, FPR, and the Latency for both classifiers show similar trends to those with P_{CP} and P_{PT} errors. The TPR for both classifier is greater than 0.70, the FPR is lower than 0.30, and the Latency is 1.0 in all scenarios. As expected, a drop in the TPR and an increase in the FPR in a V_{OPR} scenario are observed.

The results on the detection of P_{CP}, P_{PT}, and P_{GR} errors with the parametric classifiers in CoDe provide evidence that an adaptive error detection can be achieved and the performance is significantly better than a random classifier. However, the findings from this experiment also raise many other questions. Questions such as what would be the results if non-parametric classifiers were used instead? Although the FPR is lower than 0.35 in all cases, is it possible to achieve an even lower FPR? How about TPR? The detection threshold used is the default recommended value, what if different values are used?

To address these questions, Section 6.5 investigates the performance of two non-parametric statistical classifiers with the same faults. This is followed by an investigation on varying the detection threshold in Section 6.6.

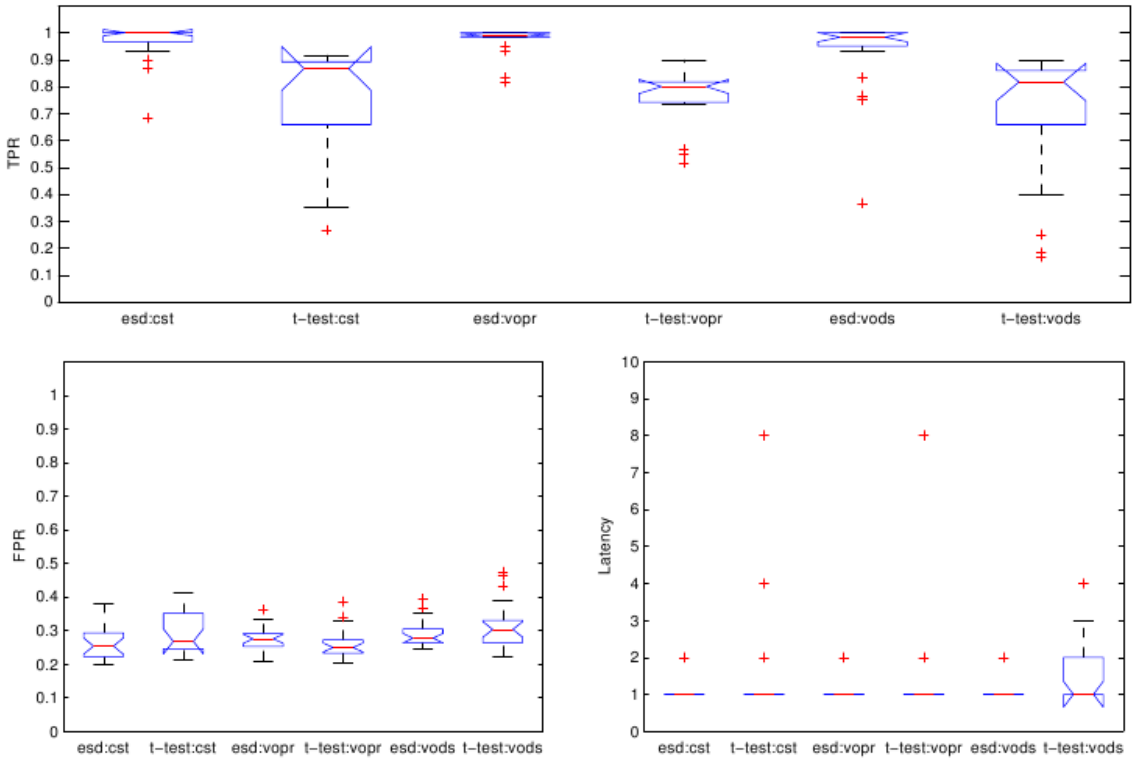


Figure 6.11: Boxlots for the TPR, FPR, and the Latency for the ESD and T-test classifiers in detecting P_{GR}.

6.5 Conventional Non-Parametric Classifiers

As discussed in Section 6.4, non-parametric techniques do not assume that the sample data belongs to any particular distribution. In this section, two non-parametric classifiers were implemented in the context of the CoDe scheme for the same faults in Section 6.4. They are the Quartile-based classifier and Q-test classifiers.

6.5.1 Experimental Setup

Experimental Objective: To evaluate the performance of two conventional non-parametric classifiers for error detection in both non-dynamic and dynamic environments and compare it with those of parametric classifiers.

Detection of outliers using the quartiles is a well known technique in statistics. With this classifier, a data instance x is classified as an outlier if it falls outside the 1.5 interquartile range (IQR) of either the first quartile which is the 25^{th} percentile ($x_{.25}$), or the third quartile which is the 75^{th} percentile ($x_{.75}$) of the data [140]. To

calculate the first and third quartiles, the data need to be arranged in an ascending order. The IQR is the range between the third quartile and the first quartile, i.e. IQR = $x_{.75}$ - $x_{.25}$. The IQR multiplier k of 1.5 is the *de-factor* value in statistics to define a mild outlier whilst a multiplier of 3 is for an extreme outlier [141].

$$x_{.25} - 1.5 \times IQR \geq x \geq x_{.75} + 1.5 \times IQR \tag{6.6}$$

For this experiment, the quartiles are calculated using the data from neighbours. The detection threshold k for the Quartile-based classifier is 1.5.

Dixon's Q test [135], or simply the Q test, is another non-parametric technique used for the identification and the rejection of outliers. To use a Q-test to determine whether a particular data instance x is an outlier, arrange the data in an ascending order and calculate the Q-value for x (Eq. 6.7). The gap is the absolute difference between x and the closest number to it, and range is the difference between the maximum and minimum value in the sample. The calculated Q-value (Q_{calc}) is then compared with the critical value of α confidence level in the Q-table (Table A.1). If Q_{calc} is greater than the critical value, x is considered as an outlier.

$$Q_{calc} = \frac{\texttt{gap}}{\texttt{range}} \tag{6.7}$$

For this experiment, results for the Q-test classifier is based on an α value of 90%. Using the data from Section 6.4.2, the Quartile-based classifier and the Q-test classifier were implemented in the context of the CoDe scheme to detect the P_{CP}, the P_{PT} and the P_{GR} errors.

6.5.2 Experimental Results

For ease of comparison, the results (the median) for the two non-parametric classifiers are presented together with those of parametric classifiers. Table 6.1 is the result for P_{CP} errors, Table 6.2 for P_{PT} errors, and Table 6.3 for P_{GR} errors. In the tables, the first multirow is the performance of the four classifiers in detecting errors in a CST scenario; the second multirow for the results in a V_{OPR} scenario; and the last multirow for the results in a V_{ODS} scenario. Also, the first two columns are the results for the parametric classifiers whilst the last two columns are the results for the non-parametric classifiers.

Examining the results on the TPR in Table 6.1, it shows that the parametric classifiers has a higher TPR compared to the non-parametric classifiers in almost all scenarios. However, it is the opposite for the results on the FPR. The FPR of the non-parametric classifiers is significantly lower than the parametric clas-

Table 6.1: The median TPR, FPR, and Latency for the ESD, T-test, Quartile-based, and Q-test classifiers in detecting P_{CP} errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	0.95	0.77	0.70	0.86
	FPR	0.28	0.33	0.12	0.12
	Latency	1.0	1.0	1.0	1.0
V_{OPR}	TPR	0.88	0.66	0.46	0.64
	FPR	0.31	0.29	0.10	0.13
	Latency	1.0	1.0	1.0	1.0
V_{ODS}	TPR	0.95	0.73	0.59	0.75
	FPR	0.31	0.35	0.13	0.13
	Latency	1.0	1.0	1.0	1.0

sifiers. This is also evident in the results for P_{PT} (Table 6.2), and P_{GR} (Table 6.3). This is an interesting result because it indicates that the non-parametric classifiers do not outperformed the parametric classifiers in every aspect. However, as the results are based on a default detection threshold and thus not conclusive. Therefore it may be worth to conduct a follow up experiment with different detection thresholds. Nevertheless, this result does give evidence that non-parametric classifiers can also provide an adaptive error detection with a performance that is significantly better than a random classifier.

Table 6.2: The median TPR, FPR, and Latency for the ESD, T-test, Quartile-based, and Q-test classifiers in detecting P_{PT} errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	0.96	0.72	0.78	0.92
	FPR	0.28	0.31	0.12	0.20
	Latency	1.0	1.0	1.0	1.0
V_{OPR}	TPR	0.92	0.65	0.70	0.89
	FPR	0.30	0.29	0.10	0.22
	Latency	1.0	2.0	1.0	1.0
V_{ODS}	TPR	0.97	0.78	0.74	0.89
	FPR	0.30	0.34	0.11	0.20
	Latency	1.0	1.0	1.0	1.0

Revisiting the observation on the difficulty in detecting errors in a V_{OPR} scenario with parametric classifiers, the results for the non-parametric classifiers further strengthen the conjecture on the effects of the limited objects on the detection. For example, the results on the TPR for the Q-test classifier in Table 6.1 drop from 0.85 in a CST scenario to 0.64 in a V_{OPR} scenario, a drop of 0.21. Similar, for the Quartile-based classifier, a drop of 0.24 is observed.

The same observation that the environmental changes in a V_{ODS} scenario has a lesser effects on the performance of detection compared to those in a V_{OPR} scenario is also evident from the results for the non-parametric classifiers. By comparing the results in V_{OPR} and V_{ODS} scenarios for the non-parametric classifiers in Table 6.1, Table 6.2, and Table 6.3, it is evident the TPR is higher in most cases (4 out of 6) in a V_{ODS} scenario.

Results on the performance for non-parametric classifiers (in the context of the CoDe scheme) in V_{OPR} and V_{ODS} scenarios show the TPR is significantly greater than 0.50 and the FPR that is significantly less than 0.50 further strengthen the evidence that an adaptive error detection can be achieved with CoDe.

Table 6.3: The median TPR, FPR, and Latency for the ESD, T-test, Quartile-based, and Q-test classifiers in detecting P_{GR} errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	1.00	0.87	0.81	1.00
	FPR	0.25	0.27	0.10	0.12
	Latency	1.0	1.0	1.0	1.0
V_{OPR}	TPR	0.99	0.80	0.77	0.97
	FPR	0.28	0.25	0.10	0.13
	Latency	1.0	1.0	1.0	1.0
V_{ODS}	TPR	0.98	0.82	0.76	0.98
	FPR	0.28	0.30	0.11	0.13
	Latency	1.0	1.0	1.0	1.0

6.6 Varying the Detection Threshold

The results for the experiments in Section 6.4.2 and Section 6.5.2 for the statistical classifiers are based on a fixed (default/recommended) detection threshold. A potential problem with using the default value is that the results may be not be optimal and better results may be obtained if a different value is used. The chosen detection threshold should be problem-based and varies from one application to another. Thus, this section investigate the effects on the performance for the four classifiers as the detection threshold is varied.

Experimental Objective: To evaluate the performance of the statistical classifiers as the detection threshold is varied in detecting errors in both non-dynamic and dynamic environments.

The simulation setting is the same as specified in Section 6.4 and Section 6.5.

For each variant of the classifiers, twenty repeated runs were conducted and the medians are used for the results.

6.6.1 The ESD Classifier

For the ESD classifiers, the value of k is varied from 0.0 to 3.0 with an increment of 0.5. The results on the performance with each fault as k is varied are shown in Figure 6.12, Figure 6.13, and Figure 6.14. In the figures, each point in a graph is the median and the lines connecting two points are drawn for clarity.

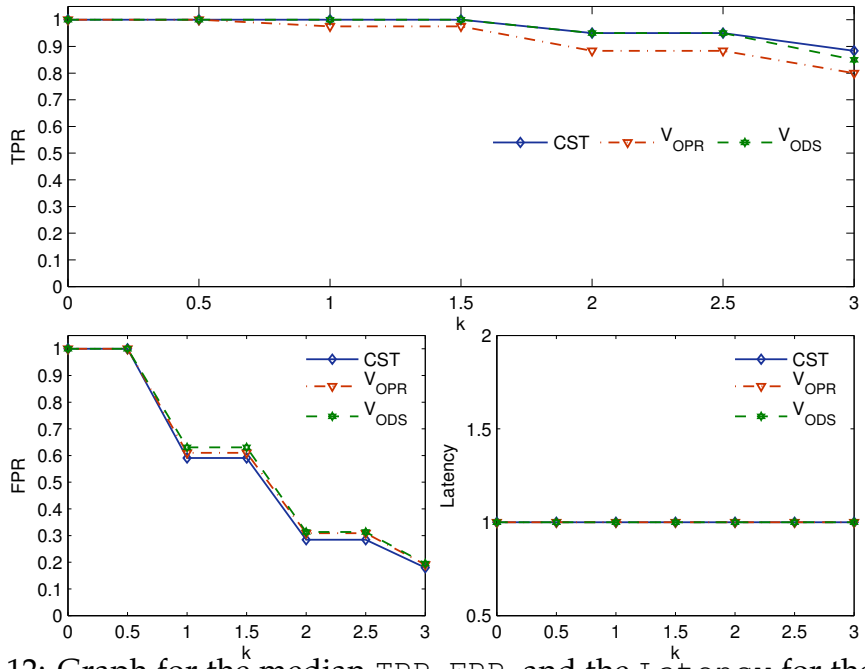


Figure 6.12: Graph for the median TPR, FPR, and the Latency for the ESD classifier in detecting P_{CP} errors as the detection threshold k is varied.

The results in Figure 6.12 with P_{CP} errors show that as the value of k increases, the TPR as well as the FPR decreases. This is observed in all scenarios. With different faults, the same observation is also seen as shown in the results with P_{PT} errors (Figure 6.13) and P_{GR} errors (Figure 6.14). From these results, it appears that a larger k is more appropriate for the ESD classifier with current data. Particularly, with a value of k that is greater than 1.5. This is because for a value of k equal to or less than 1.5, the FPR is greater than 0.50. However, as seen from the trend of decreasing TPR, if k is too large then more errors might be missed. Therefore, a tradeoff between the TPR and the FPR is needed depending on their relative importance.

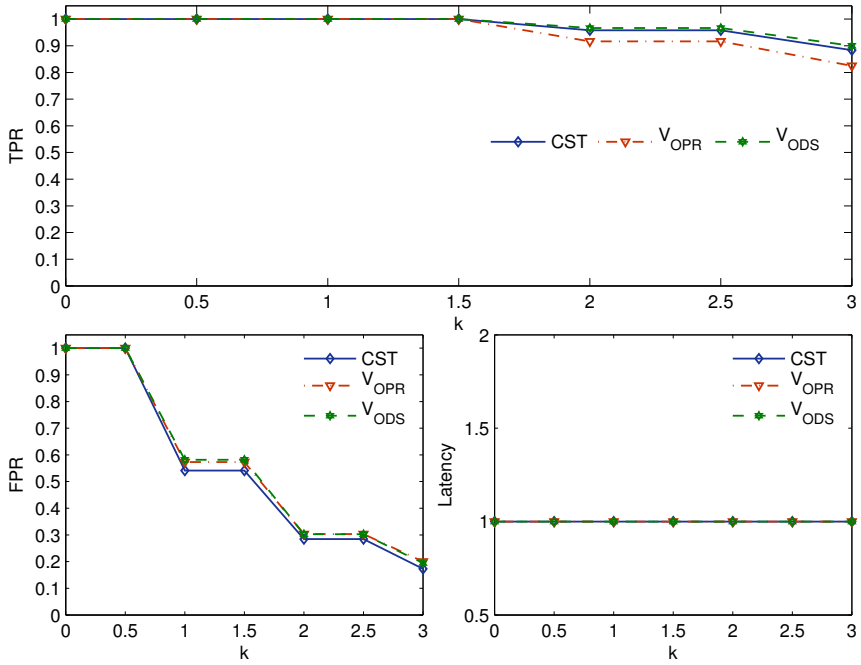


Figure 6.13: Graph for the median `TPR`, `FPR`, and the `Latency` for the ESD classifier in detecting $\mathtt{P}_{\mathtt{PT}}$ errors as the detection threshold k is varied.

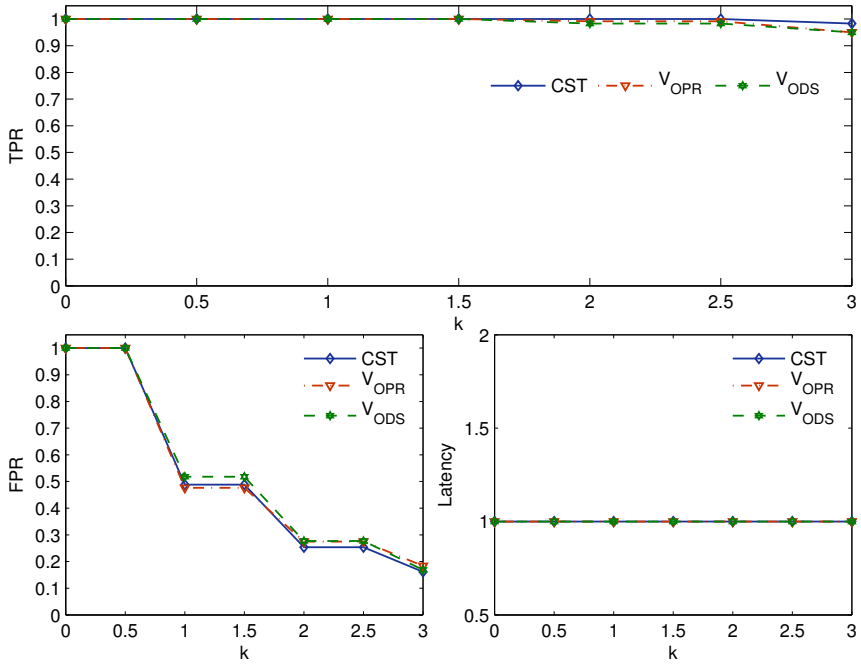


Figure 6.14: Graph for the median `TPR`, `FPR`, and the `Latency` for the ESD classifier in detecting $\mathtt{P}_{\mathtt{GR}}$ errors as the detection threshold k is varied.

6.6.2 The T-test Classifier

For the T-test classifier, all values of p in the T-test table (A.2) were tested. The p values are 0.0005, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, and 0.40 were tested. Figure

6.15 contains the results with $\texttt{P}_{\texttt{CP}}$ errors, Figure 6.16 for $\texttt{P}_{\texttt{PT}}$ errors, and Figure 6.17 for $\texttt{P}_{\texttt{GR}}$ errors as p is varied.

From these figures, it can be seen that as the value of p increases, the TPR as well as the FPR also increases. However, it is not a hundred percent detection (TPR=1.00). Also, the results indicate that there appears to be a peak value for the TPR. With $\texttt{P}_{\texttt{CP}}$ errors, a peak TPR is reached with p equals to 0.01 whilst the same state is reached with p equals to 0.005 with $\texttt{P}_{\texttt{PT}}$ errors. With $\texttt{P}_{\texttt{GR}}$ errors, a different pattern is observed. The TPR remained constant as p is varied. Notice that the same patterns are seen for the SRS in the three different environments (albeit with a different absolute value for the TPR).

As with the TPR, the FPR increases as p increases. However, the FPR does not appear to stop at a peak value but rather continue increasing as p gets larger. Therefore, depending on whether the objective is either to maximise the TPR or to minimise the FPR, the value of p equal to 0.01 or less can be used.

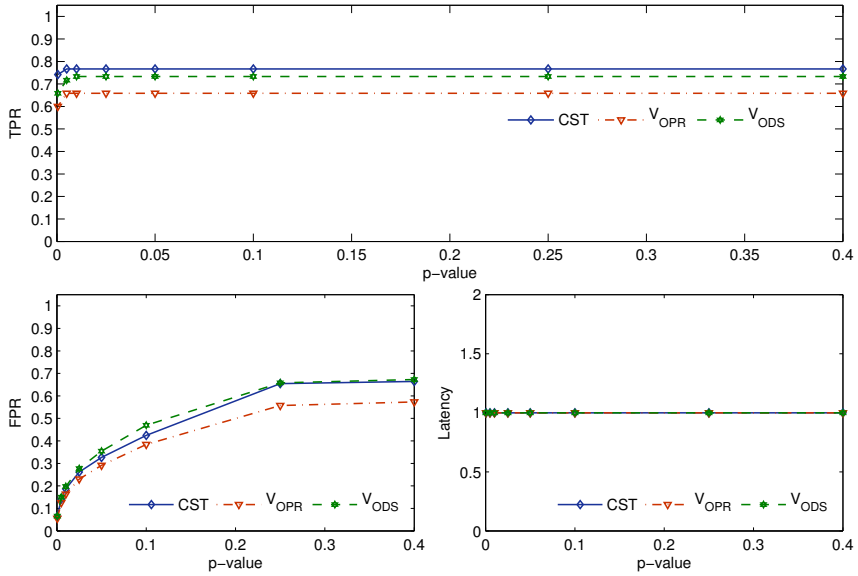


Figure 6.15: Graph for the median TPR, FPR, and the Latency for the T-test classifier in detecting $\texttt{P}_{\texttt{CP}}$ errors as the detection threshold p is varied.

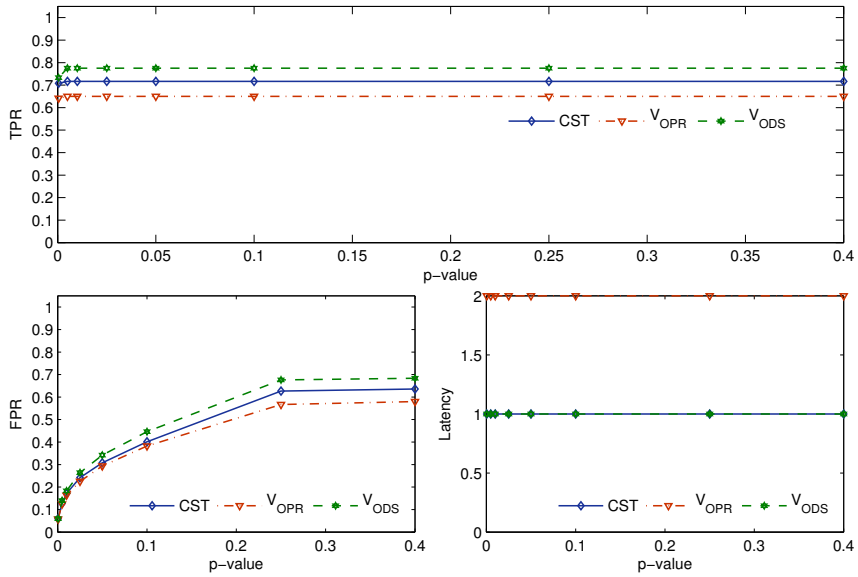


Figure 6.16: Graph for the median TPR, FPR, and the Latency for the T-test classifier in detecting P_{PT} errors as the detection threshold p is varied.

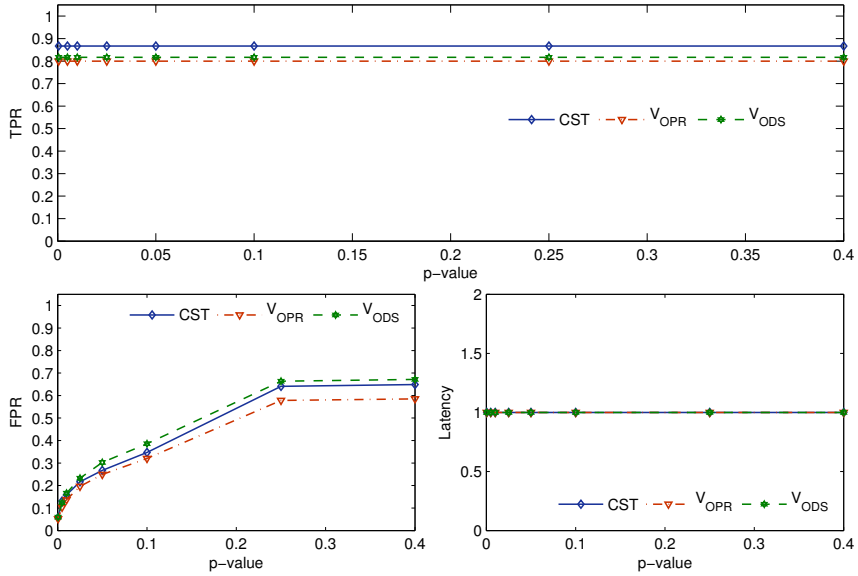


Figure 6.17: Graph for the median TPR, FPR, and the Latency for the T-test classifier in detecting P_{GR} errors as the detection threshold p is varied.

6.6.3 The Quartile-based Classifier

For the Quartile-based classifier, the value of k is varied from 0.0 to 3.0 with an increment of 0.5. The effects on the performance for the Quartile-based classifier as the value of k is varied with faults are shown in Figure 6.18, Figure 6.19, and Figure 6.20.

Results in the figures show that as k increases, the TPR as well as the FPR decreases. This trend of decreasing TPR and FPR is consistent with all faults and

across all scenarios. With both the TPR and the FPR exhibiting a same trend of decreasing in value, it is apparent that some form of tradeoff is required. Nevertheless, in the P_{CP} errors (Figure 6.18), a k less than 1.5 should be used to achieve a TPR that is greater than 0.50 in all scenarios. Also, a k that is greater than zero is required to achieve a FPR that is less than 0.50.

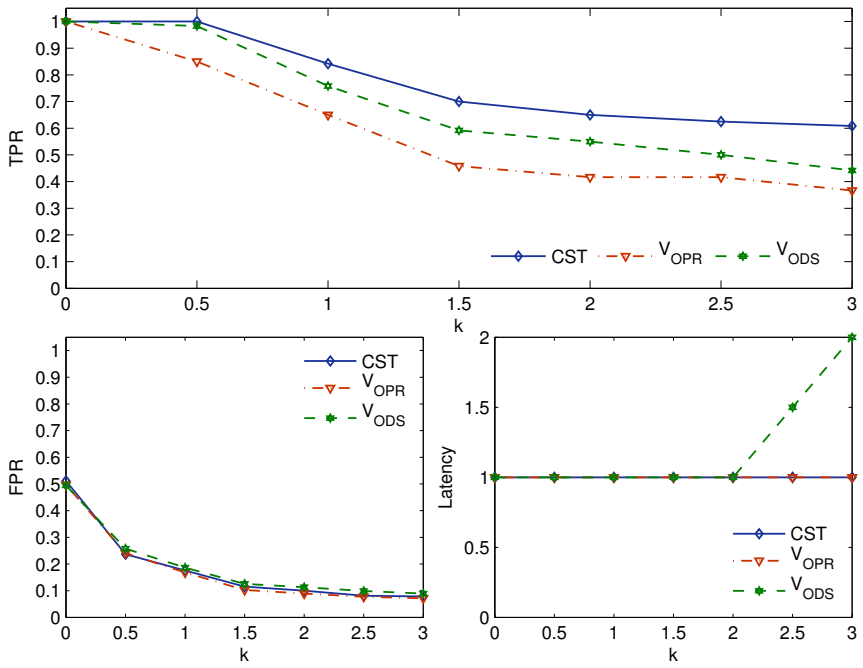


Figure 6.18: Graph for the median TPR, FPR, and the Latency for the Quartile-based classifier in detecting P_{CP} errors as the detection threshold k is varied.

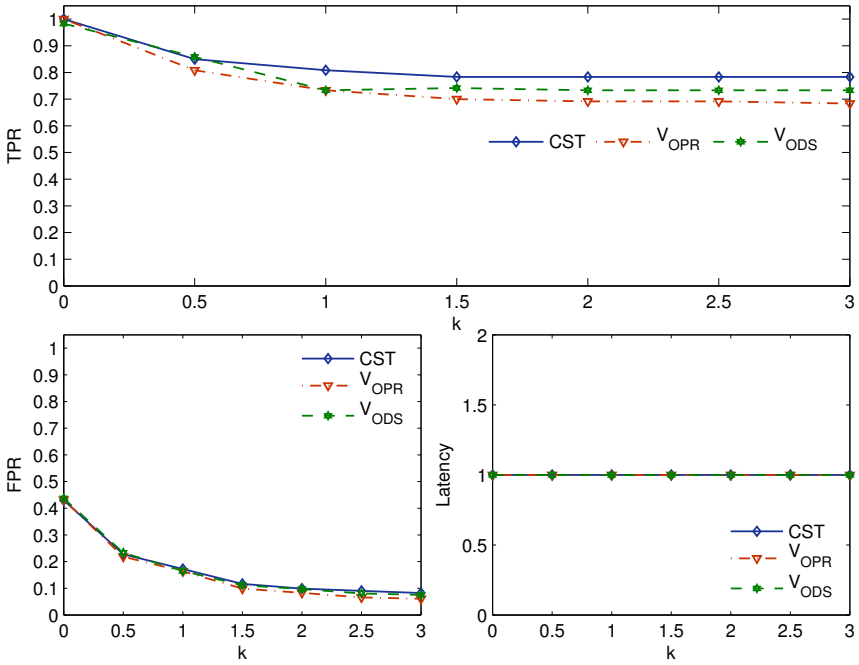


Figure 6.19: Graph for the median `TPR`, `FPR`, and the `Latency` for the Quartile-based classifier in detecting $\texttt{P}_{\texttt{PT}}$ errors as the detection threshold k is varied.

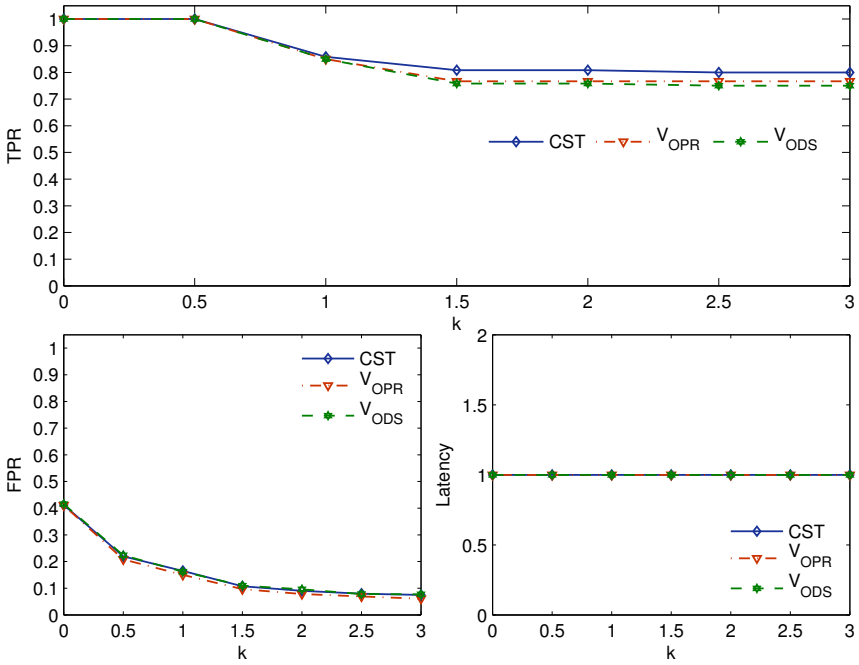


Figure 6.20: Graph for the median `TPR`, `FPR`, and the `Latency` for the Quartile-based classifier in detecting $\texttt{P}_{\texttt{GR}}$ errors as the detection threshold k is varied.

6.6.4 The Q-test Classifier

For the Q-test classifier, all values for the confidence level α in the Q-test table (Table A.1) were tested. The values of α are 80%, 90%, 95%, 96%, and 99% were

tested. These are the values in the Q-test table (Table A.1). The effects on the performance with the faults for the Q-test classifier as α is varied are shown in Figures 6.21, 6.22, and 6.23.

From the figures, an apparent trend of decreasing TPR and FPR as α increases is observed. This trend is consistent with different faults and for all scenarios. With P_{CP} errors, a TPR that is greater than 0.50 in all scenarios is obtained when α is equal and less than 96%. With other faults, all α produced a TPR that is significantly greater than 0.50.

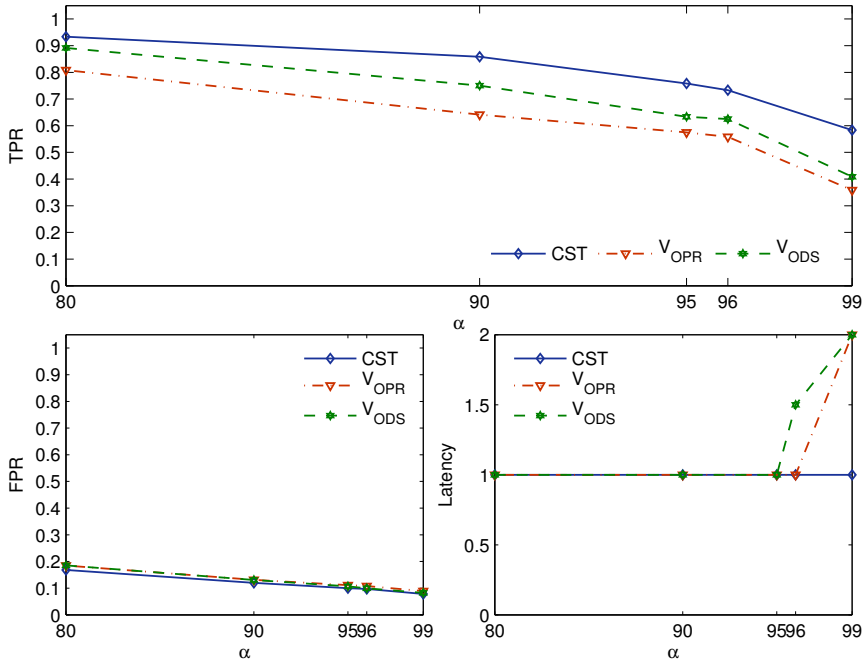


Figure 6.21: Graph for the median TPR, FPR, and the Latency for the Q-test classifier in detecting P_{CP} errors as the detection threshold α is varied.

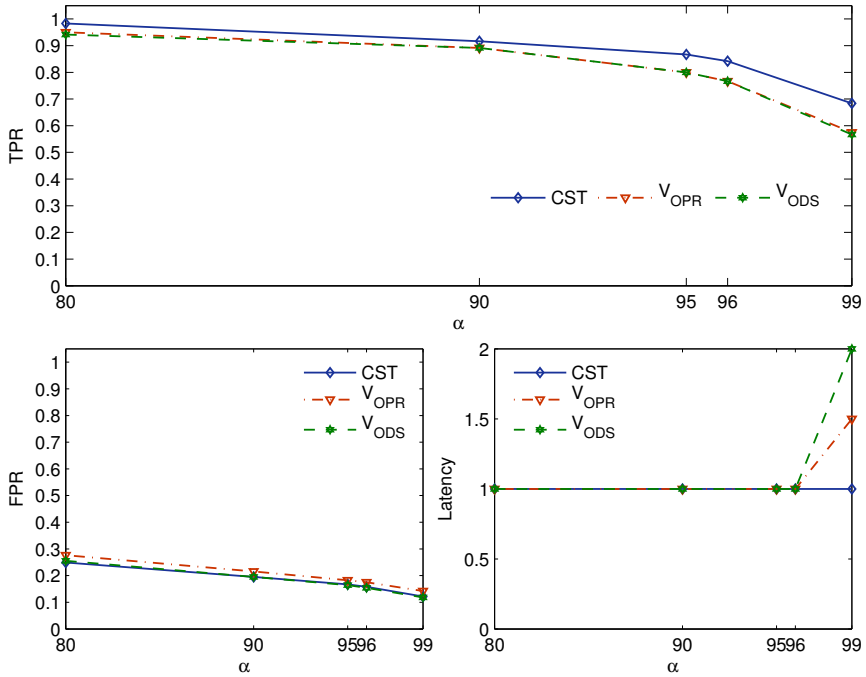


Figure 6.22: Graph for the median `TPR`, `FPR`, and the `Latency` for the Q-test classifier in detecting $\mathtt{P}_{\mathtt{PT}}$ errors as the detection threshold α is varied.

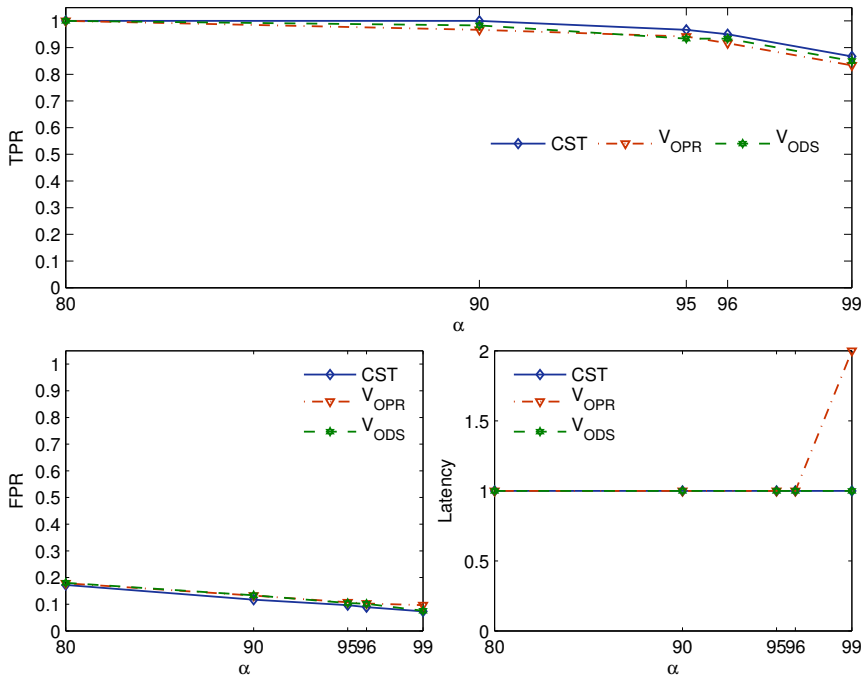


Figure 6.23: Graph for the median `TPR`, `FPR`, and the `Latency` for the Q-test classifier in detecting $\mathtt{P}_{\mathtt{GR}}$ errors as the detection threshold α is varied.

6.6.5 Discussion

From the results on the performance for all classifiers, it can be seen that it is difficult to determine which threshold value provides the best overall performance

when both the TPR and FPR increases or decreases as the detection threshold is varied. Similarly, if given the TPRs and the FPRs of two classifiers, it is often difficult to judge which one is superior if one classifier has a higher TPR whilst the other has a lower FPR. For these cases, the Matthews correlation coefficient (MCC) [142] can be a good indicator. It has been highlighted that there is no a single best metric to compare the performance of the classifiers but the MCC is often regarded as the most appropriate. The calculation of a MCC score is given in Eq. 6.8.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6.8}$$

The results in Table 6.4 are the best result of each classifier based on the MCC score. The detection threshold with the highest MCC score in detecting P_{CP} errors across all scenarios is k = 3 for the ESD classifier, p = 0.0005 for the T-test classifier, k = 0.5 for the Quartile-based classifier, and α = 80% for the Q-test classifier. The full results for each classifier are presented in Table B.1 in Appendix B. Based on the MCC score, it is evident that the default or recommended threshold value is not necessarily the one that give the highest MCC score. Therefore, it would be interesting if the detection threshold can be dynamically adjusted (if possible). This is a potential work that can be investigated further.

In Table 6.4, the first multirow is the performance of the four classifiers in detecting P_{CP} errors in a CST scenario; the second multirow for the results in a V_{OPR} scenario; and the last multirow for the results in a V_{ODS} scenario. The classifier with the highest MCC score for both the parametric classifiers namely the T-test classifier, and for non-parametric classifiers namely the Q-test classifier, is highlighted in bold.

The results for P_{CP} errors in the CST scenario in Table 6.4 show that all classifiers achieved a reasonably good result with a TPR above 0.74 and a FPR below 0.24. Among the classifiers, the Quartile-based classifier has the highest TPR with a perfect detection rate (TPR = 1.00). The other classifiers scored a TPR between 0.74 to 0.93, with the T-test classifier being the classifier with the lowest TPR. However, the Quartile-based classifier also has the highest FPR with a value of 0.23 whilst the T-test classifier has the lowest FPR with a value of 0.06.

These results demonstrated that although a classifier may be able to detect all errors, it does not guarantee a zero false detection. This can be seen in the results for the Quartile-based classifier. This may be an artefact of the random

Table 6.4: The median TPR, FPR, Latency, and MCC in detecting P_{CP} errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	0.88	**0.74**	1.00	**0.93**
	FPR	0.18	**0.06**	0.24	**0.17**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.44	**0.56**	0.44	**0.47**
V_{OPR}	TPR	0.80	**0.60**	0.85	**0.81**
	FPR	0.19	**0.06**	0.24	**0.18**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.38	**0.50**	0.35	**0.38**
V_{ODS}	TPR	0.85	**0.66**	0.98	**0.89**
	FPR	0.20	**0.07**	0.26	**0.19**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.41	**0.51**	0.41	**0.44**

behaviours (stochasticity) in the SRS due to the interactions between the robots as well as between the robots and the environment. If this is an artefact of the detection threshold, then it should be evident from the results with different detection thresholds. However, the full results in Table B.1 in Appendix B showed otherwise. The results showed that as the TPR increases, the FPR also increases. Therefore, a perfect detection with zero false positives is unlikely and a tradeoff between the TPR and the FPR is generally involved.

The tradeoff between the TPR and the FPR poses a difficulty when a comparison is made on which classifier is superior. Thus, this work refers to the MCC score. Based on the MCC score in Table 6.4, the T-test classifier is superior to Quartile-based classifier in detecting P_{CP} errors in a CST scenario. Also, between the parametric classifiers, the T-test classifier is better than the ESD classifier; and between non-parametric classifiers, the Q-test classifier is better than the Quartile-based classifier. This also applies to the detection of P_{CP} errors in V_{OPR} (second multirow) and V_{ODS} (third multirow) scenarios.

Based on the MCC score, the results from the Vargha-Delaney A-test [143] in Table 6.5 revealed that the T-test classifier is a significantly superior parametric classifier when compared to ESD classifier in detecting P_{CP} errors. This may be attributed to the fact that the T-test classifier involves the use of the moving average of data instances for the detection whilst the ESD classifier only uses the current data instance. Therefore, any spike in the data due to the stochasticity of the SRS is treated as an error by the ESD classifier resulting in a false positive but may not by the T-test classifier as the magnitudes of the spikes are reduced by the averaging operation.

Table 6.5: The results of Vargha-Delaney A-test for scientific significance on the MCC score between the four classifiers. A score of 0.5 = no effect, 0.56 = a small effect, 0.64 = a medium effect, and 0.71 = a big effect.

Classifier	ESD/T-test	Quartile/Q-test	T-test/Q-test
CST	0.80	0.80	0.74
V_{OPR}	0.74	0.64	0.72
V_{ODS}	0.66	0.68	0.57

For non-parametric classifiers, although the Q-test classifier has a higher MCC score across all scenarios compared to the Quartile-based classifier the difference is only significant in the CST scenario as indicated by the Vargha-Delaney A-test results in Table 6.5. Finally, between the T-test classifier and the Q-test classifier, the results in Table 6.5 show that the T-test classifier is significantly better at detecting P_{CP} errors in CST and V_{OPR} scenarios.

The best performance for the classifiers with P_{PT} errors is shown in Table 6.6. Based on the highest MCC score for each classifier across all scenarios, the threshold value for the ESD classifier is k = 3, p = 0.0005 for the T-test classifier, k = 3.0 for the Quartile-based classifier, and α = 90% for the Q-test classifier. With the exception of the non-parametric classifiers, the detection threshold that give the highest MCC score for the P_{CP} errors for other classifiers also give the highest MCC score for the P_{PT} errors.

Table 6.6: The median TPR, FPR, Latency, and MCC in detecting P_{PT} errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	0.88	**0.71**	0.78	**0.92**
	FPR	0.17	**0.06**	0.08	**0.20**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.45	**0.56**	0.53	**0.43**
V_{OPR}	TPR	0.83	**0.64**	0.68	**0.89**
	FPR	0.20	**0.06**	0.06	**0.22**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.39	**0.51**	0.51	**0.39**
V_{ODS}	TPR	0.90	**0.73**	0.73	**0.89**
	FPR	0.19	**0.06**	0.08	**0.20**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.43	**0.57**	0.52	**0.42**

Consistent with the results with P_{CP} errors, the results for the parametric classifiers with P_{PT} errors in Table 6.6 show that the T-test classifier has a higher MCC score compared to the ESD classifier. In a similar fashion, for the non-parametric

classifier, the Q-test classifier has a higher MCC score compared to the Quartile-based classifier. Results from the Vargha-Delaney A-test in Table 6.7 show that the difference is significant with the T-test classifier being superior to other classifiers in detecting $\texttt{P}_{\texttt{PT}}$ errors.

Table 6.7: The Vargha-Delaney A-test results on the MCC score between classifiers in detecting $\texttt{P}_{\texttt{PT}}$ errors.

Classifier	ESD/T-test	Quartile/Q-test	T-test/Q-test
CST	0.99	0.90	0.99
$\texttt{V}_{\texttt{OPR}}$	0.97	0.91	0.97
$\texttt{V}_{\texttt{ODS}}$	0.98	0.81	1.00

The best performance for the classifiers with $\texttt{P}_{\texttt{GR}}$ errors is shown in Table 6.8. Based on the highest MCC score for each classifier across all scenarios, the threshold value for the ESD classifier is k = 3, p = 0.0005 for the T-test classifier, k = 3.0 for the Quartile-based classifier, and α = 96% for the Q-test classifier. Interestingly, the α value that gives the highest MCC score with $\texttt{P}_{\texttt{GR}}$ errors is 96% whilst the detection threshold for other classifiers remained the same.

Table 6.8: The median TPR, FPR, Latency, and MCC in detecting $\texttt{P}_{\texttt{GR}}$ errors.

Env.	Metric	ESD	T-test	Quartile	Q-test
CST	TPR	0.98	**0.87**	0.80	**0.97**
	FPR	0.16	**0.06**	0.08	**0.10**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.52	**0.68**	0.57	**0.63**
$\texttt{V}_{\texttt{OPR}}$	TPR	0.95	**0.80**	0.77	**0.94**
	FPR	0.18	**0.05**	0.06	**0.11**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.48	**0.65**	0.56	**0.56**
$\texttt{V}_{\texttt{ODS}}$	TPR	0.95	**0.82**	0.75	**0.93**
	FPR	0.17	**0.06**	0.08	**0.10**
	Latency	1.0	**1.0**	1.0	**1.0**
	MCC	0.51	**0.63**	0.55	**0.60**

Consistent with previous results for the $\texttt{P}_{\texttt{CP}}$ and $\texttt{P}_{\texttt{PT}}$ errors, the T-test classifier is significantly better than the ESD classifier in detecting $\texttt{P}_{\texttt{GR}}$ errors. This is evident from the Vargha-Delaney A-test results in Table 6.9. However, the Q-test classifier is only significantly better than the Quartile-based classifier in detecting $\texttt{P}_{\texttt{GR}}$ errors in the $\texttt{V}_{\texttt{ODS}}$ scenario. Between the T-test classifier and the Q-test classifier, the difference in the performance is only significant in the $\texttt{V}_{\texttt{OPR}}$ scenario and not the other scenarios.

Table 6.9: The Vargha-Delaney A-test results on the MCC score between classifiers in detecting $\mathtt{P}_{\mathtt{GR}}$ errors.

Classifier	ESD/T-test	Quartile/Q-test	T-test/Q-test
CST	0.77	0.74	0.63
$\mathtt{V}_{\mathtt{OPR}}$	0.90	0.58	0.80
$\mathtt{V}_{\mathtt{ODS}}$	0.78	0.75	0.59

Considering the Vargha-Delaney A-test results between the T-test classifier and the Q-test classifier with all faults, it shows that the T-test classifier is superior in most scenarios. However, as mentioned earlier, this is not definitive as it also depends on whether the relative importance between minimising the FPR or maximising the TPR.

Comparing the TPR and the FPR between the T-test classifier and the Q-test classifier, there is not a single classifier that outperformed others in both the FPR and the TPR tests. Although the T-test classifier has a consistently lower FPR compared to the Q-test classifier, the TPR is also consistently lower.

6.7 Summary

In this chapter, a scheme called the collective self-detection (CoDe) scheme was proposed for a data-driven error detection for systems with multiple homogeneous agents such as a robot swarm. In Section 6.1, it was demonstrated that detecting errors from the perspective of a single robot is insufficient when dealing with dynamic environments. Therefore, in Section 6.2, a detection scheme called the CoDe scheme was proposed that utilises the information available in the collective. This scheme is proposed based on the assumptions that the performance of identical agents will be similar even in dynamic environments, and that by cross-referencing one's data with others the presence of a fault can be inferred.

The CoDe scheme was implemented with two parametric statistical classifiers in Section 6.4 and results demonstrated that an adaptive error detection is possible with a TPR greater than 0.50 and a FPR lower than 0.50. Given the positive results with the parametric statistical classifiers, it raises the question as to whether a better performance may be obtained with non-parametric classifiers? Therefore, Section 6.5 evaluates and compares the performance of non-parametric and parametric classifiers. Results showed that indeed the non-parametric classifiers produced a even lower FPR than parametric classifiers. Results from these two sections show that the application of statistical classifiers in the context of the

proposed scheme can produce an adaptive error detection. This is evident from results that the statistical classifiers produced a significantly better performance in detecting errors in dynamic environments compared to a random classifier.

For a more detailed comparative study, a further investigation was conducted in Section 6.6 by varying the detection threshold of the statistical classifiers and evaluate whether there is a change in the performance of the classifiers. Results show that as the detection threshold is varied, a better performance can be achieved. However, there was a difficulty in determining which threshold value provides the best overall performance when both the `TPR` and `FPR` increases or decreases as the detection threshold is varied. Similar difficulty was encountered when one classifier has a higher `TPR` whilst the other has a lower `FPR`. For these reasons, the Matthews correlation coefficient (MCC) was used as a determinant. Based on the MCC score, the T-test classifier is superior compared to other classifiers.

However, as the results for individual performance metrics show, none of the classifiers is superior than the others in both the `TPR` and `FPR` cases. Given that the results so far is positive, it raises the question of whether it is good enough? Ideally, if improvements can still be achieved then it should be continued. The next chapter will investigate the application of an immune inspired classifier to see whether a recently developed bio-inspired classifier called the RDA classifier [115] can produce a similar results or even better performance than currently implemented statistical classifiers.

CHAPTER

SEVEN

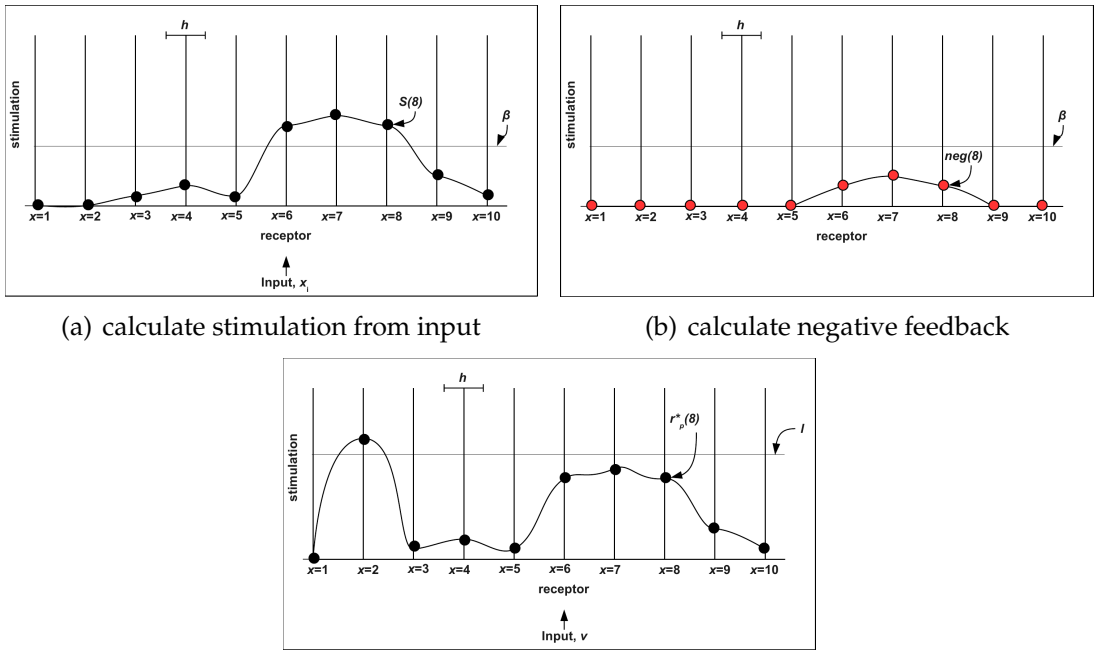
An Immune-Inspired Classifier

This chapter presents work to examine the application of an immune-inspired algorithm within the proposed scheme for adaptive error detection. In Section 7.1, the implementation of the RDA classifier for error detection is presented and the result is compared with previously implemented statistical classifiers. Since the magnitude of the three faults tested were only for a fixed value, a more comprehensive comparisons were subsequently conducted. In Section 7.2, a number of variants of the $\mathtt{P}_{\mathtt{PT}}$ was evaluated. Then, a number of variants of the $\mathtt{P}_{\mathtt{GR}}$ was evaluated in Section 7.3. The performance in detecting these variants between the RDA classifier and the previously implemented statistical classifiers was then compared. As part of a general goal to develop an efficient and lightweight error detection system, Section 7.4 investigates the potential of reducing the FPR of the RDA classifier by increasing the size of detection window as proposed in Christensen [9]. Then, Section 7.5 analyses the relationship between the RDA's parameters and the performance of detection. Results from this experiment provide insights on how the value of these parameters can be changed to optimise particular performance metrics.

7.1 The RDA for Error Detection

The biological inspiration and the development of the RDA algorithm were presented in Section 3.5.4. To illustrate how the RDA can be applied for error detection, Figure 7.1 is an illustrative example. Figure 7.1(a) corresponds to the left equation in Eq. 3.7 whilst Figure 7.1(b) corresponds to the right equation. In Figure 7.1(a), the $r_p(\mathbf{x})$ is calculated with data from other robots. If calculated $r_p(\mathbf{x})$

is greater than the constant β, a negative feedback $r_n(\mathbf{x})$ is generated. To test a data instance v from current robot, Figure 7.1(c) shows updated receptor positions $r_p^t(\mathbf{x})$ corresponds to Eq. 3.8. If $r_p^t(\mathbf{x})$ is greater than ℓ, an error is considered as detected as given in Eq. 3.9.



(a) calculate stimulation from input

(b) calculate negative feedback

(c) calculate updated receptor position

Figure 7.1: An illustration on using the RDA for error detection in a foraging SRS. (a) Calculate the stimulation level of each receptor from input data of neighbouring robots of a communication range. (b) If the stimulation level of a receptor is higher than the base negative barrier β, negative feedback is generated. (c) The stimulation level (receptor position) is updated when input from current robot is added. If any resulting receptor position is higher than ℓ, an error is detected, as seen at receptor x=2.

To demonstrate the implementation of the RDA classifier in CoDe, an example is given below. Assuming an observer robot R1 receives the data stream on `obj` from its neighbours as shown in Table 7.1.

Table 7.1: Data stream received by robot R1.

Robot	R1	Rx_1	Rx_2	Rx_3	Rx_4	Rx_5
Data	1	6	5	4	6	5

Let's set the number of equally spaced receptors to 21 with a minimum value of 0 to a maximum value of 8. Total stimulation for each receptor in Eq. 3.7 is calculated using Gaussian kernel with kernel width h[*obj*] = 1.0, β=0.01, b=0.1,

gb=1.1, and a=1. During the initialisation stage (Eq. 3.7), only neighbourhood data is used to calculate the receptor stimulation and negative feedback for each receptor. In this example, the data from neighbours are 6, 5, 4, 6, 5. Then at the testing stage (Eq. 3.8 and Eq. 3.9), data from the current robot is tested.
To determine whether robot R1 is faulty, the RDA classifier works as follows:

Step 1 Initialisation

- Calculate $r_p(x)$ (Left Eq. 3.7):
  $r_p(0)$=0.0000, $r_p(0.4)$=0.0001, $r_p(0.8)$=0.0004, $r_p(1.2)$=0.0014, $r_p(1.6)$=0.0042, $r_p(2.0)$=0.0105, $r_p(2.4)$=0.0232, $r_p(2.8)$=0.0450, $r_p(3.2)$=0.0772, $r_p(3.6)$=0.1188, $r_p(4.0)$=0.1651, $r_p(4.4)$=0.2094, $r_p(4.8)$=0.2434, $r_p(5.2)$=0.2593, $r_p(5.6)$=0.2523, $r_p(6.0)$=0.2226, $r_p(6.4)$=0.1764, $r_p(6.8)$=0.1242, $r_p(7.2)$=0.0770, $r_p(7.6)$=0.0416, $r_p(8.0)$=0.0195
- Calculate $r_n(x)$ (Right Eq. 3.7):
  $r_n(0)$=0.0000, $r_n(0.4)$=0.0000, $r_n(0.8)$=0.0000, $r_n(1.2)$=0.0000, $r_n(1.6)$=0.0000, $r_n(2.0)$=0.0005, $r_n(2.4)$=0.0132, $r_n(2.8)$=0.0350, $r_n(3.2)$=0.0672, $r_n(3.6)$=0.1088, $r_n(4.0)$=0.1551, $r_n(4.4)$=0.1994, $r_n(4.8)$=0.2334, $r_n(5.2)$=0.2493, $r_n(5.6)$=0.2423, $r_n(6.0)$=0.2126, $r_n(6.4)$=0.1664, $r_n(6.8)$=0.1142, $r_n(7.2)$=0.0670, $r_n(7.6)$=0.0316, $r_n(8.0)$=0.0095

Step 2 Testing

- Calculate l: 0.0665
- Calculate new receptor position (Eq. 3.8):
  $r_p(0)$=0.0444, $r_p(0.4)$=0.0611, $r_p(0.8)$=0.0717, $r_p(1.2)$=0.718.
  $r_p(1.6)$=0.0615, $r_p(2.0)$=0.0449, $r_p(2.4)$=0.0166, $r_p(2.8)$=0.0000, $r_p(3.2)$=0.0000, $r_p(3.6)$=0.0000, $r_p(4.0)$=0.0000, $r_p(4.4)$=0.0000, $r_p(4.8)$=0.0000, $r_p(5.2)$=0.0000, $r_p(5.6)$=0.0000, $r_p(6.0)$=0.0000, $r_p(6.4)$=0.0000, $r_p(6.8)$=0.0000, $r_p(7.2)$=0.0000, $r_p(7.6)$=0.0000, $r_p(8.0)$=0.0000
- Classify R1's data (Eq. 3.9): Since $r_p(0.8) \geq l$ and $r_p(1.2) \geq l$, therefore R1's data is errorneous.

7.1.1 Experimental Setup

Experimental Objective: To evaluate the performance of the RDA classifier in CoDe for error detection in both non-dynamic and dynamic environments, and compare it with those of the statistical classifiers.

The objective of this experiment is to examine the performance of the RDA classifier in CoDe for adaptive error detection and to determine whether there is significant difference between the performance of the RDA classifier and the implemented statistical classifiers.

The parameter value for the various RDA parameters is influenced by the nature of the input data as well as the relative importance of each performance metric. In this section, the results are based on the following: the number of receptors = 21, β=0.01, b=0.1, gb=1.1, a=1.0, and the kernel is Gaussian kernel defined as

$$K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(\mathbf{x}-\mathbf{x}_i)^2}{2h^2}}.$$

These values are obtained from manual inspections on the performance of the RDA classifier for some preliminary runs.

Using the data from Section 6.6.5, the RDA classifier was applied to detect the same $\mathtt{P}_{\mathtt{CP}}$ (complete), $\mathtt{P}_{\mathtt{PT}}$ (partial), and $\mathtt{P}_{\mathtt{GR}}$ (gradual) errors. The performance for the RDA classifier in detecting each error is presented in tabular form. In each table, the best results for the parametric and non-parametric classifiers namely the T-test and the Q-test classifiers are also included for the convenience of comparison.

7.1.2 Experimental Results

Table 7.2 shows the results for the $\mathtt{P}_{\mathtt{CP}}$ errors. The results in the table show that the RDA classifier has a high TPR and a low FPR when compared with the other two classifiers. The TPR for the RDA classifier is the highest in the CST (constant OPR) and $\mathtt{V}_{\mathtt{ODS}}$ (varying object distribution) scenarios when compared to the T-test and Q-test classifiers. For the FPR, it is significantly lower than the Q-test classifier and only slightly higher than the T-test classifier. Particularly, the difference to the T-test classifier is less than 0.01 for the CST scenario, 0.04 for the $\mathtt{V}_{\mathtt{OPR}}$ (varying OPR) scenario, and 0.02 for the $\mathtt{V}_{\mathtt{ODS}}$ scenario. This is an interesting result as it hints that the RDA classifier may be a particularly suitable classifier for current problem.

Referring to the MCC score, the RDA classifier has the highest value in all scenarios and significantly better than the T-test and the Q-test classifiers except in a $\mathtt{V}_{\mathtt{OPR}}$ scenario. However, as revealed by the Vargha-Delaney A-test results in Table 7.3, the difference is not significant ($A < 0.71$).

The results in Table 7.4 for the $\mathtt{P}_{\mathtt{PT}}$ errors are consistent with the results for the $\mathtt{P}_{\mathtt{CP}}$ errors in which the RDA classifier has a very high TPR and a very low

Table 7.2: The median TPR, FPR, Latency, and MCC in detecting P_{CP} errors.

Env.	Metric	RDA	T-test	Q-test
CST	TPR	**1.00**	0.74	0.93
	FPR	**0.07**	0.06	0.17
	Latency	**1.0**	1.0	1.0
	MCC	**0.72**	0.56	0.47
V_{OPR}	TPR	0.78	**0.60**	0.81
	FPR	0.09	**0.06**	0.18
	Latency	1.0	**1.0**	1.0
	MCC	0.49	**0.50**	0.38
V_{ODS}	TPR	**0.98**	0.66	0.89
	FPR	**0.08**	0.07	0.19
	Latency	**1.0**	1.0	1.0
	MCC	**0.67**	0.51	0.44

Table 7.3: The Vargha-Delaney A-test results on the MCC score between the RDA, T-test, and Q-test classifiers.

	Classifier	CST	V_{OPR}	V_{ODS}
P_{CP}	RDA/T-test	0.97	0.52	0.94
	RDA/Q-test	1.00	0.85	0.99
P_{PT}	RDA/T-test	0.99	0.95	0.99
	RDA/Q-test	1.00	1.00	1.00
P_{GR}	RDA/T-test	0.79	0.58	0.88
	RDA/Q-test	0.92	0.99	0.98

FPR. The MCC score for the RDA classifier is higher than the T-test and the Q-test classifiers in all scenarios. From the Vargha-Delaney A-test results in Table 7.3, the RDA classifier significantly outperformed the other two classifiers in the detection of P_{PT} errors.

The results for the P_{CP} and P_{PT} errors give a clear indication that the RDA classifier can be a more effective classifier for the error detection task in this thesis. Therefore, it is expected that a similar result may be obtained for the RDA classifier in detecting P_{GR} errors. The results in Table 7.5 confirm this. The results show that the RDA is superior ($A \geq 0.71$) to the T-test and Q-test classifier.

The positive results on the performance of the RDA classifier is promising. However, it will be noted that the simulated P_{PT} and P_{GR} errors were only at a particular magnitude. Therefore, it would be worth varying the magnitude of the P_{PT} and P_{GR} to see whether the same results are obtained. Also, ideally, it would be beneficial to be able to detect the errors as soon as possible before it propagates further in the system. In addition, altering the magnitude of the P_{PT}

Table 7.4: The median TPR, FPR, Latency, and MCC in detecting P_{PT} errors.

Env.	Metric	RDA	T-test	Q-test
CST	TPR	**1.00**	0.71	0.92
	FPR	**0.06**	0.06	0.20
	Latency	**1.0**	1.0	1.0
	MCC	**0.75**	0.56	0.43
V_{OPR}	TPR	**1.00**	0.64	0.89
	FPR	**0.09**	0.06	0.22
	Latency	**1.0**	1.0	1.0
	MCC	**0.64**	0.51	0.39
V_{ODS}	TPR	**1.00**	0.73	0.89
	FPR	**0.07**	0.06	0.20
	Latency	**1.0**	1.0	1.0
	MCC	**0.72**	0.57	0.42

Table 7.5: The median TPR, FPR, Latency, and MCC in detecting P_{GR} errors.

Env.	Metric	RDA	T-test	Q-test
CST	TPR	**1.00**	0.87	0.95
	FPR	**0.06**	0.06	0.09
	Latency	**1.0**	1.0	1.0
	MCC	**0.74**	0.68	0.63
V_{OPR}	TPR	**1.00**	0.80	0.92
	FPR	**0.09**	0.05	0.10
	Latency	**1.0**	1.0	1.0
	MCC	**0.66**	0.65	0.56
V_{ODS}	TPR	**1.00**	0.82	0.93
	FPR	**0.07**	0.06	0.10
	Latency	**1.0**	1.0	1.0
	MCC	**0.70**	0.63	0.60

and P_{GR} can reveal how flexible the classifiers are in detecting the variants.

7.2 Variations in Partial Failure to the Wheels, P_{PT}

For completeness, the variants of the P_{PT} were tested with all classifiers including the ESD and the Quartile-based classifiers to investigate how variations in the P_{PT} affect the performance of the classifiers.

7.2.1 Experimental Setup

Experimental Objective: To evaluate the performance of implemented classifiers in detecting a number of variants of the $\texttt{P}_{\texttt{PT}}$ in both non-dynamic and dynamic environments.

Eight variants of the $\texttt{P}_{\texttt{PT}}$ were tested: $\texttt{P}_{\texttt{PT}}$ = {45, 60, 75, 80, 85, 90, 95, 105} $\times 10^{-3}$ m.s^{-1}. A $\texttt{P}_{\texttt{PT}}$ of 0 $\times 10^{-3}$ m.s^{-1} is equivalent to the $\texttt{P}_{\texttt{CP}}$. Recall that the $\texttt{P}_{\texttt{PT}}$ is simulated by instantly reducing the speed of a faulty robot from a normal speed of 150 $\times 10^{-3}$ m.s^{-1} to the specified speed. The magnitude of the $\texttt{P}_{\texttt{PT}}$ can influence how the faults may be manifested as errors in the data; the more severe is the $\texttt{P}_{\texttt{PT}}$, i.e. slower movement of the wheels, its effects should be more apparent in the data.

Each $\texttt{P}_{\texttt{PT}}$ variant was injected independently to a single robot in the SRS in $\texttt{CST}$, $\texttt{V}_{\texttt{OPR}}$, and $\texttt{V}_{\texttt{ODS}}$ scenarios. For each scenario, twenty repeated runs were conducted. From the twenty runs, the median value of each performance metric was calculated. For each statistical classifier, the same detection thresholds as in Section 6.6 has been tested and the best results based on the MCC score are presented in this section. The full results are presented in Appendix C and will be referred to as appropriate.

7.2.2 Experimental Results

The results for the $\texttt{P}_{\texttt{PT}}$ variants in a $\texttt{CST}$ scenario are shown in Figure 7.2. The points on the graphs in each figure are the medians and lines connecting the points are drawn for clarity. The labels on the x-axes are the $\texttt{P}_{\texttt{PT}}$ variants, in which a higher value for motor speed means that the $\texttt{P}_{\texttt{PT}}$ is more subtle.

The results show that as the magnitude of $\texttt{P}_{\texttt{PT}}$ decreases (higher value for the $\texttt{P}_{\texttt{PT}}$), the errors became more difficult to detect. Take the results for the Quartile-based classifier as an example, the $\texttt{TPR}$ for $\texttt{P}_{\texttt{PT}}$ = 45 $\times 10^{-13}$ m.s^{-1} is 0.80 and the $\texttt{TPR}$ continues to drop to end up with a $\texttt{TPR}$ of only 0.60 at $\texttt{P}_{\texttt{PT}}$ = 105 $\times 10^{-3}$ m.s^{-1}. This general trend of decreasing $\texttt{TPR}$ was observed with all classifiers. This is an unsurprising result and rather an expected outcome because if the $\texttt{P}_{\texttt{PT}}$ is too subtle to be manifested on the data, it is unlikely to be detected. This is evident for the results for the T-test and the RDA classifiers when the $\texttt{P}_{\texttt{PT}}$ is greater than 95 $\times 10^{-3}$ m.s^{-1}, the $\texttt{TPR}$ is less than 0.50.

A further investigation on the data for $\texttt{P}_{\texttt{PT}}$ = 105 $\times 10^{-3}$ m.s^{-1} in the $\texttt{CST}$ scenario as shown in Figure 7.3 revealed that there are overlapping in the data (the $\texttt{obj}$ and the $\texttt{eng}$) between the faulty robot R1 and fault-free robots in the system.

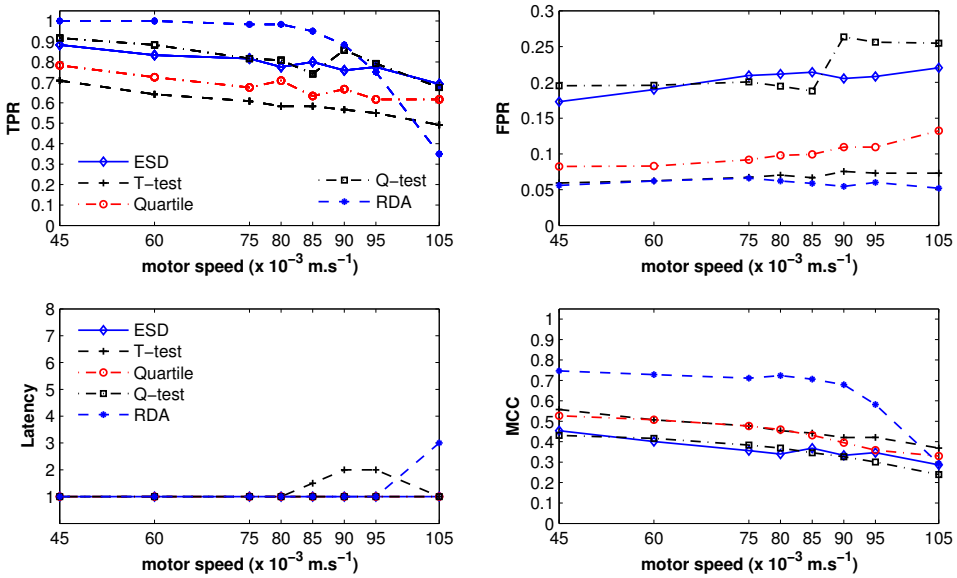
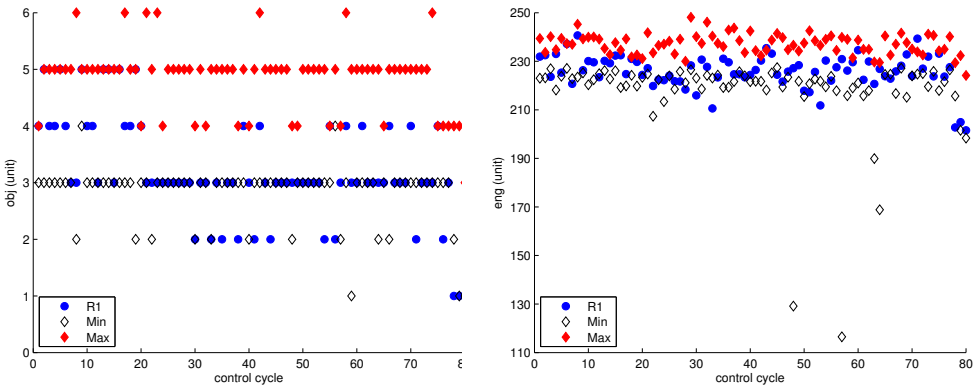


Figure 7.2: The median TPR, FPR, Latency and MCC for all classifiers as the P_{PT} is varied in a CST scenario.

This is likely the reason for the poor performance for the RDA and the T-test classifier at this magnitude. As for other classifiers, a clear trend of increasing FPR indicates that more false positives have been produced.



(a) number of objects collected, obj (b) energy used, eng

Figure 7.3: Scatterplots showing the minimum and maximum value for the obj and eng in the system, and for faulty robot R1 at each control cycle in a CST scenario.

For the FPR, as the P_{PT} becomes more subtle the FPR also increases. However this is only a slight increase. However, for the Q-test classifier, a steep increase in the FPR can be seen with $P_{PT} = 90 \times 10^{-3}$ m.s^{-1} and it remains reasonably constant for the more subtle P_{PT} (i.e. $P_{PT} > 90 \times 10^{-3}$ m.s^{-1}). This steep increase indicates that the Q-test classifier has become significantly less capable of differentiating

between normal and erroneous instances as P_{PT} becomes more subtle.

In a V_{OPR} scenario, the same pattern of an increasing FPR and a decreasing TPR is also observed as shown in Figure 7.4. This result further demonstrates the difficulty in detecting errors in systems deployed in dynamic environments. However, albeit with a lower performance, the results suggest that the errors can still be detected by all classifiers with a TPR that is significantly greater than 0.50 and a FPR that is significantly lower than 0.50.

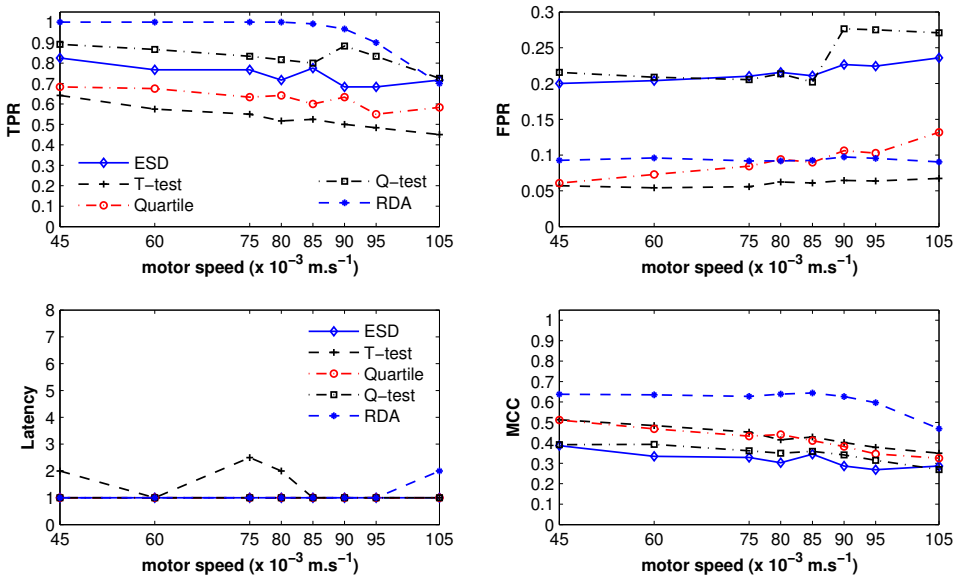


Figure 7.4: The median TPR, FPR, Latency and MCC for the classifiers as the P_{PT} is varied in V_{OPR}.

The results for the P_{PT} variants in a V_{ODS} scenario in Figure 7.5 are similar to previous experiments. The MCC score is higher for all classifiers in a V_{ODS} scenario when compared to a V_{OPR} scenario.

Similar to the results in CST and V_{OPR} scenarios, as the P_{PT} became more subtle in a V_{ODS} scenario, the performance of the classifiers decreases (see Figure 7.5). Again, this is unsurprising but it does raise the question of how subtle can the P_{PT} scenario be in real robots. This is a limitation of this experiment as the variations of P_{PT} were synthetic as to the best of our knowledge no published data is available on how much reduction in the motor speed when real robots experience partial faults to the wheels. Addressing this issue is future work. However, results from this experiment allow us to look at the range of P_{PT} errors that can be detected and the associated performance (TPR, FPR, MCC) by all classifiers. For example, if a target TPR of 0.80 is desired in all scenarios, then the results show that only the ESD, the Q-test, and the RDA classifiers can meet the performance target. On top of that, it is only achievable with P_{PT} less than 80 $\times 10^{-3}$ m.s^{-1} for

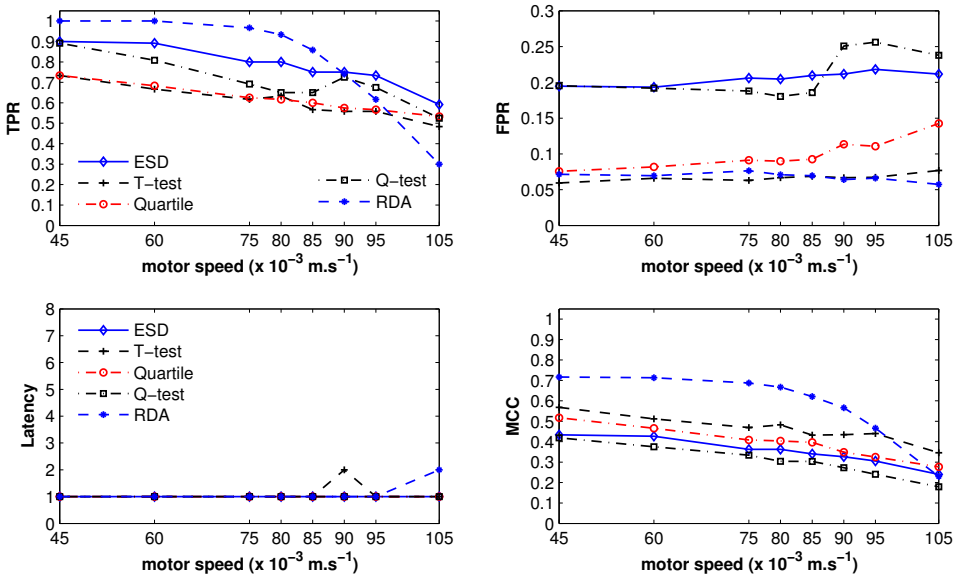


Figure 7.5: The median TPR, FPR, Latency and MCC for the classifiers as the P_{PT} is varied in V_{ODS}.

the Q-test classifier, P_{PT} less than 60 $\times 10^{-3}$ m.s^{-1} for the ESD classifier, and P_{PT} equals to and less than 85 $\times 10^{-3}$ m.s^{-1} for the RDA classifier.

The results for the TPR are in Figure 7.2, Figure 7.4, and Figure 7.5 show that the RDA classifier has the highest TPR compared to other classifiers with P_{PT} less than 90 $\times 10^{-3}$ m.s^{-1}. Recall that it is suspected that with P_{PT} greater than 95 $\times 10^{-3}$ m.s^{-1}, the faults might be too subtle to be manifested on the data and thus may not be differentiable from a fault-free condition by the RDA classifier. However, the FPR for the RDA classifier remained the same around 0.05 for the P_{PT} variants. Overall, the MCC score showed that the RDA classifier significantly outperformed other classifiers for P_{PT} less than 95 $\times 10^{-3}$ m.s^{-1}. These results demonstrates that the RDA classifier can be a more effective classifier to achieve an adaptive error detection for P_{PT} in certain dynamic environments with performance significantly better than the implemented statistical classifiers.

7.3 Variations in Gradual Failure to the Wheels, P_{GR}

This experiment investigates the ability of the classifiers to detect a number of variants of the P_{GR}. The variation in the P_{GR} has an effect on the time taken for a fault to be manifested as an error on the data. The more subtle a P_{GR}, the longer it takes for the errors to be manifested on the data. Thus, a change in the Latency is expected as the P_{GR} is varied.

7.3.1 Experimental Setup

Experimental Objective: To evaluate the performance of implemented classifiers in detecting a number of variants of the $\mathtt{P}_{\mathtt{GR}}$ in both non-dynamic and dynamic environments.

Eleven variants of the $\mathtt{P}_{\mathtt{GR}}$ were tested: $\mathtt{P}_{\mathtt{GR}}$= {5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100} $\times 10^{-5}$ m.s^{-2}. The bigger the significant digits of the $\mathtt{P}_{\mathtt{GR}}$, the more severe is the failure. This means that a $\mathtt{P}_{\mathtt{GR}}$ of 100 $\times 10^{-5}$ m.s^{-2} is more severe than a $\mathtt{P}_{\mathtt{GR}}$ of 80 $\times 10^{-5}$ m.s^{-2}, and should not be detected later.

Each variant of the $\mathtt{P}_{\mathtt{GR}}$ was independently injected to a robot in a SRS in `CST`, $\mathtt{V}_{\mathtt{OPR}}$, and $\mathtt{V}_{\mathtt{ODS}}$ scenarios. Twenty repeated runs per scenario were conducted. The classifiers are then applied and a median is calculated for each performance metric. For each classifier, the same detection thresholds as in previous experiments were tested and the best results are presented in this section. The full results are presented in Appendix C.

7.3.2 Experimental Results

For the variants of $\mathtt{P}_{\mathtt{GR}}$ in a `CST` scenario, the results in Figure 7.6 show that as the magnitude of the $\mathtt{P}_{\mathtt{GR}}$ increases, detecting the errors becomes easier and the time to identify the first occurrence of the error is shorter. This can be seen from the increase in the `TPR` and a decrease in the `Latency`. In the top left figure, an increase in the `TPR` can be seen for most classifiers as the magnitude of $\mathtt{P}_{\mathtt{GR}}$ increases from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}. In the bottom left figure, as the magnitude of the $\mathtt{P}_{\mathtt{GR}}$ increases, the `Latency` decreases. This is an expected result as the magnitude of $\mathtt{P}_{\mathtt{GR}}$ increases, the errors became more apparent on the data and thus can be significantly different from those for fault-free conditions.

The increase in the `TPR` for all classifiers between two consecutive $\mathtt{P}_{\mathtt{GR}}$ variants is only about 1%. This is an artefact of the variants of the $\mathtt{P}_{\mathtt{GR}}$ tested because the effect of $\mathtt{P}_{\mathtt{GR}}$ is accumulative. This means that as the time progresses, the faulty robot will move with gradually reducing speed until it stops completely. Therefore, the difference in the performance between the two variants of $\mathtt{P}_{\mathtt{GR}}$ is influenced by the difference in the time taken to reach a complete stop by a faulty robot. The number of control cycles (T_s) that have passed when the faults becomes permanent is given by the equation:

$$T_s = (\frac{initial\ speed}{\mathtt{P}_{\mathtt{GR}}})/control\ cycle's\ length \tag{7.1}$$

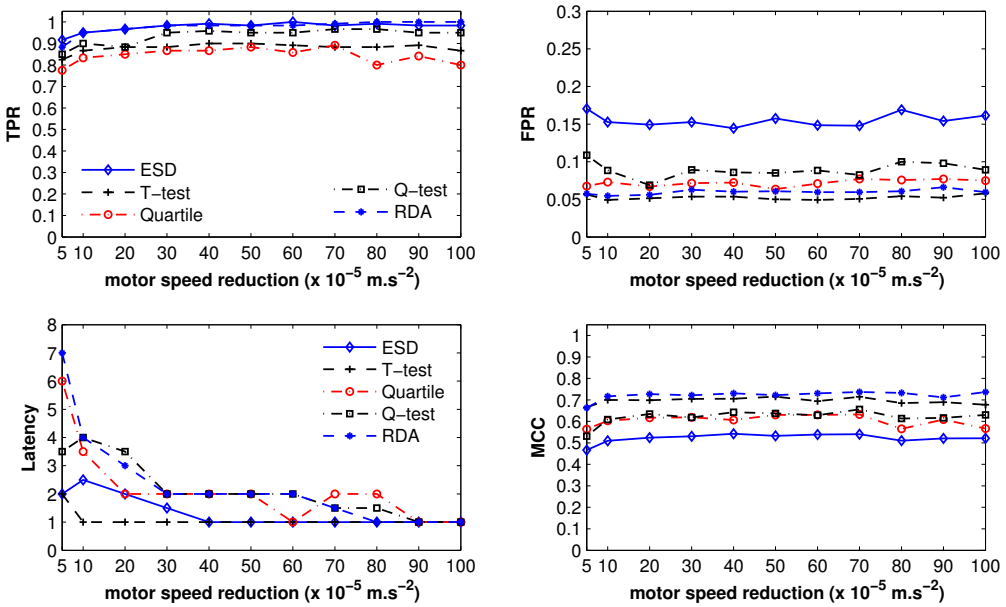


Figure 7.6: The median TPR, FPR, Latency, and MCC for the classifiers as the P_{GR} is varied in a CST scenario.

For example, with P_{GR} of 5 $\times 10^{-5}$ m.s^{-2}, $T_s = (\frac{0.15}{0.00005})/250$ = 12 control cycles. Similarly, with P_{GR} of 10 $\times 10^{-5}$ m.s^{-2}, $T_s = (\frac{0.15}{0.00010})/250$ = 6 control cycles. Therefore, if a classifier is working properly and the operational condition is the same for two P_{GR} variants, (in principle) the maximum difference in the TPR is about (12-6)/60 = 0.10 (10%). In practise, this difference in the TPR is likely to be lower as the P_{GR} might be detected before the faulty robot comes to a complete stop.

For the FPR, as the magnitude of the P_{GR} increases, the FPR also increases but only slightly. For most classifiers, the FPR remained roughly the same as the P_{GR} is varied. Overall, the FPR is less than 0.10 except for the ESD classifier.

The effect of the P_{GR} variants on the Latency is more apparent when compared to the P_{PT}. The results on the Latency in the bottom left figure in Figure 7.6 show that the Latency is high when the P_{GR} is subtle as evident with P_{GR} of 5 $\times 10^{-5}$ m.s^{-2}. Then, as the magnitude of P_{GR} increases, the Latency decreases until it reaches a value of 1.0 with P_{GR} of 90 $\times 10^{-5}$ m.s^{-2}. Again, this is an expected outcome with the P_{GR} due to its accumulative nature.

Overall, almost all classifiers managed to achieved a TPR greater than 0.80 and a FPR less than 0.10 for the P_{GR} variants. Among these classifiers, the classifier with the highest MCC score is the RDA classifier for all P_{GR} variants.

The results for the variants of the P_{GR} in V_{OPR} (Figure 7.7) and V_{ODS} (Figure 7.8) scenarios are consistent with those in a CST scenario. A slight increase in the TPR can be seen as the magnitude of the P_{GR} increases. Again, this is as expected as

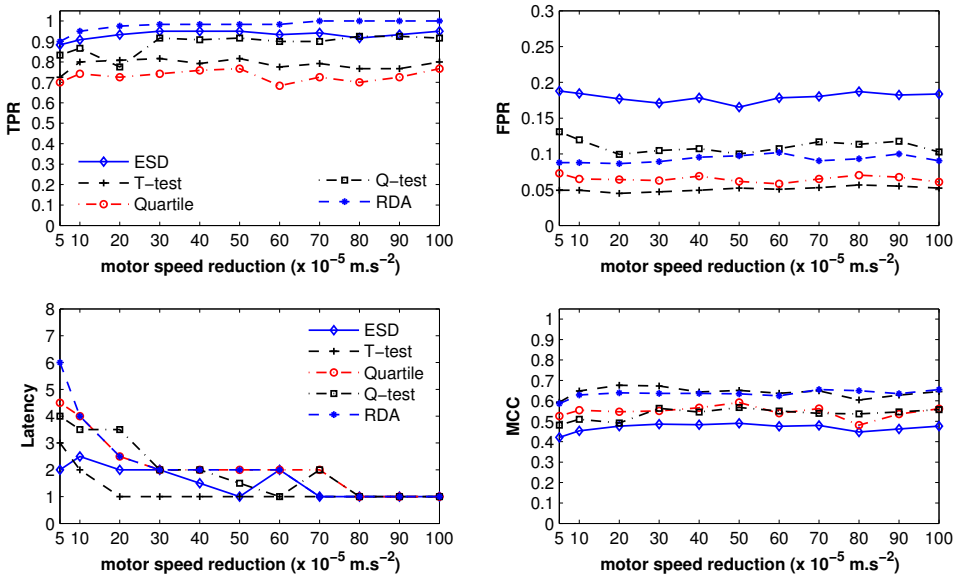


Figure 7.7: The median TPR, FPR, Latency, and MCC for the classifiers as the P_{GR} is varied in a V_{OPR} scenario.

a more severe failure should be easier to detect. Similarly a slight increase in the FPR can also be seen. Finally, the decrease in the Latency as the magnitude of P_{GR} increases is also observed. However, all classifier reached a Latency of 1.0 with P_{GR} greater and equals to 80 $\times 10^{-5}$ m.s^{-2}.

The results for the variants of P_{GR} in V_{OPR} and V_{ODS} scenarios for all classifiers show that all classifiers managed to achieve a TPR that is significantly greater than 0.50 and a FPR that is significantly less than 0.50. These results provide a good indicator of the ability of the classifiers implemented for an adaptive error detection in certain dynamic environments particularly in V_{OPR} and V_{ODS} scenarios.

The performance of the RDA classifier for the variants of the P_{GR} in CST, V_{OPR}, and V_{ODS} scenarios that consistently superior to other classifiers. This gives a clear indicator that the RDA classifier can be as at least as effective as the other implemented classifier for current problem.

Due to the potential of the RDA classifier, the work in the next sections will look at the aspects of minimising the FPR and sensitivity analysis on the RDA's parameters.

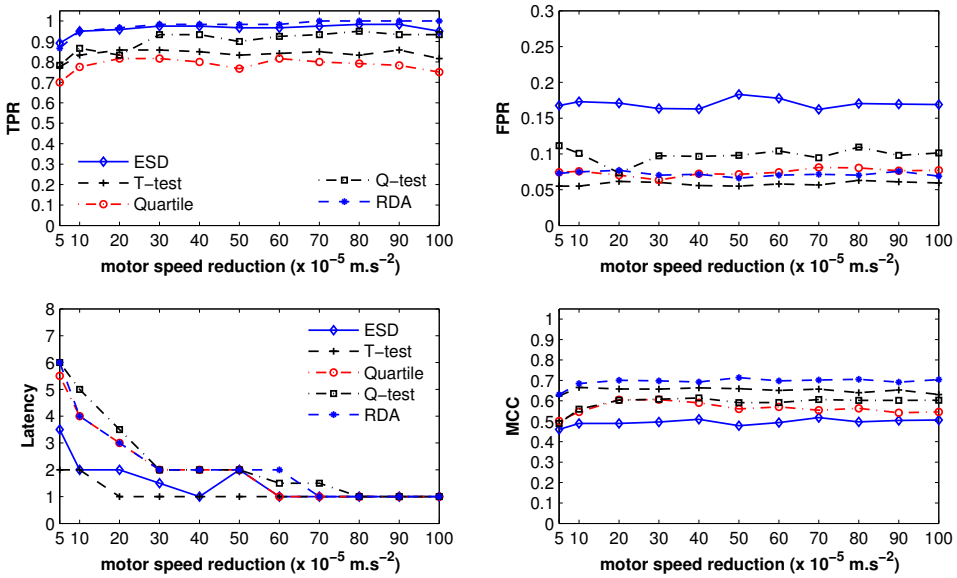


Figure 7.8: The median TPR, FPR, Latency, and MCC for the classifiers as the P_{GR} is varied in a V_{ODS} scenario.

7.4 Reducing the False Positive Rate, FPR

For many robotic task, a latency of a few control cycles is unlikely to cause significant negative impacts on the performance as long as a fault is eventually detected [9]. However, a high false positives is often undesirable as recovery actions are often expensive and time consuming. Therefore, minimising the false positives is often desirable.

One way of reducing the number of false positives is to increase the detection threshold of the chosen classifier and it has been explored in Chapter 6. Full results as presented in Appendix B show that by increasing the detection threshold, the number of errors not being detected also increases. Therefore, a tradeoff is required and for this work a detection threshold that give the highest MCC score considering both the TPR and FPR was selected.

Another way to reduce the number of false positives is by increasing the size of the detection window (DW) which was investigated in Christensen et al. [12]. Their work showed that the number of false positives can be significantly reduced as the detection window is increased. In Christensen et al. [12], it was referred to as *smoothing with a moving average* in which a moving average over 25 control cycles of the classifier's output was used to determine whether an error has occurred. In other words, the size of the DW is 25. For the current implementation, the performance of the classifier is based on the output at current control cycle, i.e., DW = 1.

In principle, an increase in the size of the DW will result in a decrease of the number of false positives. However, it also likely to result in more errors not being detected, i.e. a decrease in the TPR. Thus, the choice of the size of the DW is crucial. Aggregating the results for the number of false positives for the RDA classifier in detecting P_{CP} errors, Figure 7.9 shows the distribution for the length of false positive (ℓ_{FP}), i.e. how long the series of false positives last. On the y-axis is the frequency of the ℓ_{FP} over the total number of false positives, and on the x-axis is the ℓ_{FP}. A value of 2 on the x-axis means a spike in the false positive lasts for two consecutive control cycles. From the figure, it can be seen that more than 80 percent of the false positives last for only one control cycle. Therefore, if the size of the DW is increased to 2, more than 80 percent of the false positives can be eliminated.

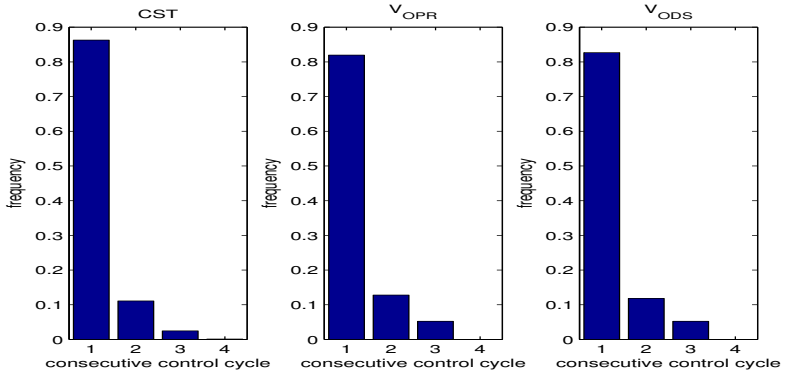


Figure 7.9: Histogram showing the distribution on the length of false positives.

7.4.1 Experimental Setup

Experimental Objective: To evaluate the potential of reducing the FPR by increasing the size of the detection window from 1 to 2.

To further investigate the effect of increasing the size of the DW on the performance metrics, the same data in Section 7.1.2 was analysed with the RDA classifier with a DW of size 2 (RDA-DW2) and the results are compared with the original RDA classifier.

7.4.2 Experimental Results

Table 7.6 shows the differences in performance for detecting P_{CP} errors, Figure 7.10 for detecting P_{PT} errors, and Figure 7.11 for detecting P_{GR} errors. A positive

value in the table or graph means an increase in the value, a zero means no difference, and a negative value means a decrease in the value for each metric. For example, the value -0.02 for TPR in the CST scenario in Table 7.6 means that with the RDA-DW2 classifier, the TPR for detecting P_{CP} errors in the CST scenario is 0.02 (2.0 percent) less than the original RDA classifier.

Table 7.6: The difference in TPR, FPR, Latency, and MCC between the RDA-DW2 classifier and the original RDA classifier in detecting P_{CP} errors.

Metric		TPR	FPR	Latency	MCC
CST	RDA (A1)	1.00	0.07	1.0	0.72
	RDA-DW2 (A2)	0.98	0.01	2.0	0.93
	Difference (A2-A1)	-0.02	-0.06	1.0	0.21
V_{OPR}	RDA (A1)	0.78	0.09	1.0	0.49
	RDA-DW2 (A2)	0.67	0.01	2.0	0.66
	Difference (A2-A1)	-0.11	-0.08	1.0	0.17
V_{ODS}	RDA (A1)	0.98	0.08	1.0	0.69
	RDA-DW2 (A2)	0.95	0.01	2.0	0.91
	Difference (A2-A1)	-0.03	-0.07	1.0	0.22

It can be seen from the results in Table 7.6, Figure 7.10 and Figure 7.11 that by increasing the size of DW from 1 to 2, a lower TPR and FPR is obtained. However, even with a decrease in TPR, a better overall performance as measure with MCC score is achieved as evident with positive values for the MCC scores. As for the Latency, an increase from one control cycle to greater than or equal to two control cycles was observed. This was expected. Note that the minimum Latency ($Latency_{min}$) correlates to the size of DW, i.e. $Latency_{min}$ = DW - 1.

Results from this investigation show that by increasing the size of DW from 1 to 2, a significant reduction in FPR can be obtained. However, choosing the appropriate size of the DW is crucial as it also likely to have a negative effect on the TPR and Latency.

No further investigation with other DWs were conducted because the aim of this investigation is to see whether the FPR is also reduced by varying the size of DW and results have demonstrated to be the case. In addition, as results in Figure 7.9 have shown, a further increase in DW size might not provide further significant benefits. Having said that, it is also dependent on whether how expensive is the recovery measure. A larger DW would tend to introduce greater latency in detection and the number of false positives that last for three consecutive control cycles or more is very small. In practise, the recovery measure might be expensive and thus it should only be carried out when necessary. Thus, using a larger DW would be preferable. It is noted these results are specific to our experiments

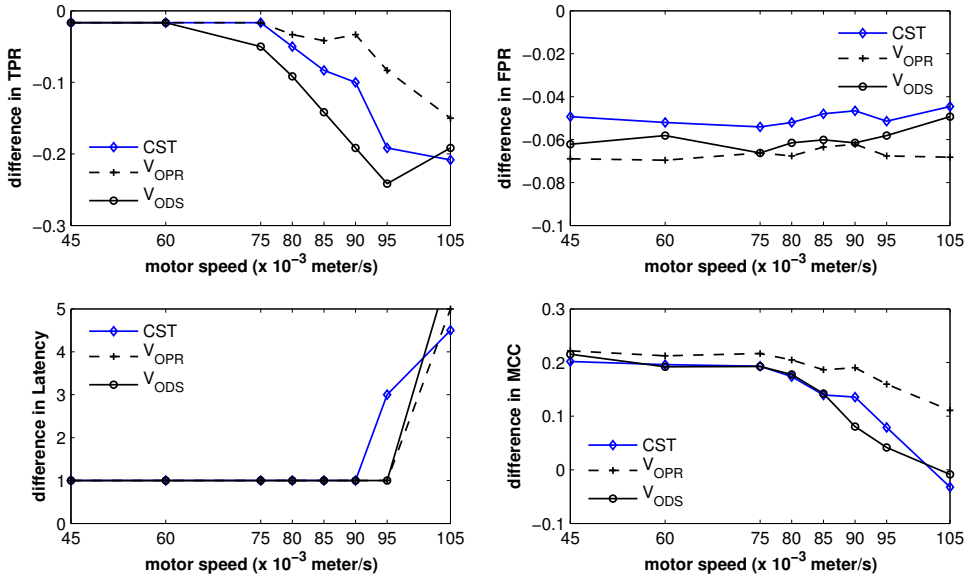


Figure 7.10: The difference in performance between the original RDA with the RDA-DW2 in detecting P_{PT} errors. A positive value denotes an increment, a negative value denotes a reduction, and zero means no difference.

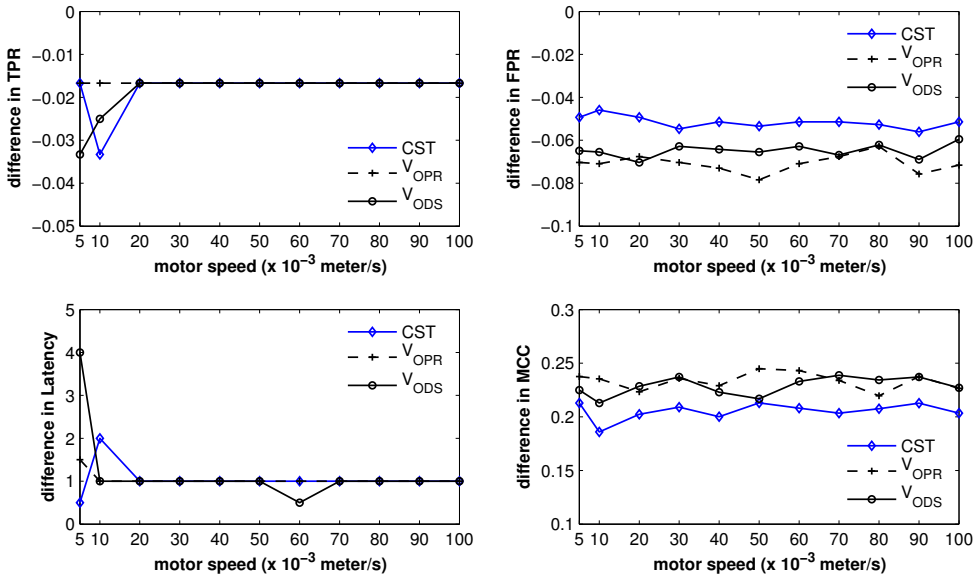


Figure 7.11: The difference in performance between the original RDA with the RDA-DW2 in detecting P_{GR} errors. A positive value denotes an increment, a negative value denotes a reduction, and zero means no difference.

and for other systems the findings may need to be re-visited.

7.5 Analysis of the RDA's Parameters

This section presents an analysis on the RDA's parameters by investigating the effect on the `TPR`, `FPR` and the `Latency` of detection as three parameters of the RDA classifier are varied. The objective of this experiment is to examine the effect of changing the value of the parameters and see the corresponding change of the performance metrics. Results from this experiment show on how the value of the parameters can be changed to achieve the desired performance. Referring to Eq. 3.8, there are three main parameters that affect the error-detection ability of the RDA classifier. These parameters are: receptor position decay rate b, negative feedback decay rate a, and current input stimulation rate gb. Note that the analysis of the RDA's parameters is presented here and not earlier because the results can reveal how sensitive are the parameters. If a small change to the parameters causes a big change in the performance, then a greater consideration is needed when choosing a value for each parameter.

For each parameter, the parameter value is varied and the results are compared qualitative and quantitatively. Only one parameter is varied at a time, and when an optimal value is selected, it is then used for subsequent analyses of other parameters. The complete list of the value ranges of the RDA parameters is presented in Table 7.7. For each parameter being varied, there is an initial value, an upper value, an increment value, and a default value. The default value is used for parameters that are not varied. For example, when analysing the parameter b, a default value of 1.1 for gb and 1.0 for a is used. Each value of the parameters was tested for $\texttt{P}_{\texttt{CP}}$, $\texttt{P}_{\texttt{PT}}$ = 45 $\times 10^{-3}$ m.s^{-1}, and $\texttt{P}_{\texttt{PT}}$ = 100 $\times 10^{-5}$ m.s^{-2} using the RDA-DW2 classifier.

Table 7.7: The value ranges for the main RDA parameters: receptor position decay rate b, negative feedback decay rate a, and current input stimulation rate gb.

Parameter	Initial	Increment	End	Default
b	0.02	0.02	0.22	0.1
gb	0.5	0.1	1.5	1.1
a	0.5	0.1	2.0	1.0

7.5.1 The b Parameter

The parameter b is the decay rate for the position of the receptors as seen in Eq. 3.8. This parameter controls the progression of the receptor's position towards ℓ to signal the detection of an error. This progression is based on the stimulation

generated from the input from other robots during initialisation phase.

The results on the effect of b parameter to the TPR, FPR, and the Latency as the value of b is varied are presented in graphs. The graphs are aligned in a single column, one after another for a better visual comparison and interpretation of results.

Results for the P_{CP} errors in Figure 7.12 show that as the value of the b parameter increases, the FPR also increases. This effect is also evident in the results for the P_{PT} errors in Figure 7.13, and the P_{GR} errors in Figure 7.14. However, the results show the increase in the FPR is very small and barely noticeable in the figures. This is mainly because the increment in value for the b parameter is only 0.02. Nevertheless, it is apparent from the results that a small value for the b parameter should used.

With respect to the TPR, as the value of the b parameter is increased, only in the case of the P_{CP} that there is an increase in the TPR. This occurs at b = 0.12. However, as with the FPR, the increase in the TPR is very small. For other models of fault of the wheels, the TPR remained constant throughout for all values of b.

On the Latency, the results for all modes of failure show that for all values of b tested, the Latency remained constant at 2.0. Thus, it appears that the variations in the b parameter have little effect on the latency of detection.

Based on the results in Figure 7.12, Figure 7.13, and Figure 7.14, it suggests that a small value of b should be used. Thus, b = 0.02 is chosen as the default value for the subsequent analyses on the gb and a parameters.

7.5.2 The gb Parameter

The gb parameter controls the stimulation rate for the current input namely the input from a current robot running the error detection mechanism.

The results of the effect of varying the parameter gb on the TPR, FPR, and the Latency for P_{CP} is presented in Figure 7.15. The top figure shows that the TPR is zero with gb less than 0.9 and dramatically increased to greater than 0.80 when the gb is greater than 0.9. For the CST and the V_{ODS} scenarios, a peak is reach when gb equals to 1.1 and it remained constant as gb is further increased. For the V_{OPR} scenario, the TPR maintains a steady increase as the gb increases from 0.9. On the FPR (middle figure in Figure 7.15), with gb less than 1.0 the FPR is zero. It then steadily increased as the gb increases. Finally, for the Latency, there is a distinct binary pattern where the Latency is 60 for gb less than 1.0, and the Latency is 2 for gb equal and greater than 1.0.

With the P_{PT} scenario, the results in Figure 7.16 show a similar effects on the

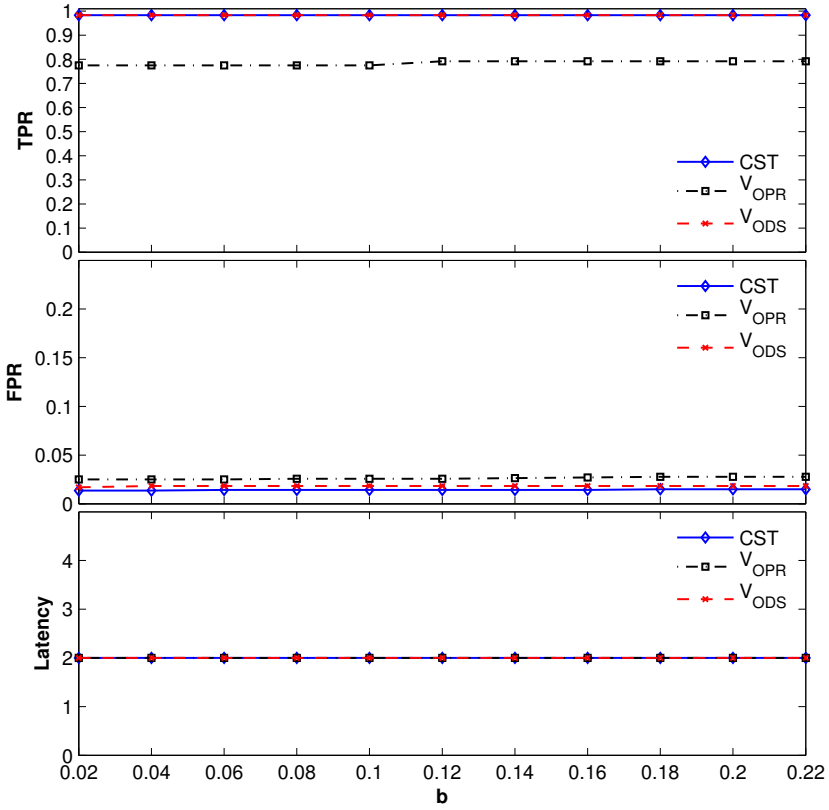


Figure 7.12: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different b in detecting the P_{CP} errors.

performance of detection as with the P_{CP}. A peak TPR is reached at gb equals to 1.1, and zero when gb equals to and less than 0.9. The FPR starts to increase with gb equals to 1.0 and continued steadily as the value of gb is increased. As for the Latency, a constant value of 2.0 is reached when gb is greater than or equal to 1.1.

A similar result with the P_{GR} in Figure 7.17 indicates that the gb equals to 1.1 is most appropriate value for current problem.

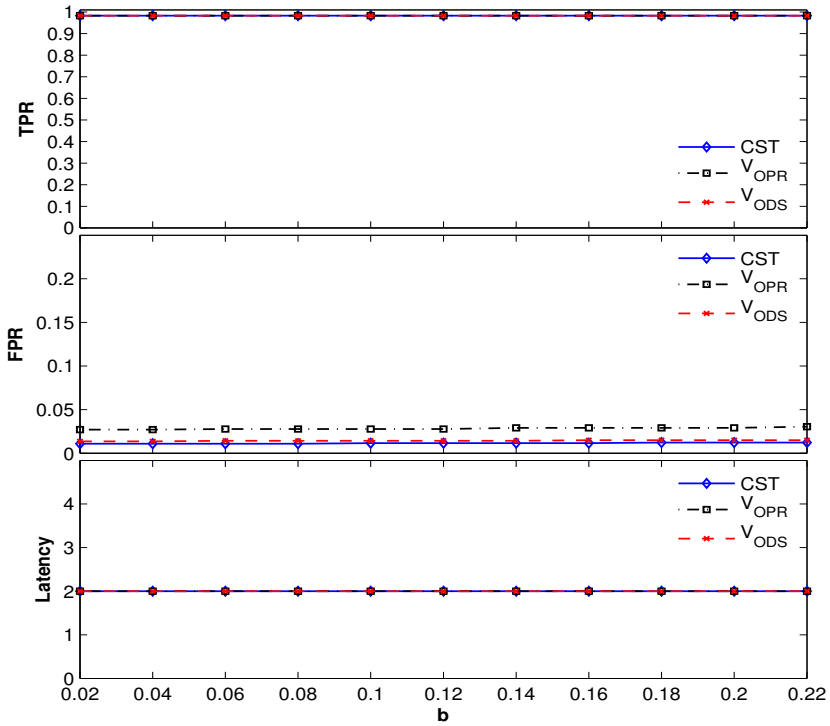


Figure 7.13: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different b in detecting the P_{PT} errors.

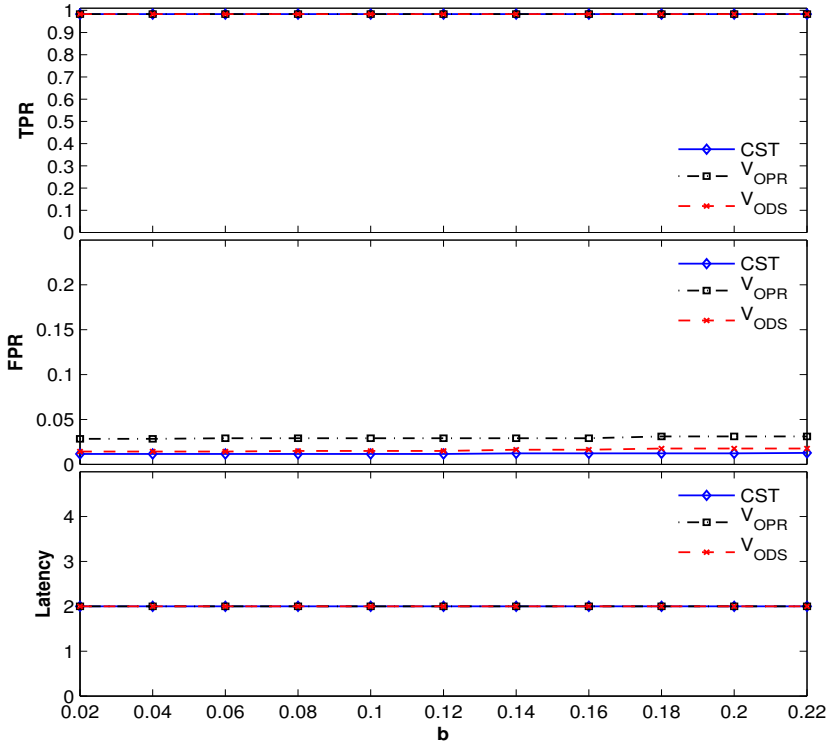


Figure 7.14: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different b in detecting the P_{GR} errors.

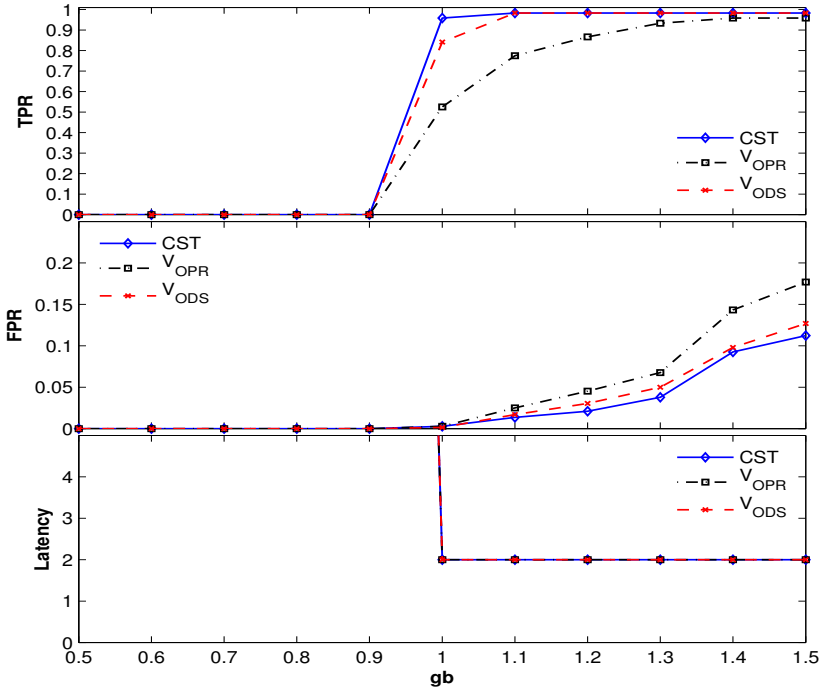


Figure 7.15: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different gb in detecting the P_{CP} errors

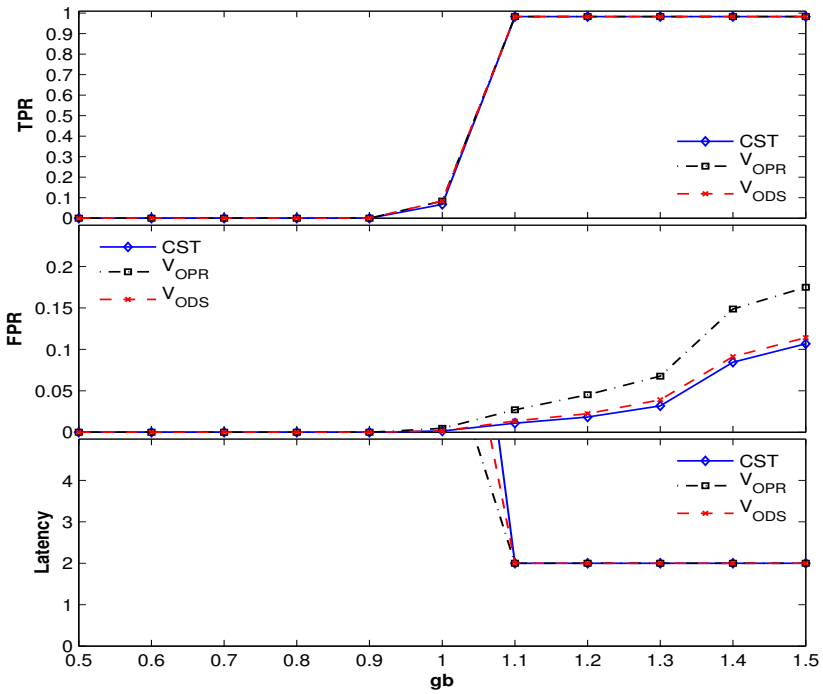


Figure 7.16: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different gb in detecting the P_{PT} errors.

7.5.3 The a Parameter

The a parameter controls the decay rate for the negative feedback generated from the input during the training phase in Eq. 3.9. From the previous two experi-

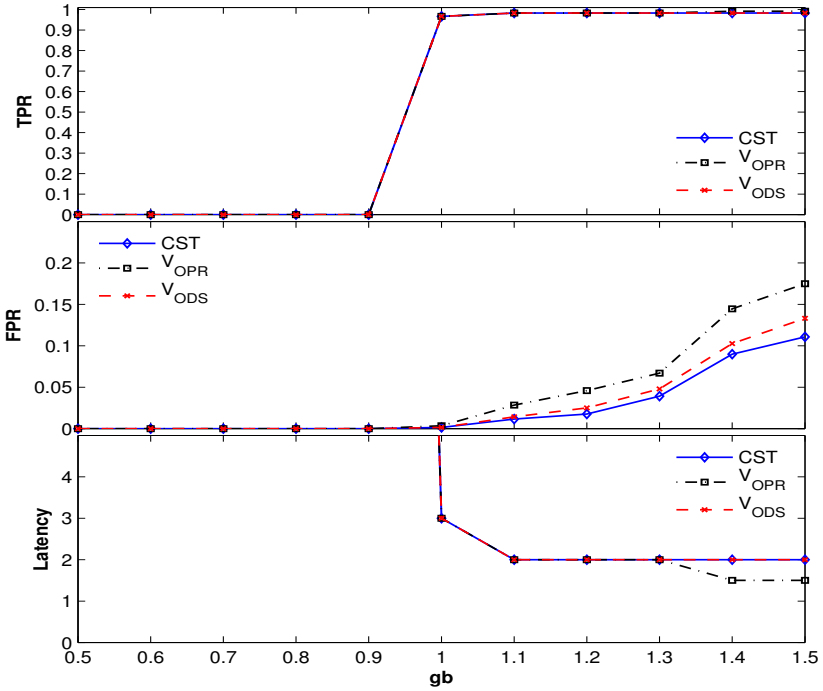


Figure 7.17: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different gb in detecting the P_{GR} errors.

ments, the chosen $b = 0.02$, and $gb = 1.1$ is used for the analyses of this parameter.

Figure 7.18 shows the results on the TPR, FPR, and the Latency for the RDA-DW2 classifier as a is varied with the P_{CP}. It can be seen that the TPR and the Latency remained constant as the parameter a is varied. Also, the same trend is seen in different scenarios. From these results, it appears that with the chosen values of b and gb, the TPR and the Latency is uncorrelated with a. This is also evident in the results with the P_{PT} errors in Figure 7.19 and with the P_{GR} errors in Figure 7.20. Therefore, the selection of a is solely based on the FPR.

Examining the results for the FPR with the P_{CP}, P_{PT}, and P_{GR} errors, it can be seen that as the value of a increases from 0.5 to 2.0 the FPR decreases. Interestingly, a minimum value of the FPR is reached with a equal to 1.7. The exact value of a when this occurs is the same with all faults. Therefore, it is suggested that the value of 1.7 is the most suitable for the parameter a for current problem.

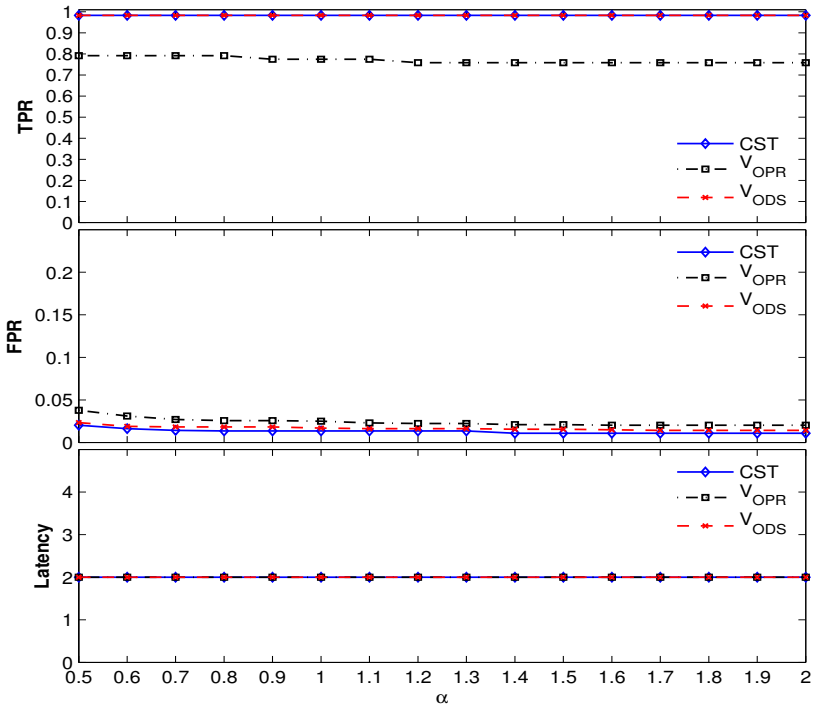


Figure 7.18: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different α in detecting the P_{CP} errors.

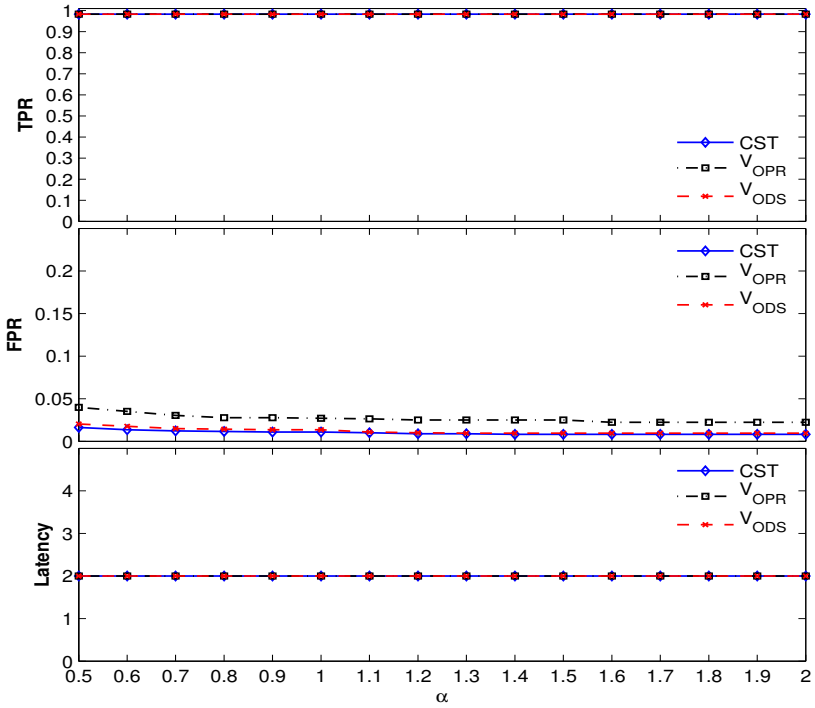


Figure 7.19: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different α in detecting the P_{PT} errors.

7.5.4 Summary

From the tuning exercise, a set of optimal parameter values for the current problem is b=0.02, gb=1.1 and a=1.7. It is noted that the order in which the parameters

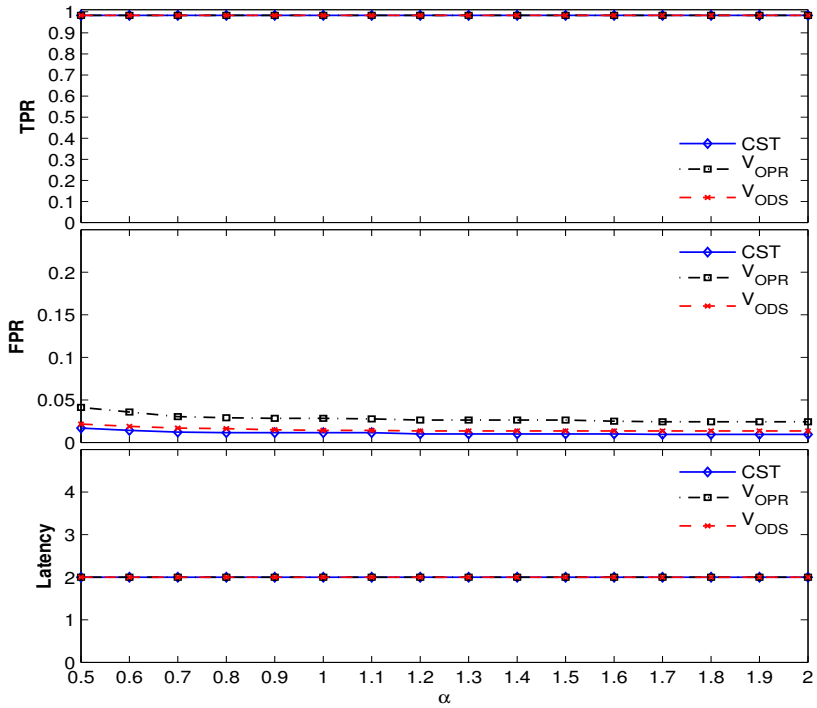


Figure 7.20: The TPR, FPR, and the Latency for the RDA-DW2 classifier with different α in detecting the P_{GR} errors.

are tested is relevant. If the parameter a is analysed first, it is likely that the suggested value for b and gb will be different. Also, each parameter affects the performance of detection differently as summarised in Table 7.8.

Table 7.8: Summary on the influence of the RDA's parameters on the performance of detection with current settings.

Parameter	TPR	FPR	Latency
b	yes	yes	little
gb	yes	yes	yes
α	little	yes	little

Recently, an independent analysis on optimising the RDA parameters using GA has been published in Hilder et al. [144] and thus will be referred. In Hilder et al. [144], the tuning was carried out for an anomaly detection on mass-spectrometry data.

7.6 Summary

In this chapter, an immune-inspired anomaly detection algorithm called the RDA was tested in the context of the CoDe scheme to detect the errors associated with

the wheels in the three operational environments. In Section 7.1, the RDA was implemented and the results showed that the RDA classifier is capable of adaptive error detection. When compared with the T-test and the Q-test classifiers, the RDA classifier produces a superior result with higher MCC scores. In general, the TPR of the RDA classifier is as high as the Q-test classifier whilst the FPR is as low as the T-test classifier.

However, the superior results of the RDA classifier are for the default variant of the P_{PT} and P_{GR}. For a more extensive investigation of the error-detection ability of all classifiers, a number of variants of the P_{PT} and P_{GR} were simulated. In Section 7.2, eight variants of the P_{PT} were tested and the results showed that the RDA classifier produced the highest MCC score in almost all scenarios. In Section 7.3, eleven variants of the P_{GR} were tested and the RDA classifier again produced the highest MCC score in nearly all scenarios. These results give evidence that the RDA classifier can not only be successfully implemented to provide an adaptive error detection for the simulated dynamic environments, it also produced a superior performance compared to other implemented statistical classifiers.

Having demonstrated the performance of the RDA classifier, Section 7.4 then proceeds to investigate the aspect of minimising the FPR as proposed in Christensen et al. [12]. The results from this section demonstrates that by simply increasing the size of the detection window from the default value of 1 to 2, the number of false positives can be significantly reduced.

Finally, Section 7.5 presents an analysis on the sensitivity of various parameters of the RDA classifiers. The results give insights on how these parameters can be optimised to provide a better performance for each of the performance metrics.

The CoDe scheme proposed in Chapter 6 requires that, for each control cycle, each robot communicates its data with other robots within its communication range. However, this constant wireless communication is energy expensive. Therefore, the next chapter analyses two strategies for the CoDe scheme that aims to reduce the communication overhead without significantly degrading the performance of detection.

CHAPTER

EIGHT

Strategies to Reduce Communication Overheads

The proposed CoDe scheme for adaptive error detection in Chapter 6 requires constant wireless communications among robots for data exchange. Since wireless communications is expensive in terms of energy, two strategies are investigated aimed at reducing the communication overheads. In section 8.1, the motivation for proposing the strategies is presented. Then the two proposed communication strategies are given in Section 8.2 and Section 8.3. The two strategies are analysed in Section 8.4 in terms of the number of communication overheads and the performance of detection.

8.1 Motivation

The proposed CoDe scheme for an adaptive error detection assumes that robots communicate data through a wireless medium either through broadcasting or direct one-to-one communications. Potential problems with the wireless communication medium such as contentions, localisations, interferences, or transceiver faults are out of the scope of this thesis. Thus, it is assumed that data exchanges are carried out without any problems, e.g, messages being lost or uncompleted. However, wireless communications consume a lot of resources [145]. Thus, it would be beneficial if the number of communications can be minimised. Here, two communication strategies are proposed in the context of CoDe. This does not apply if the exchange of data is through other mediums such as physical or

stigmergic (indirect) interactions.

At control cycle t in the original implementation, each robot exchanges its data with other robots within its communication range. Here, the data exchanges between robots is considered as 1 unit of communication. Thus, the number of communications is equal to the number of control cycles in a stimulation, i.e., number of communications = 80 per pair of robots. However, not all of these constant communications are necessary as the probability of a fault occurring continuously throughout the lifetime of a robot should be low. Based on this assumption, two strategies are proposed to minimise the communication overhead by emphasising the internal detection and only communicating with other robots for cross-validation when an error is detected internally.

Section 8.2 introduces the first strategy called the Optimistic Communication Strategy in which a robot only communicate with other robots if an error is detected internally using the data from only the individual robot. The second strategy called the Pessimistic Communication Strategy that actively carries out cross-validation based on both internal and external detection is presented in Section 8.3 .

8.2 Strategy 1 - Optimistic Communication

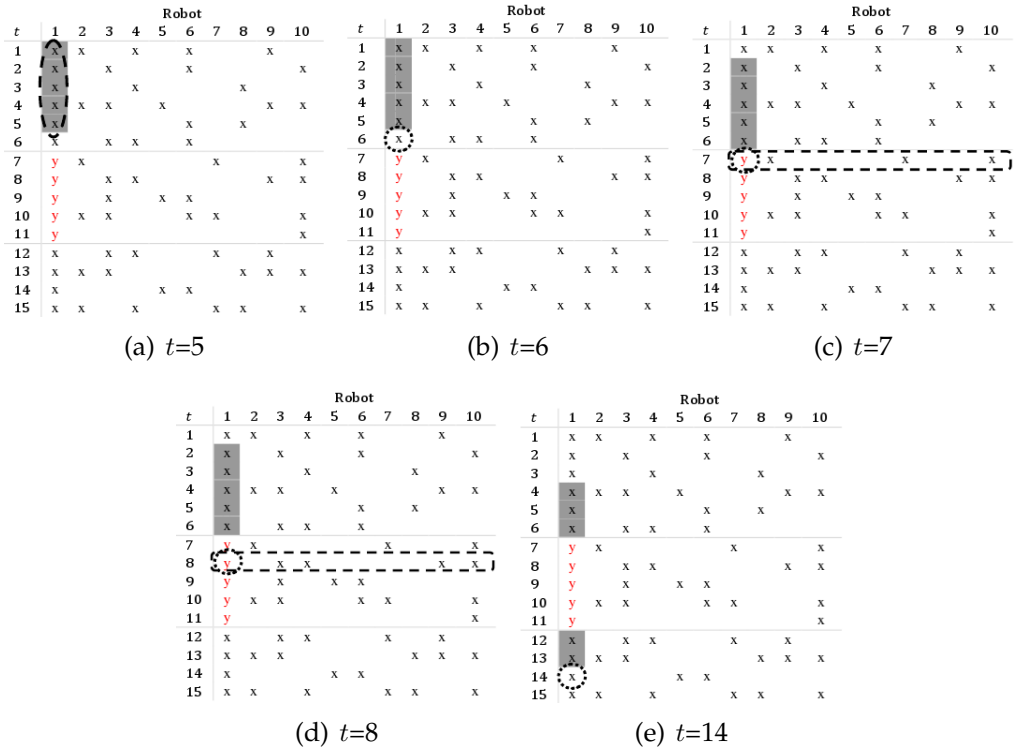
The first strategy is called the Optimistic Communication Strategy in which data exchanges are only carried out if an error is detected internally. For simplicity, it is referred to as `OpCom`. A distinction is made between internal and external errors and refers to the former as $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ for the later. For an internal detection, the classifier determines whether an error has occurred based on only the robot's own data; whereas for external detection the classifier uses data from all the robots within a robot's communication range.

This strategy is presented in Algorithm 2. It begins with the initialisation of an internal buffer $\mathcal{W}$ with a robot's own data from the first m control cycles (this is effectively using a time window as described in Section 6.2). The size of the buffer, $\mathcal{W}$, can vary; it is set to five in this research. After initialisation, at control cycle t, the classifier evaluates the current data instance v with those in $\mathcal{W}$ (internal detection). If the classifier classifies v as an error, $\texttt{err}_{\texttt{int}}$ is considered detected. Then, the classifier will perform a further evaluation using the data from other robots $\mathcal{DN}$ (external detection). If the classifier also classifies v as an error with respect to $\mathcal{DN}$, then $\texttt{err}_{\texttt{ext}}$ is considered detected. An error is only confirmed if it was detected from both the internal and external detection. If not, v replaces the oldest entry in $\mathcal{W}$.

Algorithm 2: Pseudocode for `OpCom`

```
Input: current data instance v, neighbourhood data DN, classifier A
Output: result of detection
initialise internal buffer W;
foreach control cycle t do
    err_int = A(v, W);
    if err_int then
        if CalculateNeighbour(DN) < 2 then
            err_ext = A(v, DN_temp);
        else
            err_ext = A(v, DN);
            DN_temp=DN;
        end
        if err_ext then
            report err_ext;
        else
            update(W);
        end
    else
        update(W);
    end
end
```

To illustrate how this strategy works, an example of error detection in robot R1 is shown in Figure 8.1. In Figure 8.1(a), initialisation of $\mathcal{W}$ is carried out with data instances from the first five control cycles. For clarity, data in $\mathcal{W}$ is shaded grey; dotted circle is the current data instance; dotted rectangle is the neighbourhood data; 'x' is normal; and 'y' is anomalous. After initialisation, at control cycle $t = 6$, internal detection is carried out. As no $\texttt{err}_{\texttt{int}}$ is detected, $\mathcal{W}$ is updated as seen in Figure 8.1(c) with shaded grey at $t = 6$. At $t = 7$, an $\texttt{err}_{\texttt{int}}$ is detected and robot R1 communicates with other robots to get the neighbourhood data $\mathcal{DN}$. As an $\texttt{err}_{\texttt{ext}}$ is detected, $\mathcal{W}$ was not updated. At $t = 8$, internal and external detection was carried out as shown in Figure 8.1(d), and a similar result is obtained as the one at $t = 7$. As shown in Figure 8.1(e), $\mathcal{W}$ is only updated with data not flagged as an error by both internal and external detection. In this example, the number of communications is only 5 whilst it is 15 with the original implementation.



(a) t=5 (b) t=6 (c) t=7

(d) t=8 (e) t=14

Figure 8.1: Strategy S1: Optimistic Communication Strategy (`OpCom`). A fault is introduced to robot R1 from control cycle t = 7 to t = 11. In the figures, a 'y' is anomalous and a 'x' is normal. (a) Initialisation. Fill internal buffer $\mathcal{W}$ of size 5 with data instances from the first five control cycles. Data in $\mathcal{W}$ is shaded grey. (b) At t = 6, evaluate whether a current data instance in circle is anomalous w.r.t data in $\mathcal{W}$. If no $\texttt{err}_{\texttt{int}}$ detected, update $\mathcal{W}$ with current data instance. (c) At t = 7, current data instance is classified as an error w.r.t data in $\mathcal{W}$, i.e., $\texttt{err}_{\texttt{int}}$ is detected, get data from other robots $\mathcal{DN}$. From the external detection, current data instance is also classified as an error w.r.t $\mathcal{DN}$, i.e., err_{ext} is detected, current data instance will not be added to $\mathcal{W}$. (d) At t = 8, both $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ are detected. Same as (c). (e) At t = 14, no $\texttt{err}_{\texttt{int}}$ detected; current data instance is added to $\mathcal{W}$.

8.3 Strategy 2 - Pessimistic Communication

The second strategy is called the Pessimistic Communication Strategy in which the external detection is activated based on the results from both internal and external detection. This strategy is similar to `OpCom` but with the addition of a parameter c to control the frequency of communication. This strategy is referred to as `PeCom`.

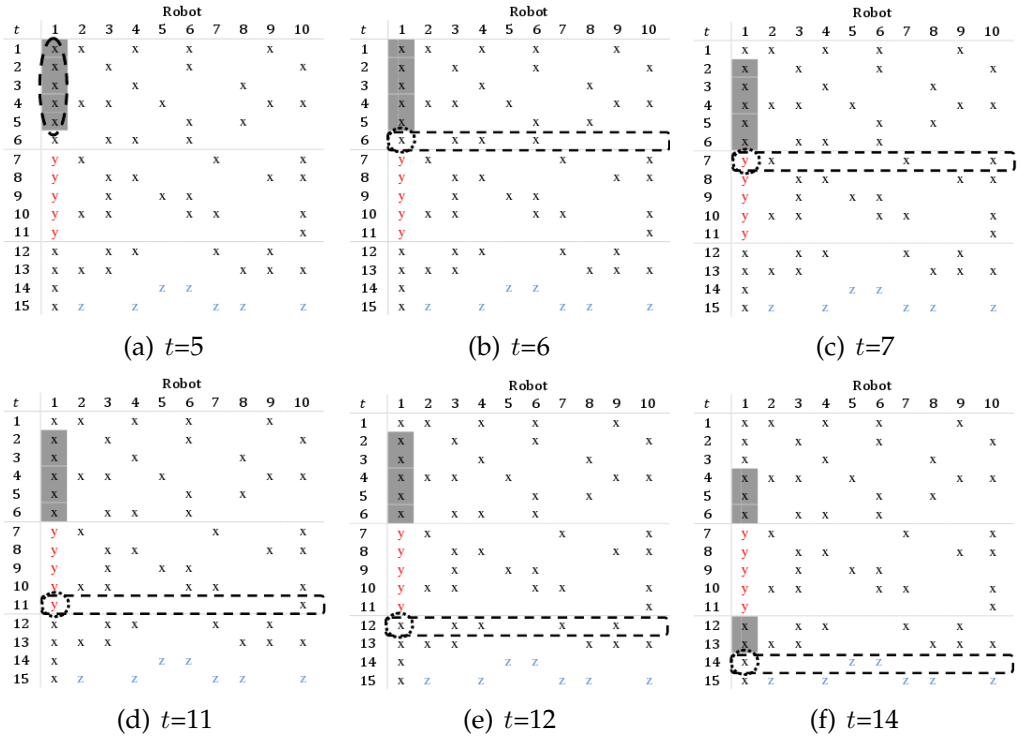
Algorithm 3 is the pseudocode for `PeCom`. Similar to `OpCom`, it starts with the initialisation of internal buffer $\mathcal{W}$. In addition, an external detection counter c is added and initially set to 1. This counter dictates the next external detection, which is after cth control cycle. For example, c = 1 means that the next external detection is one control cycle after the current cycle (which is the next control cycle), whereas c = 4 means that the next external detection is 4 control cycles after the current cycle. The maximum value for c is controlled by the parameter K. In principle, the parameter K can be ignored and thus if no $\texttt{err}_{\texttt{int}}$ is found, no external detection (communication) is activated. However, for practical reasons due to the uncertainty in physical world and imperfect components, it is included.

An internal detection is still carried out at every control cycle. If $\texttt{err}_{\texttt{int}}$ is detected, subsequent external detection with communicated data $\mathcal{DN}$ is conducted. If both $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ are detected, an error is reported and c is reset to 1. Even if there is no $\texttt{err}_{\texttt{int}}$, external detection is still carried out subject to c. At this stage, if there is no $\texttt{err}_{\texttt{ext}}$ the value of c is incremented by 1. The increment stops when c reaches K. Thus, the frequency of external detection changes according to results from both the internal and external detection.

An example of using `PeCom` to detect errors in robot R1 is shown in Figure 8.2. In the figure, data in $\mathcal{W}$ is shaded grey; dotted circle is the current data instance; dotted rectangle is the neighbourhood data; 'x' is normal from R1's perspective; 'y' is anomalous, and 'z' is new normal.

Algorithm 3: Pseudocode for `PeCom`

```
Input: current data instance v, neighbourhood data DN, classifier A,
       parameter K
Output: result of detection
initialise sliding window W;
counter c ← 1 , tc ← (size of W + 1);
foreach control cycle t do
    err_int = A(v, W);
    if err_int then
        if CalculateNeighbour(DN) < 2 then
            err_ext = A(v, DN_temp);
        else
            err_ext = A(v, DN);
            DN_temp=DN;
        end
        if err_ext then
            report err_ext;
            c ←1;
        else
            update(W);
        end
        tc = t + c;
    else
        if t == tc then
            err_ext = A(v, DN);
            if err_ext then
                report err_ext;
                c ←1;
                tc = t + c;
            else
                if c < K then
                    c++;
                end
            end
            tc = t + c;
        else
            update(W);
        end
    end
end
```



(a) t=5 (b) t=6 (c) t=7

(d) t=11 (e) t=12 (f) t=14

Figure 8.2: Strategy S2: Pessimistic Communication Strategy (PeCom). A fault is introduced to robot R1 from control cycle t = 7 to t = 11 and t = 14 to t = 15. In the figures, a 'y' is anomalous, a 'x' is normal from R1's perspective, and 'z' is new normal. (a) Initialisation. Fill $\mathcal{W}$ with data instances from the first five control cycles. Assign c = 1 and tc = 6. Data in $\mathcal{W}$ is shaded grey. (b) At t = 6, evaluate whether a current data instance in circle is anomalous w.r.t data in $\mathcal{W}$. No $\texttt{err}_{\texttt{int}}$ detected. Since $t = tc$, activate external detection, found no $\texttt{err}_{\texttt{ext}}$. The current data instance is added to $\mathcal{W}$ replacing the oldest entry in $\mathcal{W}$. (c) At t = 7, $\texttt{err}_{\texttt{int}}$ is detected. A further evaluation with $\mathcal{DN}$ classified the current data instance as an $\texttt{err}_{\texttt{ext}}$. Thus, the $\mathcal{W}$ is not updated, reset c=1 and tc = 8. (d) At t = 11, both $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ were detected; $\mathcal{W}$ not updated; reset c = 1 and tc = 12. (e) At t = 12, no $\texttt{err}_{\texttt{int}}$, $c = c + 1 = 2$. Update $\mathcal{W}$ with current data instance and tc = 14. (f) At t = 14, no $\texttt{err}_{\texttt{int}}$ but due to tc, external detection is conducted. An $\texttt{err}_{\texttt{ext}}$ was detected, reset c = 1 and tc = 15.

In Figure 8.2(a), $\mathcal{W}$ is initialised with data instances from the first five control cycles. At this stage, c is set to 1 and tc is set to 6. At t = 6 in Figure 8.2(b), an internal detection was carried out and no $\texttt{err}_{\texttt{int}}$ was found. At t = 7 in Figure 8.2(c) , $\texttt{err}_{\texttt{int}}$ was detected and thus an external detection is subsequently carried out. Here, $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ were both detected. Thus, error is reported and c is reset to 1 and tc was set to 8. From t = 8 to t = 11, both internal and external detection were carried out. Because both $\texttt{err}_{\texttt{int}}$ and $\texttt{err}_{\texttt{ext}}$ were detected, c = 1 throughout. At t = 12 in Figure 8.2(e), faults no longer persist and thus no $\texttt{err}_{\texttt{int}}$ was detected. External detection did not detect any $\texttt{err}_{\texttt{ext}}$ and thus c is

incremented by 1 and thus the next external detection is at t = 14.

If, for some reason, (e.g. either due to circuitry faults, interference or deliberate sabotage) the data on R1 remains unchanged. For example in Figure 8.2(f) data for R1 is still 'x' whilst the new normal value is now 'z'. The `OpCom` strategy will fail to detect this error. The failed detection can also happen for gradual errors that result in drift in normality, or if an error occurs at the same time as the food density goes up (if the two effects cancel each other out). However, `PeCom` should be able to detect this error. However, how fast can this type of error be detected is dependent on the value of c. If c is large, then the error will be detected very late. This is the reason why the parameter K was introduced to avoid the value of c become too large. In this example, the number of communications is 9, at t = 6, 7, 8, 9, 10, 11, 12, 14, and 15.

8.4 Experimental Results

By examining the two strategies proposed, it can be seen that the number of communications is now proportional to the duration of a fault (how many control cycles a fault lasts) and not the length of a simulation. This may significantly reduce the communication overhead and thus the power usage.

To examine the potential of the strategies `OpCom` and `PeCom`, the strategies were tested in the context of the RDA classifier in detecting $\mathtt{P}_{\mathtt{PT}}$ = 45 $\times 10^{-3}$ m.s^{-1} and $\mathtt{P}_{\mathtt{GR}}$ = 100 $\times 10^{-5}$ m.s^{-2} with data from Section 6.4.2. The size of $\mathcal{W}$ and K was set to 5.

The results for the number of communications with the two strategies are shown in Table 8.1 and Table 8.2. In the tables, Robot R1 is faulty whilst the rest are fault-free. In the original implementation of the RDA classifier in CoDe, the number of communications for R1 was 80 which is greater than the number of instances with a fault of the wheels (i.e. 60). However, for the current implementations with `OpCom` and `PeCom`, the number of communications is significantly reduced to 60 units for `OpCom` and about 65 units for `PeCom`. For fault-free robots, the number of communications is even lower, with less than 5 units with `OpCom` and less than 25 units with `PeCom`.

An obvious question is which strategy is better? The simple answer is it depends. `OpCom` has the advantage of having the least communication overheads, subjects to err_{int}. However, a mis-classification is likely to be accumulated and affects the overall `TPR` as demonstrated in Figure 8.3. In the figure, the `TPRs` over the 20 runs were plotted. It can be seen that at simulation number 9, 19, and 20, the `TPRs` are significantly different from other runs. This is resulted from the

Table 8.1: The number of communications by each robot using different communication strategies in detecting $\mathtt{P}_{\mathtt{PT}}$ = 45 $\times 10^{-3}$ m.s^{-1} with the RDA classifier.

Env.	Strategy	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
CST	Original	80	80	80	80	80	80	80	80	80	80
	OpCom	60	0.5	0	0	0	0	1	0	0	1
	PeCom	65	19	18.5	19	20	18.5	19	19.5	19.5	19
V_{OPR}	OpCom	60	3	3	3	2.5	3	2.5	3	2	2
	PeCom	63.5	25	21.5	24.5	23	23.5	24	23.5	23.5	23.5
V_{ODS}	OpCom	60	2	2.5	2	3	3	2.5	2	3	2
	PeCom	65.5	22	22.5	21	19	21.5	18	22	20	20

Table 8.2: The number of communications by each robot using different communication strategies in detecting $\mathtt{P}_{\mathtt{GR}}$ = 100 $\times 10^{-5}$ m.s^{-2} with the RDA classifier.

Env.	Strategy	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
CST	Original	80	80	80	80	80	80	80	80	80	80
	OpCom	60	0.5	0	0.5	1	0	1	0	1	0
	PeCom	63	20	18.5	19	20	19.5	19.5	19.5	20	19
V_{OPR}	OpCom	60	2	3	3	3	2.5	3	2.5	2	2
	PeCom	63	21.5	24	25	22.5	24	22	24	22	23.5
V_{ODS}	OpCom	60	2.5	2	2	2	2	1	2	2.5	2
	PeCom	63	21	20	19	20	21	21	21	20.5	18.5

false negatives in the detection. For these runs, a false negative when the fault was first injected causes the first errorneous instance to be included in the buffer $\mathcal{W}$. This has a cascading effect that subsequent errorneous instances are classified as normal instances.

On the other hand with PeCom, such false negatives were not accumulated as further cross-referencing was carried out, subject to the parameter c and K (refer Algorithm 3). Therefore, in some cases, there is a tradeoff between the communication overhead and the TPR. However, overall, both strategies do not result in reductions of the error-detection ability as shown in Table 8.3 and Table 8.4. Instead, the performance may even be improved as shown in the tables with increased MCC scores.

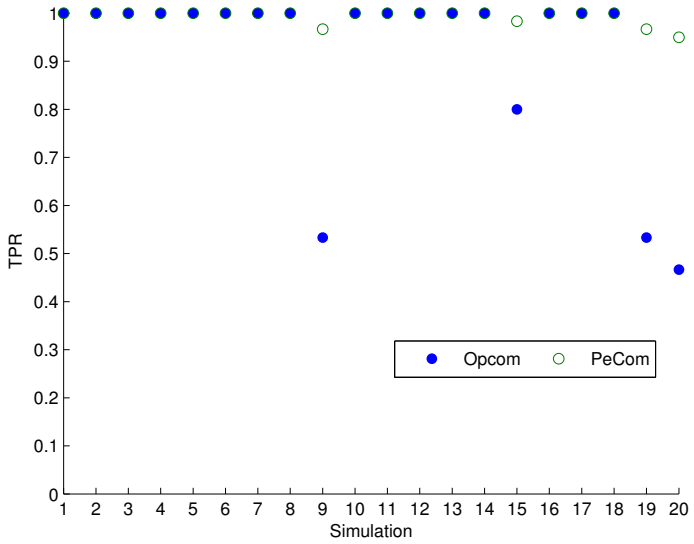


Figure 8.3: The TPR in detecting P_{PT} = 45 $\times 10^{-3}$ m.s^{-1} in a CST scenario for the 20 runs for the RDA classifier with with OpCom and PeCom communication strategy. At simulation 9, 15, 19, and 20, Opcom missed some errorneous instances but PeCom was still able to detect them due to additional external detections subject to c and K.

Table 8.3: Comparison on the median TPR, FPR, Latency, and MCC in detecting P_{PT} = 45 $\times 10^{-3}$ m.s^{-1} for the RDA classifier without and with the proposed communication strategies.

Env.	Strategy	TPR	FPR	Latency	MCC
CST	Original	1.00	0.061	1.0	0.75
	OpCom	1.00	0.00	1.0	0.97
	PeCom	1.00	0.04	1.0	0.82
V_{OPR}	Original	1.00	0.09	1.0	0.64
	OpCom	1.00	0.02	1.0	0.89
	PeCom	1.00	0.07	1.0	0.72
V_{ODS}	Original	1.00	0.07	1.0	0.72
	OpCom	1.00	0.01	1.0	0.92
	PeCom	1.00	0.03	1.0	0.85

8.5 Summary

In this chapter, two communication strategies were proposed in the context of CoDe that were aimed at reducing the communication overhead involved in an adaptive error detection in dynamic environments. In Section 8.1, the motivation looks into ways for reducing the communication overhead is presented. The motivation was that robots in a SRS are potentially resource limited and the proposed CoDe scheme for an adaptive error detection requires data to be communicated

Table 8.4: Comparison on the median TPR, FPR, Latency, and MCC in detecting P_{GR} = 100 $\times 10^{-5}$ m.s^{-2} for the RDA classifier without and with the proposed communication strategies.

Env.	Strategy	TPR	FPR	Latency	MCC
CST	Original	1.00	0.06	1.0	0.74
	OpCom	1.00	0.00	1.0	0.97
	PeCom	1.00	0.04	1.0	0.81
V_{OPR}	Original	1.00	0.09	1.0	0.66
	OpCom	1.00	0.02	1.0	0.90
	PeCom	1.00	0.07	1.0	0.73
V_{ODS}	Original	1.00	0.07	1.0	0.70
	OpCom	1.00	0.01	1.0	0.92
	PeCom	1.00	0.04	1.0	0.81

through a wireless medium at each control cycle t. However, the probability of a fault occurring continuously to a robot is low. Therefore, there is no need for the robots to constantly communicate data for error detection.

Therefore, two communication strategies were proposed in which a robot only communicates with other robots if an error has been detected internally. The first strategy is called OpCom and it is presented in Section 8.2 whilst the second strategy called PeCom is presented in Section 8.3. The main difference between OpCom and PeCom is that on top of the communications with other robots whenever an error is detected internally, PeCom also periodically carry out extra communications subject to a control variable c.

In Section 8.4, the two strategies were tested for the RDA classifier is detecting P_{PT} of 45 $\times 10^{-3}$ m.s^{-1} and P_{GR} of 100 $\times 10^{-5}$ m.s^{-2}. Results from the experiments show that the proposed strategies can significantly reduce the number of communications involved without a significant degradation on the performance of detection.

CHAPTER

NINE

Conclusion

This final chapter concludes the thesis, summarises its contribution, and makes suggestions for potential future work. In Section 9.1, the contributions are presented. In Section 9.2, a reflection on the limitations of the current work is offered and the direction of future work is recommended. Finally, concluding remarks on the work presented in this thesis are given in Section 9.3.

9.1 Contributions

This section summarises the contributions in this thesis.

Proposal of the Collective Self-Detection (CoDe) scheme

This work proposed a collective self-detection scheme called the CoDe scheme to address the problem of adaptive error detection in swarm robotics. From the literature review, it was identified that most studies on error detection in swarm robotics approach the problem from the perspective of a single robot. The problem with a single-robot's perspective to error detection is that the detection mechanism might not be able to tolerate changes in the dynamic environment, that is changes that also affect the detection metrics. To address this problem, the CoDe scheme was proposed. This scheme detects errors through cross-referencing a robot's behaviour with other robots within a logical neighbourhood. This is assuming that changes in the environment affect every robots within that logical neighbourhood. Through this cross-referencing mechanism, the *noise* from the environment can be filtered and errors can be effectively detected.

The CoDe scheme was investigated with the use of four statistical classifiers. Each classifier was tested on three fault models of the wheels in three different operational environments. In addition, each classifier was tested with a variety of detection thresholds. The results from the experiments show that the proposed CoDe scheme is effective at detecting all simulated faults. These results give evidence to support that data-driven error detection from the perspective of a collective can provide adaptive detection in dynamic environments.

Implementation of the Receptor Density Algorithm (RDA) Classifier Error Detection

In addition to the use of four statistical classifiers, this work also implemented an RDA classifier in the context of the CoDe scheme. Results from all the experiments (three fault models to the wheels in three operational environments) showed that the RDA classifier can also achieve adaptive error detection. In fact, the RDA classifier also significantly outperformed the other four statistical classifiers. These results further support the finding that adaptive error detection can be achieved with a collective-based approach.

With the RDA classifier, a further investigation was carried out on the possibility of reducing the number of false positives by increasing the detection window. Results showed that by increasing the detection window from 1 to 2, the number of false positives can be significantly reduced whilst at the same time maintaining the true positive rate.

To provide a better understanding on the effect of three RDA parameters (b and a, gb) on the performance of detection, a set of experiments was conducted by varying the values of those parameters. Results showed that the b and a parameters have more observable effects on the true positive rate (`TPR`) and false positive rate (`FPR`) compared with the `Latency` of detection. The gb parameter had a significant effect on the `TPR`, the `FPR` as well as the `Latency` of detection.

Strategies to Reduce Communication Overheads

There are potential limitations to the proposed CoDe scheme, in particular on the communication overhead involved. To address the limitations, two communication strategies: `OpCom` and `PeCom`, were proposed. Both strategies were implemented in the context of the RDA classifier and analysed with respect to the reduction of the number of communications involved as well as the effect on the performance of error detection. Results from the experiments showed that the two strategies produced a significantly lower communication overhead whilst still maintaining the level of performance in error detection. The `OpCom` strat-

egy produced a low communication overhead. However, any false positives will have an accumulative effect on the overall performance. On the other hand, the `PeCom` strategy produced a higher communication overhead than `OpCom` but it was unaffected by the false positives in previous control cycles.

9.2 Limitations and Future Work

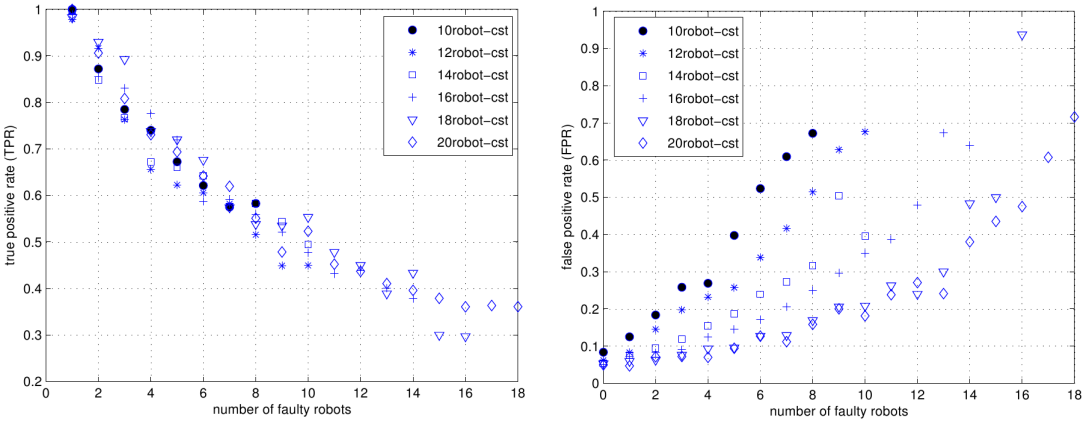
Implementation on Physical Robots

One of the main limitations of the work in this thesis is that the proposed scheme and strategies have not been implemented and tested on physical robots. Sometimes a solution which works in simulation might not work as effectively when tested on physical robots, e.g. *The Reality Gap* [76]. However, the proposed solution in this thesis should work in physical robot swarms (assuming they can communicate if in proximity) because the detection is from the perspective of a collective. Therefore, any interference from the environment should affects all robots instead of only a few individuals. Having said that, the immediate requirement for future work will be to implement the proposed solution in physical swarm robotic systems.

Faults on Multiple Robots

The work in this thesis has been demonstrated on faults of the wheels of a single robot. However, this is a limited scenario because faults can occur in multiple robots simultaneously. It would therefore be an interesting research study to investigate whether the proposed CoDe scheme with the RDA classifier will still be able to provide an adaptive error detection when there are multiple faulty robots in the swarm. In principle, the proposed scheme should still work because the probability of a robot encountering more faulty robots than fault-free robots in a collective should be lower when compared with the opposite scenario. An interesting question is that if the proposed scheme does work, at what stage will it fail. Intuitively, the size of the swarm is very important because the scheme should work if there are significantly more healthy robots compared with faulty robots in the swarm. However, at what proportion is the tipping point? Answers to these questions could give insights into how many robots to deploy to ensure that the swarm can still operate reliably.

Another important variable that might affect the detection of errors on multiple robots is the size of the logical neighbourhood. As demonstrated by the results from a preliminary study in Figure 9.1, with a fixed logical neighbourhood the increase of swarm size does not necessary result in an increase in the



(a) TPR (b) FPR

Figure 9.1: The TPR and FPR in detecting errors on multiple robots for different swarm sizes.

TPR (Figure 9.1(a)). Therefore, it would be interesting to find out what is the correlations between the number of faulty robots, the swarm size, and the size of the logical neighbourhood.

Multi-Resolution Detection

As pointed out in Section 5.2, the detection of errors in this work was carried out at a fixed control cycle. A potential drawback with a fixed control cycle is that the minimum time which elapses before an error can be detected is fixed. For time-critical applications, the goal is to be able to detect the errors as soon as possible. Therefore, having a multi-resolution detection is an attractive option. Multi-resolution detection may be implemented by having control cycles of varying lengths. In principle, a shorter control cycle can pick up errors with a large magnitude whilst a longer control cycle can pick up the more subtle errors. Having said that, having a multi-resolution detection requires additional storage and computation. Therefore, a viable solution is to have a control mechanism that can tune the resolutions on-line.

Heterogeneous Swarms

This proposed CoDe scheme assumes that the swarm is (functionally) homogeneous as indicated by **E3** (that the robots are functionally homogeneous and act relatively independently from each other and no cooperation is involved to carry out the task). For other multi-robot systems, the swarm may consist of heterogeneous robots. It would be an interesting work to investigate how the proposed scheme can be extended for heterogeneous robots.

Dynamic Environments

This study only investigated two scenarios of dynamic environment. For the scenario with biased distribution, the V_{ODS}, only four regions were considered. More distributions can be considered, e.g. where objects are only placed in $\frac{1}{8}$, $\frac{1}{4}$ or $\frac{1}{2}$ segment of the arena. Therefore, for future work, other dynamic environments (those that also affect the variables used in the evaluation metrics) can be investigated.

Fault Models

The fault models in this study only consider some magnitudes of the faults. For the partial failure (P_{PT}), the variants tested in Section 7.2 only range from 45 $\times 10^{-3}$ m.s^{-1} to 105 $\times 10^{-3}$ m.s^{-1}. Other variants could also be included such as 0 $\times 10^{-3}$ m.s^{-1}, 15 $\times 10^{-3}$ m.s^{-1}, and 30 $\times 10^{-3}$ m.s^{-1}. The reason why these variants were not included previously is because it was assumed that a P_{PT} is analogous to complete failure (P_{CP}) in which no objects will be collected by the faulty robot as it is not moving. For P_{PT} of 15 $\times 10^{-3}$ m.s^{-1} and 30 $\times 10^{-3}$ m.s^{-1}, it was assumed that if the proposed scheme can detect errors at 45 $\times 10^{-3}$ m.s^{-1}, it should also detect P_{PT} = 15 $\times 10^{-3}$ m.s^{-1} and P_{PT} = 30 $\times 10^{-3}$ m.s^{-1}, which are more severe. For the gradual failure (P_{GR}), the variants tested in Section 7.3 only ranges from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}. These variants of the P_{GR} last from one control cycle to twelve control cycles. For a more severe P_{GR}, other variants could be tested. For example, for a P_{GR} that lasts until the end of simulation, a P_{GR} = 1 $\times 10^{-5}$ m.s^{-2} can be tested.

Requirement of Fault-free Robots

The self-detection approach in the CoDe scheme requires that there are some fault-free robots in the swarm. If this is not the case, the errors cannot be detected. However, situations might occur in which all robots can be faulty at the same time. An example of such a situation would be the occurence of faults to robots in particular regions in a large environment. For such situations, it might be difficult for any detection and recovery to be initiated. One potential approach might be to have a periodic self-detection or diagnostic routine. The challenge then would be how to derive the appropriate time interval to activate the routine without much draining of resources.

Application to Other Domains

The proposed CoDe scheme with the RDA classifier has been demonstrated to work on a simulated homogenous swarm robotic system. The proposed solu-

tion can be easily adapted to other areas which show similar characteristics and problems such as wireless sensor networks (WSN). With WSN being particularly resource-constrained, the proposed strategies to reduce the communication overhead could be particularly useful.

9.3 Research Question Revisited

Chapter 1 defined the main research question of this thesis which was:

> *Can data-driven error detection from the perspective of a collective be used to provide adaptive detection for robot swarms deployed in time-varying environments?*

To investigate this research questions, a list of research objectives was identified. Having summarised all the chapters of this thesis, this section revisits the initial objectives of this research and draws some concluding remarks. Here are the objectives of this thesis:

- **RO1:** *To establish an experimental testbed to investigate the nature of errors and the aspect of adaptive error detection in swarm robotics.*

  Chapter 4 presented the experimental testbed to investigate the aspect of adaptive error detection in swarm robotics. The chosen platform was a foraging robot swarm consisting of ten robots which are subject to faults on the wheels and also to external influences from the operational environment. To establish the assumption that faults of the wheels and from external influence from the environment would be observable in the data, Chapter 5 presented the corresponding experiments and the results gave evidence to support the assumption.

- **RO2:** *To investigate the feasibility of addressing error detection in swarm robotics from the perspective of a collective, and to propose a corresponding detection scheme to incorporate the aspect of adaptivity to dynamic environments.*

  Chapter 6 presented the inability of achieving an adaptive error detection from the perspective of a single robot through illustrative examples. Then, a collective self-detection scheme called the CoDe scheme was proposed. The assumptions and potential limitations associated with the proposed CoDe scheme were explicitly specified. Illustrative examples were given on the application of the CoDe scheme and it was demonstrated to be able

to achieve adaptive error detection.

- **RO3:** *To examine the application of statistical classifiers within the proposed scheme for adaptive error detection.*

  The second part of Chapter 6 presented the application of four statistical classifiers in the context of the CoDe scheme to detect various errors in dynamic environments. Results from the experiments showed that the classifiers were able to provide an adaptive error detection.

- **RO4:** *To examine the application of an immune-inspired algorithm within the proposed scheme for adaptive error detection.*

  Chapter 7 presented the RDA algorithm and implemented the classifier in the context of the CoDe scheme. Results from the experiments showed that the RDA can be effectively implemented for adaptive error detection in dynamic environments.

Revisiting the research question in this thesis, on the basis of the results and the findings from the work conducted, it can be concluded that for the specific case study investigated in this thesis, data-driven error detection from the perspective of a collective can be used to provide an adaptive detection for robot swarms deployed in time-varying environments.

Bibliography

[1] Alan F.T. Winfield and Julien Nembrini. Safety in numbers: fault tolerance in robot swarms. *International Journal on Modelling, Identification and Control*, 1(1):30–37, 2006.

[2] Lynne E. Parker. Reliability and Fault Tolerance in Collective Robot Systems. In Serge Kernbach, editor, *Handbook on Collective Robotics: Fundamentals and Challenges*, page To appear. Pan Stanford Publishing, 2012.

[3] Erol Şahin. Swarm Robotics: From Sources of Inspiration to Domains of Application. In Erol Şahin and William M. Spears, editors, *Proceedings of the 1st International Workshop on Swarm Robotics*, LNCS 3342, pages 10–20. Springer, 2005.

[4] Erol Şahin, Sertan Girgin, Levent Bayindir, and Ali Emre Turgut. Swarm robotics. In Christian Blum and Daniel Merkle, editors, *Swarm Intelligence: Introduction and Applications*, pages 87–100. Springer, 2008.

[5] Levent Bayindir and Erol Şahin. A Review of Studies in Swarm Robotics. *Turkish Journal on Electrical Engineering and Computer Sciences*, 15(2):115–147, 2007.

[6] Marco Dorigo and Enrol Şahin. Guest Editorial. *Autonomous Robots: Swarm Robotics Special Issue*, 17:111–113, 2004.

[7] Roderich Groß, Francesco Mondada, and Marco Dorigo. Transport of an object by six pre-attached robots interacting via physical links. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 1317 –1323, 2006.

[8] Algirdas Avižienis. Design of fault-tolerant computers. In *Proceedings of the 1967 Fall Joint Computer Conference*, pages 733–743, 1967.

[9] Anders Lyhne Christensen. *Fault Detection in Autonomous Robots*. PhD thesis, Université Libre de Bruxeles, Brussels, Belgium, 2008.

[10] Richard Canham, Alexander H. Jackson, and Andy Tyrrell. Robot Error Detection Using an Artificial Immune System. In *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 199–207. IEEE Computer Society, 2003.

[11] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, and Marco Dorigo. Automatic Synthesis of Fault Detection Modules for Mobile Robots. In *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems*, pages 693–700. IEEE Computer Society, 2007.

[12] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, and Marco Dorigo. Exogenous Fault Detection in a Collective Robotic Task. In *Proceedings of the 9th European Conference on Artificial Life*, LNAI 4648, pages 555–564. Springer, 2007.

[13] Maizura Mokhtar, Ran Bi, Jon Timmis, and Andy M. Tyrrell. A Modified Dendritic Cell Algorithm for On-line Error Detection in Robotic Systems. In *Proceedings of the Congress on Evolutionary Computation*, pages 2055–2062. IEEE Press, 2009.

[14] Stephan Olariu and Albert Y. Zomaya, editors. *Handbook of Bioinspired Algorithms and Applications*. Chapman & Hall/CRC, 2006.

[15] Leandro N. de Castro and Jonathan Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.

[16] Nick D.L. Owens, Andy Greensted, Jon Timmis, and Andy Tyrrell. T Cell Receptor Signalling Inspired Kernel Density Estimation and Anomaly Detection. In Paul S. Andrews, Jon Timmis, Nick D.L. Owens, Uwe Aickelin, Emma Hart, and Andy M. Tyrrell, editors, *Proceedings of the 8th International Conference on Artificial Immune Systems*, pages 122–135. Springer, 2009.

[17] Gerardo Beni and Jim Wang. Swarm intelligence in cellular robotic systems. In *Proceedings of NATO Advanced Workshop on Robots and Biological Systems*, pages 26–30, 1989.

[18] Gerardo Beni. From Swarm Intelligence to Swarm Robotics. In Erol Şahin and William M. Spears, editors, *LNCS 3342*, pages 1–9. Springer, 2005.

[19] Marco Dorigo and Mauro Birattari. Swarm intelligence. *Scholarpedia*, 2(9): 1462, 2007.

[20] Jon Timmis, Paul Andrews, and Emma Hart. On Artificial Immune Systems and Swarm Intelligence. *Swarm Intelligence*, 4:247–273, 2010.

[21] I. D Couzin and N. R Franks. Self-organized lane formation and optimized traffic flow in army ants. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1511):139–146, 2003.

[22] P. P Grassé. La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes Natalensis et Cubitermes sp. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[23] Eric Bonabeau, Guy Theraulaz, Jean-Louis Deneubourg, Serge Aron, and Scott Camazine. Self-Organization in Social Insects. *Trends in Ecology and Evolution*, 12(5):188–193, 1997.

[24] Frank H. Heppner. Avian Flight Formations. *Bird-Banding*, 45(2):160–169, 1974.

[25] Wayne K. Potts. The chorus-line hypothesis of manoeuvre coordination in avian flocks. *Nature*, 309:344–345, 1984.

[26] Brain L. Partridge. The Structure and Function of Fish Schools. *Scientific American*, 246(6):90–99, 1982.

[27] Daniel Grünbaum, Steven Viscido, and Julia K. Parrish. Extracting interactive control algorithms from group dynamics of schooling fish. In A.S. Morse Vijar Kumar, Naomi Leonard, editor, *Proceedings of the Block Island Workshop on Cooperative Control*, LNCIS 309, pages 447–450. springer, 2004.

[28] Jacques Gautrais, Pablo Michelena, Angela Sibbald, Richard Bon, and Jean-Louis Deneubourg. Allelomimetic synchronization in Merino sheep. *Animal Behaviour*, 74(5):1443–1454, 2007.

[29] Eshel Ben-Jacob, Ofer Schochet, Adam Tenenbaum, Inon Cohen, Andras Czlròk, and Tamas Vicsek. Generic modelling of cooperative growth patterns in bacterial colonies. *Nature*, 368(6466):46–49, 1994.

[30] Eshel Ben-Jacob, Inon Cohen, and David L. Gutnick. Cooperative Organization of Bacterial Colonies: From Genotype to Morphotype. *Annual Review of Microbiology*, 52:779806, 1998.

[31] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

[32] Guy Theraulaz, Eric Bonabeau, and Jean-Louis Deneubourg. The Origin of Nest Complexity in Social Insects. *Complexity*, 3(6):15–25, 1998.

[33] J. L. Deneubourg, J. M. Pasteels, and J. C. Verhaeghe. Probabilistic Behaviour in Ants: A Strategy of Errors? *Journal of Theoretical Biology*, 105: 259–271, 1983.

[34] J. L. Deneubourg and S. Goss. Collective patterns and decision-making. *Ethology, Ecology & Evolution*, 1:295–311, 1989.

[35] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[36] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[37] James Kennedy and Russell Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[38] Majid M. Khodier and Christos G. Christodoulou. Linear Array Geometry Synthesis With Minimum Sidelobe Level and Null Control Using Particle Swarm Optimization. *IEEE Transactions on Antennas and Propagation*, 53: 26742679, 2005.

[39] S. Kannan, S. Mary Raja Slochanal, and Narayana Prasad Padhy. Application and Comparison of Metaheuristic Techniques to Generation Expansion Planning Problem. *IEEE Transactions on Power Systems*, 20:466–475, 2005.

[40] Craig W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4):25–34, 1987.

[41] C. Ronald Kube and Hong Zhang. Collective Robotics: From Social Insects to Robots. *Adaptive Behavior*, 2:189–218, 1993.

[42] Wenguo Liu. *Design and Modelling of Adaptive Foraging in Swarm Robotic Systems*. PhD thesis, University of the West of England, 2008.

[43] Tim Ziemke. On the role of Robot Simulations in Embodied Computer Science. *AISB Journal*, 1(4):389–399, 2003.

[44] Vito Trianni, Roderich Groß, Thomas Halva Labella, Erol Şahin, and Marco Dorigo. Evolving aggregation behaviors in a swarm of robots. In *ECAL*, Lecture Notes in Artificial Intelligence, pages 865–874. Springer Verlag, 2003.

[45] A Martinoli, A. J. Ijspeert, and L. M. Gmabardella. A probabilistic model of understanding and comparing collective aggregation mechanisms. In *Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 575–584. Springer Verlag, 1999.

[46] A. Martinoli, A. J. Ijspeert, and F. Mondada. Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 19:51–63, 1999.

[47] Alcherio Martinoli, Kjerstin Easton, and William Agassounon. Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *International Journal of Robotics Research*, 23:415–436, 2004.

[48] A. J. Ijspeert, A. Martinoli, A. Billard, and L. M. Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics: the stick pulling experiment. *Autonomous Robots*, 11(2):149–171, 2001.

[49] Rodney A. Brooks. A Robust Layered Control System For A Mobile Robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986.

[50] Roderich Groß and Marco Dorigo. Evolving a Cooperative Transport Behavior for Two Simple Robots. In *6th International Conference on Evolution Artificielle*, LNCS 2936, pages 305–316, 2004.

[51] Jakob Fredslund and Maja J Matarić. Robots in Formation Using Local Information. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems*, pages 100–107, 2002.

[52] Onur Soysal, Erkin Bahçeci, and Erol Şahin. Aggregation in Swarm Robotic Systems: Evolution and Probabilistic Control. *Turkish Journal of Electrical Engineering*, 15(2):199–225, 2007.

[53] Yasuke Hanada, Geunho Lee, and Nak Young Chong. Adaptive Flocking of a Swarm of Robots Based on Local Interactions. In *IEEE Swarm Intelligence Symposium*, pages 340 –347, 2007. doi: 10.1109/SIS.2007.367957.

[54] Julien Nembrini, Alan Winfield, and Chris Melhuish. Minimalist Coherent Swarming of Wireless Connected Autonomous Mobile Robots. In *Proceedings of the 7th International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 373–382. MIT Press, 2002.

[55] Jan Dyre Bjerknes, Alan FT Winfield, and Chris Melhuish. An Analysis of Emergent Taxis in a Wireless Connected Swarm of Mobile Robots. In *IEEE Swarm Intelligence Symposium*, pages 45–52, 2007. doi: 10.1109/SIS.2007.368025.

[56] Maja J Matarić, Martin Nilsson, and Kristian T. Simsarian. Cooperative Multi-Robot Box-Pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 556–561. IEEE, 1995. doi: 10.1109/IROS.1995.525940.

[57] C. Ronald Kube and Hong Zhang. Tasks Modelling in Collective Robotics. *Autonomous Robots*, 4:53–72, 1997.

[58] C. Ronald Kube and Eric Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30:85–101, 2000.

[59] Roderich Groß and Marco Dorigo. Cooperative Transport of Objects of Different Shapes and Sizes. In *4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, LNCS 3172, pages 107–118, 2004.

[60] Roderich Groß and Marco Dorigo. Group transport of an object to a target that only some group members may sense. In *8th International Conference on Parallel Problem Solving from Nature*, volume 3242, pages 852–861, 2004.

[61] Swarm-bots, 2011. www.swarm-bots.org. Online; accessed 16-November-2011.

[62] Michael J.B. Krieger, Jean-Bernard Billeter, and Laurent Keller. Ant-like Task Allocation and Recruitment in Cooperative Robots. *Nature*, 406:992–995, 2000.

[63] Alan F.T. Winfield. Foraging robots. In Robert A. Meyers, editor, *Encyclopedia of Complexity and System Science*, pages 3682–3700. Springer, 2009. doi: 10.1007/978-0-387-30440-3_217.

[64] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of Labor in a Group of Robots Inspired by Ants' Foraging Behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25, 2006.

[65] Tucker Balch and Ronald C. Arkin. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots*, 1(1):27–52, 1994.

[66] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots. In *Proceedings of the Association for the Advancement of Artificial Intelligence Spring Symposium*, pages 72–75. Springer, 2006.

[67] G. Caprari, P. Balmer, R. Piguet, and R. Siegwart. The Autonomous Micro Robot ALICE: A platform for Scientific and Commercial Applications. In *Proceedings of the International Symposium on Micromechatronics and Human Science*, pages 231–235, 1998.

[68] I-SWARM. www.i-swarm.org/MainPage/Project/P_Overview1.htm. Online; accessed 16-November-2011.

[69] SYMBRION REPLICATOR. www.symbrion.eu. Online; accessed 16-November-2010.

[70] Francesco Mondada, Giovanni C. Pettinaro, Andre Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella, and Marco Dorigo. SWARM-BOT: A New Distributed Robotic Concept. *Autonomous Robots: A Special Issue on Swarm Robotics*, 17(2-3):193–221, 2004.

[71] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stéhane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a Robot Designed for Education in Engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. IEEE, 2009.

[72] e-puck. www.e-puck.org. Online; accessed 16-November-2011.

[73] Serge Kernbach, Oliver Scholz, Kanako Harada, Sergej Popesku, Jens Liedke, Raja Humza, Wenguo Liu, Fabio Caparrelli, Jaouhar Jemai, Jiri Havlik, Eugen Meister, and Paul Levi. Multi-Robot Organisms: State of the Art. In *2010 IEEE International Conference on Robotics and Automation, Workshop on Modular Robots: State of the Art*, pages 1–10. IEEE, 2010.

[74] Mirko Kovac, Wassim Hraiz, Oriol Fauria Torrent, Jean-Christophe Zufferey, and Dario Floreano. The EPFL jumpglider: A hybrid jumping and gliding robot with rigid or folding wings. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1503 –1508, 2011.

[75] Swarmanoid. www.swarmanoid.org. Online; accessed 16-November-2011].

[76] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and The Reality Gap: The Use of Simulation in Eolutionary Robotics. In Federico Morán, Alvaro Moreno, Julián Merelo, and Pablo Chacó, editors, *Advances in Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 704–720. Springer, 1995.

[77] Alan F.T. Winfield, Christopher J. Harper, and Julien Nembrini. Towards Dependable Swarms and a New Discipline of Swarm Engineering. In *2004 SAB Swarm Robotics Workshop*, LNCS 3342, pages 133–148. Springer-Verlag, 2004.

[78] Algirdas Avižienis, Jean-Claude Laprie, and Brian Randell. Dependability and its Threats: A Taxonomy. In *Building the Information Society: Proceedings of the IFIP 18th World Computer Congress*, pages 91–120, 2004.

[79] Jean-Claude Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology. In *Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15)*, pages 2–11. IEEE, 1985.

[80] D. M. Lyons, R. Vijaykumar, and S. T. Venkataraman. A Representation for Error Detection and Recovery in Robot Task Plans. In *Proceedings of the SPIE Symposium on Intelligent Control and Adaptive Systems*, pages 14–25. SPIE Press, 1989.

[81] Mostafa Abd-El-Barr. *Design And Analysis of Reliable And Fault-Tolerant Computer Systems*. Imperial College Press, 2006.

[82] J. R. Sklaroff. Redundancy management technique for space shuttle computers. *IBM Journal of Research and Development*, 20(1):20–28, 1976.

[83] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis Part I: Quantitative model-based methos. *Computers and Chemical Engineering*, 27: 293–311, 2003.

[84] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis Part II: Qualitative models and search strategies. *Computers and Chemical Engineering*, 27:313–326, 2003.

[85] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis Part III: Process history based methods. *Computers and Chemical Engineering*, 27: 327–346, 2003.

[86] Amelia Ritahani Ismail and Jon Timmis. Towards Self-Healing Swarm Robotic Systems Inspired by Granuloma Formation. In *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 313–314. IEEE, 2010.

[87] Ran Bi, Jon Timmis, and Andy Tyrrell. The Diagnostic Dendritic Cell Algorithm for Robotic Systems. In *Proceedings of the 12th IEEE Congress on Evolutionary Computation*, pages 4280–4287. IEEE, 2010.

[88] Jürgen Heinze and Bartosz Walter. Moribund Ants Leave Their Nests to Die in Social Isolation. *Current Biology*, 20:249–252, 2010.

[89] Leon Festinger. A Theory of Social Comparison Processes. *Human Relations*, 7(2):117–140, 1954.

[90] Marco Henrique Terra and Renato Tinós. Fault Detection and Isolation in Robotic Manipulators via Neural Networks : A Comparison Among Three Architectures for Residual Analysis. *Journal on Robotic Systems*, 18(7):357–374, 2001.

[91] Elias N. Skoundrianos and Spyros G. Tzafestas. Finding fault. *IEEE Robotics Automation Magazine*, pages 83–90, Sept 2004.

[92] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

[93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In D. E. Rumelhart, J. L. McClelland, and PDP Research Group, editors, *Parallel Distributed Processing*, pages 318–362. MIT Press, 1986.

[94] Stergios I. Roumeliotis, Gaurav S. Sukhatme, and George A. Bekey. Sensor fault detection and identification in a mobile robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1383–1388. IEEE Computer Society Press, 1998.

[95] Puneet Goel, Göksel Dedeoglu, Stergios I. Roumeliotis, and Gaurav S. Sukhatme. Fault detection and identification in a mobile robot using multiple model estimation and neural network. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2302–2309. IEEE Computer Society Press, 2000.

[96] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-Nonself Discrimination in a Computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE, 1994.

[97] Julie Greensmith, Uwe Aickelin, and Jamie Twycross. Articulation and Clarification of the Dendritic Cell Algorithm. In *Proceedings of the 5th International Conference on Artificial Immune Systems*, pages 404–417. Springer, 2006.

[98] L. E. Parker. An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[99] B. P. Gerkey and M. J Matarić. Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *IEEE International Conference on Robotics and Automation*, pages 464–469. IEEE, 2002.

[100] M. B. Dias, M. B. Zinck, R. M. Zlot, and A. Stentz. Robust Multirobot Coordination in Dynamic Environments. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3435 – 3442. IEEE, 2004.

[101] R. G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29 (12):1104–1113, 1980.

[102] Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo. From Fireflies to Fault-Tolerant Swarms of Robots. *IEEE Transactions on Evolutionary Computation*, 13:754–766, 2009.

[103] Emma Hart and Jon Timmis. Application areas of AIS: The past, the present and the future. *Applied Soft Computing*, 8:191–201, 2008.

[104] Susan Stepney, Robert E. Smith, Jonathan Timmis, and Andy M. Tyrrell. Towards a Conceptual Framework for Artificial Immune Systems. In *Proceedings of the 3rd International Conference on Artificial Immune Systems*, pages 53–64. Springer, 2004.

[105] Yoshiteru Ishida. Fully distributed diagnosis by PDP learning algorithm: Towards immune network PDP model. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 1990)*, pages 777–782. IEEE, 1990.

[106] Hugues Bersini and Francisco J. Varela. Hints for Adaptive Problem Solving Gleaned from Immune Networks. In *Proceedings of the Workshop Parallel Problem Solving from Nature*, LNCS 496, pages 343–354. Springer, 1990.

[107] J. Timmis, E. Hart, A. Hone, M. Neal, A. Robins, S. Stepney, and A. Tyrrell. Immuno-Engineering. In *2nd IFIP International Conference on Biologically Inspired Collaborative Computing, 20th IFIPWorld Computer Congress*, pages 3–17. Springer, 2008.

[108] F. M. Burnet. *The clonal selection theory of acquired immunity*. Cambridge University Press, Cambridge, 1959.

[109] N. K. Jerne. Towards a Network Theory of the Immune System. *Annual Immunology*, 125C:373–389, 1974.

[110] Polly Matzinger. Tolerance, danger and the extended family. *Annual Review in Immunology*, 12:991–1045, 1994.

[111] Zvi Grossman and William E. Paul. The tunable activation threshold and the significance of subthreshold responses. In *Proceedings of the National Academy of Science*, volume 89, pages 10365–10369, 1992.

[112] Leandro Nunes de Castro and Fernando Von Zuben. An Evolutionary Immune Network for Data Clustering. In *Proceedings of the 6th Brazilian Symposium on Neural Networks*, pages 84–89. IEEE Computer Society, 2000.

[113] Jon Timmis and Mark Neal. Investigating the evolution and stability of a resource limited artificial immune system. In *Proceedings of the Conference on Genetics and Evolutionary Computation - special workshop on Artificial Immune Systems*, pages 40–41. AAAI press, 2000.

[114] Leandro N. de Castro and Fernando J. Von Zuben. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computing*, 6(3):239–251, 2002.

[115] Nick D. L. Owens, Jon Timmis, Andrew Greensted, and Andy Tyrrell. Modelling the Tunability of Early T Cell Signalling Events. In Peter J. Bentley, D. H. Lee, and S. W. Jung, editors, *Proceedings of the 7th International Conference on Artificial Immune Systems*, pages 12–23. Springer, 2008.

[116] C.A. Janeway, P. Travers, M. Walport, and M. J. Chlomchik. *Immunobiology: The Immune System in Health and Disease*. Garland Publishing, sixth edition, 2005.

[117] Nick D. L. Owens. *From Biology to Algorithms*. PhD thesis, University of York, 2010.

[118] Uwe Aickelin, Peter J. Bentley, Steve Cayzer, Jungwon Kim, and Julie McLeod. Danger Theory: The Link between AIS and IDS. In *Proceedings of the 2nd International Conference on Artificial Immune Systems*, pages 147–155. Springer, 2003.

[119] Ronald N. Germain and Irena Stefanová. The Dynamics of T-cell Receptor Signalling: Complex Orchestration and the Key Roles of Tempo and Cooperation. *Annual Review of Immunology*, 17:467–522, 1999.

[120] Nick D. L. Owens, Jon Timmis, Andrew Greensted, and Andy Tyrrell. Modelling the Tunability of Early T-Cell Signalling Events. In *Proceedings of the 2nd International Conference on Artificial Immune Systems*, pages 12–23. Springer, 2008.

[121] Akio Ishiguro, Yuji Watanabe, Toshiyuki Kondo, Yasuhiro Shirai, and Yoshiki Uchikawa. A Robot with a Decentralized Consensus-making Mechanism Based on the Immune System. In *Proceedings of the 3rd International Symposium on Autonomous Decentralized Systems*, pages 231–237. IEEE, 1997.

[122] Surya P. N. Singh and Scott M. Thayer. Immunology Directed Methods for Distributed Robotics: A Novel, Immunity-Based Architecture for Robust Control & Coordination. In *Proceedings of SPIE: Mobile Robots XVI*, pages 44–55. The International Society for Optical Engineering, 2001.

[123] Michael Krautmacher and Werner Dilger. AIS Based Robot Navigation in a Rescue Scenario. In *Proceedings of the 3th International Conference on Artificial Immune Systems*, pages 106–118. Springer, 2004.

[124] Amanda M. Whitbrook, Uwe Aickelin, and Jonathan M. Garibaldi. Idiotypic Immune Networks in Mobile-Robot Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(6):1581 –1598, 2007. doi: 10.1109/TSMCB.2007.907334.

[125] Mark Neal, Jan Feyereisl, Rosario Rascuna, and Xiaolei Wang. Don't Touch Me, I'm Fine: Robot Autonomy Using an Artificial Innate Immune System. In *Proceedings of the 5th International Conference on Artificial Immune Systems*, pages 349–361. Springer, 2006.

[126] Teuvo Kohonen. *Self-organising Maps*. Springer, 1995.

[127] Bojan Jakimovski and Erik Maehle. Artificial Immune System Based Robot Anomaly Detection Engine for Fault Tolerant Robots. In *Proceedings of the 5th International Conference on Autonomic and Trusted Computing*, pages 177–190. Springer, 2008.

[128] Kathrin Steck, Bill S. Hansson, and Markus Knaden. Smells like home: Desert ants, *Cataglyphis fortis*, use olfactory landmarks to pinpoint the nest. *Frontiers in Zoology*, 6, 2009. doi: 10.1186/1742-9994-6-5.

[129] Shervin Nouyan, Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Teamwork in Self-Organized Robot Colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695 –711, 2009.

[130] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[131] Jenny Owen. Player/Stage Manual. http://www-users.cs.york.ac.uk/~jowen/player/playerstage-manual.html. Online; accessed 21-March-2012.

[132] Player/Stage Documentation. http://playerstage.sourceforge.net/index.php?src=doc. Online; accessed 21-March-2012.

[133] Thesis Supplementary Materials. http://sites.google.com/site/researchmaterialshkl/thesis-related. Online; accessed 21-March-2012.

[134] GSL - GNU Scientific Library. www.gnu.org/software/gsl. Online; accessed 16-November-2011.

[135] Robert D. Gibbons. *Statistical Methods for Groundwater Monitoring*. John Wiley & Sons, Inc, 1994.

[136] Harvey Motulsky. *Intuitive Biostatistics*. Oxford University Press, 1995.

[137] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[138] Victoria J. Hodge and Jim Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.

[139] William M. K Trochim and James P. Donnelly. *The Research Methods Knowledge Base*. Atomic Dog Publishing, 2007.

[140] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[141] NIST/SEMATECH. e-handbook of statistical methods, 2010. URL www.itl.nist.gov/div898/handbook/. [Online; accessed 16-November-2010].

[142] B. W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta*, 405:442–451, 1975.

[143] András Vargha and Harold D. Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal on Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[144] James A. Hilder, Nick D. L. Owens, Peter J. Hickey, Stuart N. Cairns, David P. A. Kilgour, Jon Timmis, and Andy M. Tyrrell. Parameter Optimisation in the Receptor Density Algorithm. In *Proceedings of the 11th International Conference On Artificial Immune Systems*, pages 226–239. Springer, 2011.

[145] L. Doherty, B.A. Warneke, B.E. Boser, and K.S.J. Pister. Energy and Performance Considerations for Smart Dust. *International Journal on Parallel and Distributed Systems and Networks*, 4(3):121–133, 2001.

[146] David B. Rorabacher. Statistical Treatment for Rejection of Deviant Values: Critical Values of Dixon's "Q" Parameter and Related Subrange Ratios at the 95 Confidence Level. *Analytical Chemistry*, 63(2):139–146, 1991.

APPENDIX

A

Tables for Statistical Testing

Table A.1: Critical values of Dixon's Q-test [146].

Number of observations n	Confidence Level α					
	80%	90%	95%	96%	98%	99%
3	0.886	0.941	0.970	0.976	0.988	0.994
4	0.679	0.765	0.829	0.846	0.889	0.926
5	0.557	0.642	0.710	0.729	0.780	0.821
6	0.482	0.560	0.625	0.644	0.698	0.740
7	0.434	0.507	0.568	0.586	0.637	0.680
8	0.399	0.468	0.526	0.543	0.590	0.634
9	0.370	0.437	0.493	0.510	0.555	0.598
10	0.349	0.412	0.466	0.483	0.527	0.568
11	0.332	0.392	0.444	0.460	0.502	0.542
12	0.318	0.376	0.426	0.441	0.482	0.522
13	0.305	0.361	0.410	0.425	0.465	0.503
14	0.294	0.349	0.396	0.411	0.450	0.488
15	0.285	0.338	0.384	0.399	0.438	0.475
16	0.277	0.329	0.374	0.388	0.426	0.463
17	0.269	0.320	0.365	0.379	0.416	0.452
18	0.263	0.313	0.356	0.370	0.407	0.442
19	0.258	0.306	0.349	0.363	0.398	0.433
20	0.252	0.300	0.342	0.356	0.391	0.425
21	0.247	0.295	0.337	0.350	0.384	0.418
22	0.242	0.290	0.331	0.344	0.378	0.411
23	0.238	0.285	0.326	0.338	0.372	0.404
24	0.234	0.281	0.321	0.333	0.367	0.399
25	0.230	0.277	0.317	0.329	0.362	0.393

Table A.2: Critical values of T-test

Degree of freedom df	p					
	0.10	0.05	0.025	0.01	0.005	0.0005
1	3.078	6.314	12.706	31.821	63.657	636.619
2	1.886	2.920	4.303	6.965	9.925	31.598
3	1.638	2.353	3.182	4.541	5.841	12.941
4	1.533	2.132	2.776	3.747	4.604	8.610
5	1.476	2.015	2.571	3.365	4.032	6.859
6	1.440	1.943	2.447	3.143	3.707	5.959
7	1.415	1.895	2.365	2.998	3.499	5.405
8	1.397	1.860	2.306	2.896	3.355	5.041
9	1.383	1.833	2.262	2.821	3.250	4.781
10	1.372	1.812	2.228	2.764	3.169	4.587
11	1.363	1.796	2.201	2.718	3.106	4.437
12	1.356	1.782	2.179	2.681	3.055	4.318
13	1.350	1.771	2.160	2.650	3.012	4.221
14	1.345	1.761	2.145	2.624	2.977	4.140
15	1.341	1.753	2.131	2.602	2.947	4.073
16	1.337	1.746	2.120	2.583	2.921	4.015
17	1.333	1.740	2.110	2.567	2.898	3.965
18	1.330	1.734	2.101	2.552	2.878	3.922
19	1.328	1.729	2.093	2.539	2.861	3.883
20	1.325	1.725	2.086	2.528	2.845	3.850
21	1.323	1.721	2.080	2.518	2.831	3.819
22	1.321	1.717	2.074	2.508	2.819	3.792
23	1.319	1.714	2.069	2.500	2.807	3.767
24	1.318	1.711	2.064	2.492	2.797	3.745
25	1.316	1.708	2.060	2.485	2.787	3.725

APPENDIX

B

Statistical Classifiers Supplementary Results

This appendix presents the supplementary results for the experiments in Chapter 6 on the ability of statistical classifiers in detecting errors in non-dynamic and dynamic environments.

For each classifier, the data were tested with a range of values for the detection threshold. The best results of each classifier according to the MCC score was then used to compare the error-detection ability of the four classifiers: the ESD, T-test, Quartile-based, and Q-test classifiers. Presented here are the full results for the detection of errors due to faults of the wheels: $\mathtt{P}_{\mathtt{CP}}$, $\mathtt{P}_{\mathtt{PT}}$ = 45 $\times 10^{-3}$ m.s^{-1}, $\mathtt{P}_{\mathtt{PT}}$ = 100 $\times 10^{-5}$ m.s^{-2}.

The range of values for the detection threshold is as shown in the following table.

Classifier	Parameter	Value
ESD	k	0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0
T-Test	p	0.40, 0.25, 0.10, 0.05, 0.025, 0.01, 0.005, 0.0005
Quartile	k	0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0
Q-Test	α	80%, 90%, 95%, 96%, 99%

Table B.1: The median TPR, FPR, Latency (LT), and MCC of the classifiers as the detection threshold is varied in detecting the P_{CP} errors.

Classifier	$k/p/k/\alpha$	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
ESD	0.0	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.00	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	1.0	1.00	0.59	1.0	0.22	0.98	0.61	1.0	0.20	1.00	0.63	1.0	0.21
	1.5	1.00	0.59	1.0	0.22	0.98	0.61	1.0	0.20	1.00	0.63	1.0	0.21
	2.0	0.95	0.28	1.0	0.37	0.88	0.31	1.0	0.32	0.95	0.31	1.0	0.36
	2.5	0.95	0.28	1.0	0.37	0.88	0.31	1.0	0.32	0.95	0.31	1.0	0.36
	3.0	0.88	0.18	1.0	0.44	0.80	0.19	1.0	0.38	0.85	0.20	1.0	0.41
T-Test	0.40	0.77	0.66	1.0	0.06	0.66	0.57	1.0	0.05	0.73	0.67	1.0	0.03
	0.25	0.77	0.65	1.0	0.06	0.66	0.56	1.0	0.06	0.73	0.66	1.0	0.04
	0.10	0.77	0.42	1.0	0.18	0.66	0.38	1.0	0.17	0.73	0.47	1.0	0.16
	0.05	0.77	0.33	1.0	0.25	0.66	0.29	1.0	0.24	0.73	0.35	1.0	0.23
	0.025	0.77	0.26	1.0	0.30	0.66	0.23	1.0	0.29	0.73	0.28	1.0	0.28
	0.010	0.77	0.19	1.0	0.37	0.66	0.16	1.0	0.35	0.73	0.20	1.0	0.36
	0.005	0.77	0.14	1.0	0.43	0.66	0.13	1.0	0.40	0.72	0.15	1.0	0.40
	0.0005	0.74	0.06	1.0	0.56	0.60	0.06	1.0	0.50	0.66	0.07	1.0	0.51
Quartile	0.0	1.00	0.51	1.0	0.26	1.00	0.50	1.0	0.27	1.00	0.50	1.0	0.27
	0.5	1.00	0.24	1.0	0.44	0.85	0.24	1.0	0.35	0.98	0.26	1.0	0.41
	1.0	0.84	0.18	1.0	0.42	0.65	0.17	1.0	0.31	0.76	0.19	1.0	0.36
	1.5	0.70	0.12	1.0	0.41	0.46	0.10	1.0	0.26	0.59	0.13	1.0	0.32
	2.0	0.65	0.10	1.0	0.40	0.42	0.09	1.0	0.26	0.55	0.11	1.0	0.31
	2.5	0.63	0.08	1.0	0.40	0.42	0.08	1.0	0.25	0.50	0.10	1.5	0.29
	3.0	0.61	0.08	1.0	0.41	0.37	0.07	1.0	0.25	0.44	0.09	2.0	0.27
Q-Test	80	0.93	0.17	1.0	0.47	0.81	0.18	1.0	0.38	0.89	0.19	1.0	0.44
	90	0.86	0.12	1.0	0.51	0.64	0.13	1.0	0.36	0.75	0.13	1.0	0.42
	95	0.76	0.10	1.0	0.49	0.58	0.11	1.0	0.34	0.63	0.11	1.0	0.40
	96	0.73	0.10	1.0	0.49	0.56	0.11	1.0	0.32	0.63	0.10	1.5	0.40
	99	0.58	0.08	1.0	0.39	0.36	0.09	2.0	0.23	0.41	0.08	2.0	0.30

Table B.2: The median TPR, FPR, Latency (LT), and MCC of the classifiers as the detection threshold is varied in detecting P_{PT} = 45 $\times 10^{-3}$ m.s^{-1}.

Classifier	$k/p/k/\alpha$	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
ESD	0.0	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	1.0	1.00	0.54	1.0	0.24	1.00	0.57	1.0	0.23	1.00	0.58	1.0	0.23
	1.5	1.00	0.54	1.0	0.24	1.00	0.57	1.0	0.23	1.00	0.58	1.0	0.23
	2.0	0.96	0.28	1.0	0.38	0.92	0.30	1.0	0.33	0.97	0.30	1.0	0.37
	2.5	0.96	0.28	1.0	0.38	0.92	0.30	1.0	0.33	0.97	0.30	1.0	0.37
	3.0	0.88	0.17	1.0	0.45	0.83	0.20	1.0	0.39	0.90	0.19	1.0	0.43
T-test	0.40	0.72	0.64	1.0	0.04	0.65	0.58	2.0	0.04	0.78	0.68	1.0	0.06
	0.25	0.72	0.63	1.0	0.05	0.65	0.55	2.0	0.05	0.78	0.68	1.0	0.06
	0.10	0.72	0.40	1.0	0.18	0.65	0.38	2.0	0.15	0.78	0.45	1.0	0.17
	0.05	0.72	0.31	1.0	0.23	0.65	0.29	2.0	0.20	0.78	0.34	1.0	0.24
	0.025	0.72	0.24	1.0	0.29	0.65	0.23	2.0	0.26	0.78	0.26	1.0	0.30
	0.010	0.72	0.17	1.0	0.37	0.65	0.16	2.0	0.32	0.78	0.18	1.0	0.39
	0.005	0.72	0.13	1.0	0.43	0.65	0.12	2.0	0.38	0.78	0.14	1.0	0.43
	0.0005	0.71	0.06	1.0	0.56	0.64	0.06	2.0	0.51	0.73	0.06	1.0	0.57
Quartile	0.0	1.00	0.43	1.0	0.30	1.00	0.43	1.0	0.30	0.98	0.44	1.0	0.29
	0.5	0.85	0.23	1.0	0.36	0.81	0.22	1.0	0.34	0.86	0.23	1.0	0.36
	1.0	0.81	0.17	1.0	0.39	0.73	0.16	1.0	0.37	0.73	0.16	1.0	0.37
	1.5	0.78	0.12	1.0	0.47	0.70	0.10	1.0	0.45	0.74	0.11	1.0	0.44
	2.0	0.78	0.10	1.0	0.50	0.69	0.08	1.0	0.47	0.73	0.10	1.0	0.47
	2.5	0.78	0.09	1.0	0.51	0.69	0.07	1.0	0.50	0.73	0.08	1.0	0.50
	3.0	0.78	0.08	1.0	0.53	0.68	0.06	1.0	0.51	0.73	0.08	1.0	0.52
Q-test	80	0.98	0.25	1.0	0.42	0.95	0.28	1.0	0.38	0.94	0.26	1.0	0.40
	90	0.92	0.20	1.0	0.43	0.89	0.22	1.0	0.39	0.89	0.20	1.0	0.42
	95	0.87	0.17	1.0	0.44	0.80	0.18	1.0	0.38	0.80	0.16	1.0	0.40
	96	0.84	0.16	1.0	0.44	0.77	0.18	1.0	0.37	0.77	0.15	1.0	0.40
	99	0.68	0.12	1.0	0.39	0.58	0.14	1.5	0.29	0.57	0.12	2.0	0.33

Table B.3: The median TPR, FPR, Latency (LT), and MCC of the classifiers as the detection threshold is varied in detecting P_{GR} = 100 $\times 10^{-5}$ m.s^{-2}.

Classifiers	$k/p/k/\alpha$	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
ESD	0.0	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	1.0	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27	1.00	0.52	1.0	0.26
	1.5	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27	1.00	0.52	1.0	0.26
	2.0	1.00	0.25	1.0	0.43	0.99	0.28	1.0	0.40	0.98	0.28	1.0	0.40
	2.5	1.00	0.25	1.0	0.43	0.99	0.28	1.0	0.40	0.98	0.28	1.0	0.40
	3.0	0.98	0.16	1.0	0.52	0.95	0.18	1.0	0.48	0.95	0.17	1.0	0.51
T-test	0.40	0.87	0.65	1.0	0.13	0.80	0.58	1.0	0.11	0.82	0.67	1.0	0.08
	0.25	0.87	0.64	1.0	0.13	0.80	0.58	1.0	0.12	0.82	0.66	1.0	0.09
	0.10	0.87	0.35	1.0	0.29	0.80	0.32	1.0	0.27	0.82	0.39	1.0	0.24
	0.05	0.87	0.27	1.0	0.35	0.80	0.25	1.0	0.32	0.82	0.30	1.0	0.29
	0.025	0.87	0.22	1.0	0.40	0.80	0.20	1.0	0.38	0.82	0.23	1.0	0.35
	0.010	0.87	0.16	1.0	0.46	0.80	0.14	1.0	0.47	0.82	0.17	1.0	0.43
	0.005	0.87	0.13	1.0	0.50	0.80	0.10	1.0	0.52	0.82	0.12	1.0	0.48
	0.0005	0.87	0.06	1.0	0.68	0.80	0.05	1.0	0.65	0.82	0.06	1.0	0.63
Quartile	0.0	1.00	0.41	1.0	0.31	1.00	0.41	1.0	0.31	1.00	0.42	1.0	0.30
	0.5	1.00	0.22	1.0	0.45	1.00	0.21	1.0	0.47	1.00	0.22	1.0	0.45
	1.0	0.86	0.16	1.0	0.45	0.85	0.15	1.0	0.46	0.85	0.16	1.0	0.44
	1.5	0.81	0.11	1.0	0.52	0.77	0.10	1.0	0.49	0.76	0.11	1.0	0.48
	2.0	0.81	0.09	1.0	0.55	0.77	0.08	1.0	0.53	0.76	0.10	1.0	0.50
	2.5	0.80	0.08	1.0	0.55	0.77	0.07	1.0	0.55	0.75	0.08	1.0	0.53
	3.0	0.80	0.08	1.0	0.57	0.77	0.06	1.0	0.56	0.75	0.08	1.0	0.55
Q-Test	80	1.00	0.17	1.0	0.52	1.00	0.18	1.0	0.50	1.00	0.18	1.0	0.51
	90	1.00	0.12	1.0	0.60	0.97	0.13	1.0	0.54	0.98	0.13	1.0	0.57
	95	0.97	0.10	1.0	0.63	0.94	0.11	1.0	0.57	0.93	0.10	1.0	0.60
	96	0.95	0.09	1.0	0.63	0.92	0.10	1.0	0.56	0.93	0.10	1.0	0.60
	99	0.87	0.07	1.0	0.61	0.83	0.10	2.0	0.54	0.85	0.08	1.0	0.61

APPENDIX

C

Supplementary Results for Variants of the $\mathtt{P_{PT}}$ and $\mathtt{P_{GR}}$

This appendix presents the supplementary results as part of the experiments in Chapter 7 on the performance of the RDA classifier. Here, the performance of the statistical classifiers in detecting the variants of the $\mathtt{P}_{\mathtt{PT}}$ and $\mathtt{P}_{\mathtt{GR}}$ are presented.

C.1 Variants of the $\mathtt{P_{PT}}$

Here the full results on the performance in detecting the variants of $\mathtt{P}_{\mathtt{PT}}$ errors with the statistical classifiers are presented. The variants of the $\mathtt{P}_{\mathtt{PT}}$: 45, 60, 75, 80, 85, 90, 95, 105 $\times 10^{-3}$ m.s $^{-1}$. Results are in Tables C.1 for the ESD classifier, C.2 for the T-test classifier, C.3 for the Quartile-based classifier, and C.4 for the Q-Test classifier.

Table C.1: The median TPR, FPR, Latency (LT), and MCC of the ESD classifier using a range of k values in detecting the variants of the P_{PT} from 45 $\times 10^{-3}$ m.s^{-1} to 105 $\times 10^{-3}$ m.s^{-1}.

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
45	0.0	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	1.0	1.00	0.54	1.0	0.24	1.00	0.57	1.0	0.23	1.00	0.58	1.0	0.23
	1.5	1.00	0.54	1.0	0.24	1.00	0.57	1.0	0.23	1.00	0.58	1.0	0.23
	2.0	0.96	0.28	1.0	0.38	0.92	0.30	1.0	0.33	0.97	0.30	1.0	0.37
	2.5	0.96	0.28	1.0	0.38	0.92	0.30	1.0	0.33	0.97	0.30	1.0	0.37
	3.0	0.88	0.17	1.0	0.45	0.83	0.20	1.0	0.39	0.90	0.19	1.0	0.43
60	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.61	1.0	0.21	1.00	0.61	1.0	0.22	1.00	0.62	1.0	0.21
	1.5	1.00	0.61	1.0	0.21	1.00	0.61	1.0	0.22	1.00	0.62	1.0	0.21
	2.0	0.91	0.31	1.0	0.33	0.87	0.32	1.0	0.29	0.95	0.32	1.0	0.34
	2.5	0.91	0.31	1.0	0.33	0.87	0.32	1.0	0.29	0.95	0.32	1.0	0.34
	3.0	0.83	0.19	1.0	0.40	0.77	0.20	1.0	0.33	0.89	0.19	1.0	0.43
75	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	0.99	0.64	1.0	0.20	0.98	0.62	1.0	0.20	1.00	0.66	1.0	0.19
	1.5	0.99	0.64	1.0	0.20	0.98	0.62	1.0	0.20	1.00	0.66	1.0	0.19
	2.0	0.88	0.34	1.0	0.29	0.83	0.33	1.0	0.26	0.90	0.34	1.0	0.30
	2.5	0.88	0.34	1.0	0.29	0.83	0.33	1.0	0.26	0.90	0.34	1.0	0.30
	3.0	0.82	0.21	1.0	0.36	0.77	0.21	1.0	0.33	0.80	0.21	1.0	0.36
80	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	1.00	0.52	1.0	0.01	1.00	0.62	1.0	0.20	1.00	0.65	1.0	0.18
	1.5	1.00	0.64	1.0	0.19	1.00	0.64	1.0	0.20	1.00	0.65	1.0	0.18
	2.0	0.88	0.34	1.0	0.29	0.79	0.35	1.0	0.24	0.88	0.35	1.0	0.29
	2.5	0.88	0.34	1.0	0.29	0.79	0.35	1.0	0.24	0.88	0.35	1.0	0.29
	3.0	0.78	0.21	1.0	0.34	0.72	0.22	1.0	0.30	0.80	0.20	1.0	0.36
85	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	1.00	0.66	1.0	0.19	0.98	0.63	1.0	0.19	0.99	0.66	1.0	0.18
	1.5	1.00	0.66	1.0	0.19	0.98	0.63	1.0	0.19	0.99	0.66	1.0	0.18
	2.0	0.87	0.34	1.0	0.28	0.83	0.34	1.0	0.26	0.86	0.36	1.0	0.28
	2.5	0.87	0.34	1.0	0.28	0.83	0.34	1.0	0.26	0.86	0.36	1.0	0.28
	3.0	0.80	0.21	1.0	0.37	0.78	0.21	1.0	0.34	0.75	0.21	1.0	0.34
90	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	0.98	0.65	1.0	0.19	0.98	0.64	0.0	0.19	1.00	0.66	1.0	0.19
	1.5	0.98	0.65	1.0	0.19	0.98	0.64	1.0	0.19	1.00	0.66	1.0	0.19
	2.0	0.83	0.35	1.0	0.26	0.78	0.34	1.0	0.23	0.83	0.35	1.0	0.27
	2.5	0.83	0.35	1.0	0.26	0.78	0.34	1.0	0.23	0.83	0.35	1.0	0.27
	3.0	0.76	0.21	1.0	0.33	0.68	0.23	1.0	0.29	0.75	0.21	1.0	0.33

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
95	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	0.98	0.66	1.0	0.18	0.97	0.65	0.0	0.18	0.97	0.67	0.0	0.17
	1.5	0.98	0.66	1.0	0.18	0.97	0.65	1.0	0.18	0.97	0.67	1.0	0.17
	2.0	0.83	0.35	1.0	0.26	0.75	0.36	1.0	0.20	0.82	0.36	1.0	0.25
	2.5	0.83	0.35	1.0	0.26	0.75	0.36	1.0	0.20	0.82	0.36	1.0	0.25
	3.0	0.78	0.21	1.0	0.35	0.68	0.22	1.0	0.27	0.73	0.22	1.0	0.31
105	0.0	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	0.0	0.00	1.00	0.00	0.0	0.00
	1.0	0.95	0.67	1.0	0.16	0.97	0.65	0.0	0.17	0.97	0.68	0.0	0.16
	1.5	0.95	0.67	1.0	0.16	0.97	0.65	1.0	0.17	0.97	0.68	1.0	0.16
	2.0	0.80	0.36	1.0	0.24	0.77	0.36	1.0	0.21	0.75	0.37	1.0	0.21
	2.5	0.80	0.36	1.0	0.24	0.77	0.36	1.0	0.21	0.75	0.37	1.0	0.21
	3.0	0.69	0.22	1.0	0.29	0.72	0.24	1.0	0.29	0.59	0.21	1.0	0.24

Table C.2: The median TPR, FPR, Latency (LT), and MCC of the T-test classifier using a range of p values in detecting the variants of the P_{PT} from 45 $\times 10^{-3}$ m.s^{-1} to 105 $\times 10^{-3}$ m.s^{-1}.

P_{PT}	p	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
45	0.40	0.72	0.64	1.0	0.04	0.65	0.58	2.0	0.04	0.78	0.68	1.0	0.05
	0.25	0.72	0.63	1.0	0.05	0.65	0.57	2.0	0.05	0.78	0.68	1.0	0.06
	0.10	0.72	0.40	1.0	0.18	0.65	0.38	2.0	0.15	0.78	0.45	1.0	0.17
	0.05	0.72	0.31	1.0	0.23	0.65	0.29	2.0	0.20	0.78	0.34	1.0	0.24
	0.025	0.72	0.24	1.0	0.29	0.65	0.23	2.0	0.26	0.78	0.26	1.0	0.30
	0.010	0.72	0.17	1.0	0.37	0.65	0.16	2.0	0.32	0.78	0.18	1.0	0.39
	0.005	0.72	0.13	1.0	0.43	0.65	0.12	2.0	0.38	0.78	0.14	1.0	0.43
	0.0005	0.71	0.06	1.0	0.56	0.64	0.06	2.0	0.51	0.73	0.06	1.0	0.57
60	0.40	0.68	0.64	1.0	0.02	0.60	0.57	1.0	0.03	0.72	0.66	1.0	0.02
	0.25	0.68	0.62	1.0	0.03	0.60	0.56	1.0	0.03	0.72	0.66	1.0	0.02
	0.10	0.68	0.44	1.0	0.12	0.60	0.41	1.0	0.12	0.72	0.48	1.0	0.12
	0.05	0.68	0.33	1.0	0.19	0.60	0.32	1.0	0.18	0.72	0.36	1.0	0.18
	0.025	0.68	0.26	1.0	0.26	0.60	0.25	1.0	0.23	0.72	0.28	1.0	0.24
	0.010	0.68	0.18	1.0	0.33	0.60	0.17	1.0	0.29	0.72	0.19	1.0	0.32
	0.005	0.68	0.14	1.0	0.38	0.60	0.13	1.0	0.33	0.70	0.15	1.0	0.38
	0.0005	0.64	0.06	1.0	0.51	0.58	0.05	1.0	0.48	0.67	0.07	1.0	0.51
75	0.40	0.66	0.65	1.0	0.00	0.58	0.56	2.50	0.02	0.68	0.68	1.0	0.00
	0.25	0.66	0.63	1.0	0.00	0.58	0.55	2.50	0.02	0.68	0.66	1.0	0.01
	0.10	0.66	0.48	1.0	0.10	0.58	0.42	2.5	0.10	0.68	0.50	1.0	0.09
	0.05	0.66	0.38	1.0	0.15	0.58	0.33	2.5	0.15	0.68	0.40	1.0	0.15
	0.025	0.66	0.29	1.0	0.21	0.58	0.25	2.5	0.20	0.68	0.30	1.0	0.22
	0.010	0.65	0.20	1.0	0.29	0.58	0.17	2.5	0.28	0.68	0.21	1.0	0.29
	0.005	0.65	0.15	1.0	0.34	0.58	0.13	2.5	0.33	0.68	0.15	1.0	0.34
	0.0005	0.61	0.07	1.0	0.48	0.55	0.06	2.5	0.45	0.62	0.06	1.0	0.47

P_{PT}	p	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	Lt	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
80	0.40	0.63	0.64	1.0	0.00	0.56	0.57	1.0	0.00	0.72	0.68	1.0	0.02
	0.25	0.63	0.63	1.0	0.00	0.56	0.56	1.0	0.00	0.72	0.67	1.0	0.03
	0.10	0.63	0.49	1.0	0.08	0.56	0.43	1.0	0.07	0.72	0.53	1.0	0.11
	0.05	0.63	0.38	1.0	0.14	0.56	0.35	1.0	0.12	0.72	0.42	1.0	0.17
	0.025	0.63	0.29	1.0	0.20	0.56	0.27	1.0	0.17	0.72	0.31	1.0	0.22
	0.010	0.63	0.20	1.0	0.28	0.56	0.19	1.0	0.24	0.72	0.21	1.0	0.30
	0.005	0.63	0.15	1.0	0.32	0.56	0.14	1.0	0.30	0.71	0.16	1.0	0.36
	0.0005	0.58	0.07	1.0	0.45	0.52	0.06	2.0	0.42	0.63	0.07	1.0	0.48
85	0.40	0.65	0.64	1.0	0.00	0.58	0.57	1.0	0.01	0.65	0.67	1.0	0.00
	0.25	0.65	0.63	1.0	0.00	0.58	0.56	1.0	0.02	0.65	0.66	1.0	0.00
	0.10	0.65	0.49	1.0	0.07	0.58	0.43	1.0	0.08	0.65	0.52	1.0	0.07
	0.05	0.65	0.39	1.0	0.13	0.58	0.35	1.0	0.13	0.65	0.41	1.0	0.14
	0.025	0.65	0.30	1.0	0.19	0.58	0.27	1.0	0.18	0.65	0.32	1.0	0.19
	0.010	0.64	0.20	1.0	0.27	0.58	0.19	1.0	0.25	0.65	0.23	1.0	0.26
	0.005	0.63	0.16	1.0	0.33	0.58	0.14	1.0	0.31	0.65	0.17	1.0	0.32
	0.0005	0.58	0.07	1.5	0.44	0.53	0.06	1.0	0.43	0.57	0.07	1.0	0.43
90	0.40	0.66	0.64	1.5	0.01	0.55	0.56	1.0	0.00	0.68	0.67	1.5	0.00
	0.25	0.66	0.63	1.5	0.01	0.55	0.55	1.0	0.01	0.68	0.65	1.5	0.01
	0.10	0.66	0.49	1.5	0.08	0.55	0.43	1.0	0.07	0.68	0.52	1.5	0.08
	0.05	0.66	0.39	1.5	0.14	0.55	0.36	1.0	0.12	0.68	0.42	1.5	0.14
	0.025	0.66	0.31	1.5	0.19	0.55	0.27	1.0	0.18	0.68	0.32	1.5	0.20
	0.010	0.66	0.21	1.5	0.27	0.55	0.19	1.0	0.25	0.67	0.22	1.5	0.27
	0.005	0.65	0.17	2.0	0.32	0.55	0.14	1.0	0.29	0.65	0.16	1.5	0.33
	0.0005	0.57	0.08	2.0	0.42	0.50	0.06	1.0	0.40	0.56	0.07	2.0	0.43
95	0.40	0.63	0.63	1.0	0.01	0.54	0.56	1.0	0.00	0.70	0.66	1.0	0.02
	0.25	0.63	0.62	1.0	0.01	0.54	0.55	1.0	0.00	0.70	0.65	1.0	0.02
	0.10	0.63	0.51	1.0	0.08	0.54	0.44	1.0	0.05	0.70	0.52	1.0	0.09
	0.05	0.63	0.40	1.0	0.13	0.53	0.36	1.0	0.10	0.70	0.43	1.0	0.14
	0.025	0.63	0.31	1.0	0.19	0.53	0.28	1.0	0.16	0.70	0.32	1.0	0.21
	0.010	0.63	0.21	1.0	0.27	0.53	0.20	1.0	0.22	0.67	0.23	1.0	0.28
	0.005	0.63	0.16	1.0	0.32	0.53	0.15	1.0	0.28	0.67	0.16	1.0	0.34
	0.0005	0.55	0.07	2.0	0.42	0.48	0.06	1.0	0.38	0.56	0.07	1.0	0.44
105	0.40	0.62	0.64	1.0	0.00	0.52	0.57	1.0	0.00	0.65	0.67	1.0	0.00
	0.25	0.62	0.63	1.0	0.00	0.52	0.56	1.0	0.00	0.65	0.65	1.0	0.00
	0.10	0.62	0.52	1.0	0.05	0.52	0.46	1.0	0.04	0.65	0.55	1.0	0.06
	0.05	0.62	0.42	1.0	0.11	0.52	0.37	1.0	0.09	0.65	0.44	1.0	0.11
	0.025	0.62	0.33	1.0	0.16	0.52	0.29	1.0	0.13	0.64	0.35	1.0	0.16
	0.010	0.60	0.23	1.0	0.23	0.52	0.20	1.0	0.21	0.63	0.24	1.0	0.23
	0.005	0.60	0.17	1.0	0.29	0.52	0.15	1.0	0.25	0.62	0.18	1.0	0.28
	0.0005	0.49	0.07	1.0	0.37	0.45	0.07	1.0	0.35	0.48	0.08	1.0	0.35

Table C.3: The median TPR, FPR, Latency (LT), and MCC of the Quartile-based classifier using a range of k values in detecting the variants of the P_{PT} from 45 $\times 10^{-3}$ m.s^{-1} to 105 $\times 10^{-3}$ m.s^{-1}.

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
45	0.0	1.00	0.43	1.0	0.30	1.00	0.43	1.0	0.30	0.98	0.44	1.0	0.29
	0.5	0.85	0.23	1.0	0.36	0.81	0.22	1.0	0.34	0.86	0.23	1.0	0.36
	1.0	0.81	0.17	1.0	0.39	0.73	0.16	1.0	0.37	0.73	0.16	1.0	0.37
	1.5	0.78	0.12	1.0	0.47	0.70	0.10	1.0	0.45	0.74	0.11	1.0	0.44
	2.0	0.78	0.10	1.0	0.50	0.69	0.08	1.0	0.47	0.73	0.10	1.0	0.47
	2.5	0.78	0.09	1.0	0.51	0.69	0.07	1.0	0.50	0.73	0.08	1.0	0.50
	3.0	0.78	0.08	1.0	0.53	0.68	0.06	1.0	0.51	0.73	0.08	1.0	0.52
60	0.0	0.88	0.45	1.0	0.22	0.95	0.46	1.0	0.26	0.91	0.46	1.0	0.23
	0.5	0.78	0.25	1.0	0.32	0.78	0.25	1.0	0.31	0.81	0.25	1.0	0.33
	1.0	0.76	0.18	1.0	0.38	0.76	0.18	1.0	0.38	0.78	0.18	1.0	0.38
	1.5	0.75	0.12	1.0	0.45	0.68	0.11	1.0	0.41	0.76	0.12	1.0	0.45
	2.0	0.75	0.10	1.0	0.49	0.68	0.10	1.0	0.43	0.75	0.10	1.0	0.46
	2.5	0.75	0.09	1.0	0.51	0.68	0.08	1.0	0.46	0.72	0.09	1.0	0.48
	3.0	0.73	0.08	1.0	0.51	0.68	0.07	1.0	0.47	0.68	0.08	1.0	0.47
75	0.0	0.82	0.48	1.0	0.19	0.81	0.48	1.0	0.18	0.83	0.49	1.0	0.19
	0.5	0.72	0.25	1.0	0.28	0.70	0.25	1.0	0.26	0.76	0.27	1.0	0.30
	1.0	0.71	0.19	1.0	0.34	0.65	0.18	1.0	0.31	0.73	0.19	1.0	0.36
	1.5	0.71	0.13	1.0	0.42	0.63	0.12	1.0	0.38	0.72	0.12	1.0	0.41
	2.0	0.69	0.11	1.0	0.44	0.63	0.10	1.0	0.41	0.68	0.11	1.0	0.42
	2.5	0.68	0.09	1.0	0.48	0.63	0.08	1.0	0.43	0.63	0.09	1.0	0.41
	3.0	0.65	0.09	1.0	0.47	0.63	0.08	1.0	0.45	0.57	0.09	1.0	0.39
80	0.0	0.83	0.47	1.0	0.19	0.82	0.48	1.0	0.18	0.82	0.49	1.0	0.17
	0.5	0.76	0.25	1.0	0.29	0.68	0.25	1.0	0.25	0.75	0.26	1.0	0.30
	1.0	0.76	0.19	1.0	0.34	0.67	0.18	1.0	0.31	0.73	0.19	1.0	0.35
	1.5	0.74	0.13	1.0	0.41	0.66	0.12	1.0	0.39	0.70	0.12	1.0	0.41
	2.0	0.72	0.11	1.0	0.44	0.66	0.11	1.0	0.42	0.68	0.11	1.0	0.41
	2.5	0.71	0.10	1.0	0.46	0.64	0.09	1.0	0.44	0.62	0.09	1.0	0.40
	3.0	0.69	0.09	1.0	0.45	0.63	0.08	1.0	0.44	0.53	0.08	1.0	0.37
85	0.0	0.78	0.50	1.0	0.15	0.80	0.50	1.0	0.17	0.83	0.49	1.0	0.18
	0.5	0.71	0.26	1.0	0.25	0.69	0.27	1.0	0.25	0.76	0.26	1.0	0.29
	1.0	0.71	0.20	1.0	0.33	0.66	0.19	1.0	0.29	0.73	0.20	1.0	0.34
	1.5	0.70	0.13	1.0	0.40	0.62	0.12	1.0	0.36	0.67	0.13	1.0	0.40
	2.0	0.68	0.11	1.0	0.42	0.62	0.11	1.0	0.39	0.64	0.11	1.0	0.40
	2.5	0.63	0.10	1.0	0.43	0.60	0.09	1.0	0.41	0.60	0.09	1.0	0.40
	3.0	0.58	0.09	1.0	0.40	0.58	0.08	1.0	0.40	0.53	0.08	1.0	0.39

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
90	0.0	0.78	0.50	1.0	0.16	0.78	0.49	1.0	0.15	0.80	0.50	1.0	0.16
	0.5	0.73	0.26	1.0	0.25	0.73	0.27	1.0	0.27	0.75	0.27	1.0	0.27
	1.0	0.72	0.19	1.0	0.32	0.68	0.19	1.0	0.30	0.70	0.20	1.0	0.31
	1.5	0.68	0.12	1.0	0.39	0.66	0.12	1.0	0.38	0.65	0.13	1.0	0.37
	2.0	0.67	0.11	1.0	0.40	0.63	0.11	1.0	0.38	0.58	0.11	1.0	0.35
	2.5	0.61	0.10	1.0	0.39	0.63	0.09	1.0	0.40	0.50	0.10	1.0	0.31
	3.0	0.58	0.09	1.0	0.38	0.56	0.09	1.0	0.37	0.43	0.09	1.0	0.28
95	0.0	0.78	0.50	1.0	0.14	0.74	0.49	1.0	0.14	0.83	0.50	1.0	0.17
	0.5	0.70	0.27	1.0	0.24	0.64	0.27	1.0	0.21	0.77	0.27	1.0	0.29
	1.0	0.77	0.27	1.0	0.29	0.60	0.20	1.0	0.25	0.72	0.20	1.0	0.32
	1.5	0.67	0.13	1.0	0.36	0.56	0.12	1.0	0.32	0.65	0.13	1.0	0.34
	2.0	0.62	0.11	1.0	0.36	0.55	0.10	1.0	0.35	0.57	0.11	1.0	0.33
	2.5	0.57	0.10	1.0	0.35	0.53	0.09	1.0	0.35	0.47	0.09	1.0	0.30
	3.0	0.48	0.09	1.0	0.31	0.52	0.08	1.0	0.35	0.39	0.09	1.5	0.26
105	0.0	0.74	0.52	1.0	0.11	0.74	0.50	1.0	0.13	0.78	0.51	1.0	0.14
	0.5	0.68	0.28	1.0	0.23	0.66	0.28	1.0	0.23	0.69	0.28	1.0	0.23
	1.0	0.67	0.21	1.0	0.27	0.63	0.21	1.0	0.27	0.63	0.21	1.0	0.25
	1.5	0.62	0.13	1.0	0.33	0.58	0.13	1.0	0.33	0.53	0.14	1.0	0.28
	2.0	0.54	0.11	2.0	0.31	0.55	0.11	1.0	0.33	0.46	0.12	1.0	0.24
	2.5	0.48	0.09	2.0	0.30	0.50	0.09	1.0	0.33	0.38	0.10	1.0	0.22
	3.0	0.42	0.09	2.0	0.27	0.46	0.08	2.0	0.31	0.29	0.10	1.5	0.17

Table C.4: The median TPR, FPR, Latency (LT), and MCC of the Q-Test classifier using a range of α values in detecting the variants of the P_{PT} from 45 $\times 10^{-3}$ m.s^{-1} to 105 $\times 10^{-3}$ m.s^{-1}.

P_{PT}	α	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
45	80	0.98	0.25	1.0	0.42	0.95	0.28	1.0	0.38	0.94	0.26	1.0	0.40
	90	0.92	0.20	1.0	0.43	0.89	0.22	1.0	0.39	0.89	0.20	1.0	0.42
	95	0.87	0.17	1.0	0.44	0.80	0.18	1.0	0.38	0.80	0.16	1.0	0.40
	96	0.84	0.16	1.0	0.44	0.77	0.18	1.0	0.37	0.77	0.15	1.0	0.40
	99	0.68	0.12	1.0	0.39	0.58	0.14	1.5	0.29	0.57	0.12	2.0	0.33
60	80	0.95	0.26	1.0	0.39	0.96	0.27	1.0	0.37	0.90	0.26	1.0	0.37
	90	0.88	0.20	1.0	0.42	0.87	0.21	1.0	0.39	0.81	0.19	1.0	0.38
	95	0.79	0.16	1.0	0.40	0.82	0.18	1.0	0.39	0.69	0.16	1.0	0.35
	96	0.75	0.15	1.0	0.39	0.79	0.17	1.0	0.39	0.65	0.15	1.0	0.35
	99	0.54	0.12	1.0	0.31	0.59	0.14	2.0	0.31	0.44	0.10	1.0	0.25
75	80	0.90	0.27	1.0	0.36	0.93	0.27	1.0	0.36	0.82	0.26	1.0	0.32
	90	0.82	0.20	1.0	0.38	0.83	0.21	1.0	0.36	0.69	0.19	1.0	0.33
	95	0.73	0.16	1.0	0.37	0.73	0.16	1.0	0.36	0.54	0.15	1.0	0.29
	96	0.72	0.16	1.5	0.37	0.70	0.16	1.0	0.35	0.50	0.14	1.0	0.26
	99	0.49	0.12	3.0	0.28	0.48	0.14	1.0	0.26	0.30	0.10	1.0	0.15

P_{PT}	α	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
80	80	0.89	0.27	1.0	0.35	0.92	0.28	1.0	0.35	0.78	0.26	1.0	0.30
	90	0.81	0.19	1.0	0.37	0.82	0.21	1.0	0.35	0.65	0.18	1.0	0.30
	95	0.70	0.16	1.0	0.36	0.69	0.18	1.0	0.32	0.51	0.15	1.0	0.25
	96	0.67	0.15	1.0	0.35	0.65	0.17	1.0	0.30	0.47	0.14	1.0	0.23
	99	0.44	0.12	2.5	0.27	0.36	0.14	2.0	0.16	0.27	0.10	2.0	0.13
85	80	0.86	0.26	1.0	0.34	0.90	0.26	1.0	0.36	0.78	0.25	1.0	0.30
	90	0.74	0.19	1.0	0.35	0.80	0.20	1.0	0.36	0.65	0.19	1.0	0.30
	95	0.62	0.15	1.5	0.31	0.68	0.17	1.0	0.33	0.48	0.14	1.0	0.24
	96	0.56	0.15	1.5	0.30	0.64	0.16	0.0	0.31	0.45	0.14	1.0	0.23
	99	0.40	0.11	2.0	0.23	0.42	0.12	1.5	0.22	0.28	0.10	2.0	0.15
90	80	0.86	0.26	1.0	0.33	0.88	0.28	1.0	0.34	0.73	0.25	1.0	0.27
	90	0.71	0.20	1.0	0.31	0.74	0.21	1.0	0.33	0.54	0.18	1.0	0.25
	95	0.57	0.16	1.0	0.28	0.60	0.17	2.0	0.29	0.43	0.14	1.0	0.20
	96	0.54	0.15	1.0	0.26	0.54	0.16	2.0	0.27	0.38	0.13	1.0	0.21
	99	0.34	0.11	2.0	0.17	0.36	0.13	2.5	0.17	0.23	0.10	2.0	0.13
95	80	0.79	0.26	1.0	0.30	0.83	0.28	1.0	0.31	0.68	0.26	1.0	0.24
	90	0.67	0.18	1.0	0.29	0.68	0.21	1.0	0.30	0.52	0.19	1.0	0.22
	95	0.53	0.15	1.0	0.28	0.55	0.17	1.5	0.26	0.40	0.15	2.0	0.18
	96	0.48	0.14	1.5	0.26	0.48	0.16	1.5	0.22	0.38	0.14	2.0	0.19
	99	0.33	0.10	3.0	0.16	0.28	0.13	3.0	0.11	0.21	0.10	4.0	0.11
105	80	0.68	0.25	1.0	0.24	0.73	0.27	1.0	0.27	0.53	0.24	1.0	0.18
	90	0.53	0.18	1.0	0.22	0.58	0.20	1.0	0.24	0.38	0.16	2.0	0.15
	95	0.38	0.15	2.0	0.17	0.41	0.16	1.5	0.17	0.28	0.13	2.0	0.12
	96	0.35	0.14	3.0	0.14	0.36	0.15	2.0	0.14	0.26	0.12	2.0	0.12
	99	0.19	0.11	5.5	0.07	0.18	0.12	5.0	0.05	0.18	0.09	4.0	0.08

C.2 Variants of the $\mathtt{P}_{\mathtt{GR}}$

Presented here are the full results on the performance in detecting the variants of $\mathtt{P}_{\mathtt{GR}}$ errors for the statistical classifiers. The variants of the $\mathtt{P}_{\mathtt{GR}}$: 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 $\times 10^{-5}$ m.s $^{-2}$. Results are in Tables C.5 for the ESD classifier, C.6 for the T-test classifier, C.7 for the Quartile-based classifier, and C.8 for the Q-Test classifier.

Table C.5: The median TPR, FPR, Latency (LT), and MCC of the ESD classifier using a range of p values in detecting the variants of the $\mathtt{P}_{\mathtt{GR}}$ from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}.

$\mathtt{P}_{\mathtt{PT}}$	k	CST				$\mathtt{V}_{\mathtt{OPR}}$				$\mathtt{V}_{\mathtt{ODS}}$			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
5	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.49	1.0	0.27	0.98	0.51	1.0	0.24	0.98	0.53	1.0	0.24
	1.5	1.00	0.49	1.0	0.27	0.98	0.51	1.0	0.24	0.98	0.53	1.0	0.24
	2.0	0.96	0.26	1.0	0.40	0.92	0.29	1.5	0.35	0.93	0.28	2.5	0.36
	2.5	0.96	0.26	1.0	0.40	0.92	0.29	1.5	0.35	0.93	0.28	2.5	0.36
	3.0	0.92	0.17	2.0	0.47	0.88	0.19	2.0	0.42	0.89	0.17	3.5	0.46
10	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	0.99	0.45	1.0	0.29	0.98	0.50	1.0	0.26	1.00	0.51	1.0	0.25
	1.5	0.99	0.45	1.0	0.29	0.98	0.50	1.0	0.26	1.00	0.51	1.0	0.25
	2.0	0.98	0.24	1.5	0.42	0.96	0.28	2.0	0.37	0.97	0.27	1.0	0.40
	2.5	0.98	0.24	1.5	0.42	0.96	0.28	2.0	0.37	0.97	0.27	1.0	0.40
	3.0	0.95	0.15	2.5	0.51	0.91	0.18	2.5	0.45	0.95	0.17	2.0	0.49
20	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.26	1.00	0.50	1.0	0.26
	1.5	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.26	1.00	0.50	1.0	0.26
	2.0	0.98	0.24	2.0	0.43	0.98	0.27	2.0	0.39	0.98	0.28	2.0	0.39
	2.5	0.98	0.24	2.0	0.43	0.98	0.27	2.0	0.39	0.98	0.28	2.0	0.39
	3.0	0.97	0.15	2.0	0.52	0.93	0.18	2.0	0.48	0.96	0.17	2.0	0.49
30	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.44	1.0	0.30	1.00	0.47	1.0	0.27	1.00	0.48	1.0	0.27
	1.5	1.00	0.44	1.0	0.30	1.00	0.47	1.0	0.27	1.00	0.48	1.0	0.27
	2.0	0.98	0.24	1.0	0.44	0.98	0.26	1.0	0.41	0.98	0.27	1.0	0.41
	2.5	0.98	0.24	1.0	0.44	0.98	0.26	1.0	0.41	0.98	0.27	1.0	0.41
	3.0	0.98	0.15	1.5	0.53	0.95	0.17	2.0	0.49	0.98	0.16	1.5	0.50
40	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.27	1.00	0.50	1.0	0.27
	1.5	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.27	1.00	0.50	1.0	0.27
	2.0	1.00	0.23	1.0	0.45	0.98	0.27	1.0	0.40	0.98	0.26	1.0	0.42
	2.5	1.00	0.23	1.0	0.45	0.98	0.27	1.0	0.40	0.98	0.26	1.0	0.42
	3.0	0.99	0.15	1.0	0.54	0.95	0.18	1.5	0.48	0.98	0.16	1.0	0.51

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
50	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.27	1.00	0.51	1.0	0.26
	1.5	1.00	0.43	1.0	0.30	1.00	0.48	1.0	0.27	1.00	0.51	1.0	0.26
	2.0	1.00	0.23	1.0	0.44	1.00	0.26	1.0	0.41	1.00	0.27	1.0	0.40
	2.5	1.00	0.23	1.0	0.44	1.00	0.26	1.0	0.41	1.00	0.27	1.0	0.40
	3.0	0.98	0.16	1.0	0.53	0.95	0.17	1.0	0.49	0.97	0.18	2.0	0.48
60	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.45	1.0	0.29	1.00	0.48	1.0	0.27	1.00	0.51	1.0	0.26
	1.5	1.00	0.45	1.0	0.29	1.00	0.48	1.0	0.27	1.00	0.51	1.0	0.26
	2.0	1.00	0.24	1.0	0.44	0.98	0.27	2.0	0.40	0.98	0.27	1.0	0.40
	2.5	1.00	0.24	1.0	0.44	0.98	0.27	2.0	0.40	0.98	0.27	1.0	0.40
	3.0	1.00	0.15	1.0	0.54	0.93	0.18	2.0	0.48	0.97	0.18	1.0	0.49
70	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.43	1.0	0.30	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27
	1.5	1.00	0.43	1.0	0.30	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27
	2.0	1.00	0.23	1.0	0.44	0.98	0.28	1.0	0.40	1.00	0.26	1.0	0.42
	2.5	1.00	0.23	1.0	0.44	0.98	0.28	1.0	0.40	1.00	0.26	1.0	0.42
	3.0	0.98	0.15	1.0	0.54	0.94	0.18	1.0	0.48	0.98	0.16	1.0	0.52
80	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.46	1.0	0.29	1.00	0.49	1.0	0.26	1.00	0.52	1.0	0.26
	1.5	1.00	0.46	1.0	0.29	1.00	0.49	1.0	0.26	1.00	0.52	1.0	0.26
	2.0	1.00	0.25	1.0	0.43	0.98	0.28	1.0	0.39	1.00	0.29	1.0	0.40
	2.5	1.00	0.25	1.0	0.43	0.98	0.28	1.0	0.39	1.00	0.29	1.0	0.40
	3.0	0.99	0.17	1.0	0.51	0.92	0.19	1.0	0.45	0.98	0.17	1.0	0.50
90	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	1.0	1.00	0.44	1.0	0.30	1.00	0.51	1.0	0.26	1.00	0.49	1.0	0.26
	1.5	1.00	0.44	1.0	0.30	1.00	0.51	1.0	0.26	1.00	0.49	1.0	0.26
	2.0	1.00	0.23	1.0	0.44	0.98	0.28	1.0	0.40	1.00	0.27	1.0	0.41
	2.5	1.00	0.23	1.0	0.44	0.98	0.28	1.0	0.40	1.00	0.27	1.0	0.41
	3.0	0.98	0.15	1.0	0.52	0.93	0.18	1.0	0.46	0.98	0.17	1.0	0.50
100	0.0	1.00	1.00	1.0	0.00	1.00	0.00	1.0	0.00	1.00	0.00	1.0	0.00
	0.5	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00	1.00	1.00	1.0	0.00
	1.0	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27	1.00	0.52	1.0	0.26
	1.5	1.00	0.49	1.0	0.27	1.00	0.48	1.0	0.27	1.00	0.52	1.0	0.26
	2.0	1.00	0.25	1.0	0.43	0.99	0.28	1.0	0.40	0.98	0.28	1.0	0.40
	2.5	1.00	0.25	1.0	0.43	0.99	0.28	1.0	0.40	0.98	0.28	1.0	0.40
	3.0	0.98	0.16	1.0	0.52	0.95	0.18	1.0	0.48	0.95	0.17	1.0	0.51

Table C.6: The median TPR, FPR, Latency (LT), and MCC of the T-Test classifier using a range of p values in detecting the variants of the P_{GR} from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}.

P_{PT}	p	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
5	0.40	0.85	0.64	1.0	0.12	0.75	0.57	1.5	0.11	0.82	0.66	1.0	0.09
	0.25	0.85	0.63	1.0	0.12	0.75	0.56	1.5	0.11	0.82	0.65	1.0	0.09
	0.10	0.85	0.35	1.0	0.27	0.75	0.33	1.5	0.25	0.82	0.39	1.0	0.23
	0.05	0.85	0.28	1.0	0.32	0.75	0.26	1.5	0.30	0.82	0.31	1.0	0.29
	0.025	0.85	0.22	1.0	0.38	0.75	0.20	1.5	0.35	0.82	0.24	1.0	0.35
	0.010	0.85	0.16	1.0	0.44	0.75	0.14	2.0	0.42	0.82	0.16	1.0	0.43
	0.005	0.83	0.12	1.0	0.51	0.75	0.11	2.0	0.47	0.82	0.13	1.0	0.48
	0.0005	0.83	0.06	2.0	0.67	0.73	0.05	3.0	0.59	0.78	0.06	2.0	0.62
10	0.40	0.87	0.65	1.0	0.13	0.80	0.58	1.0	0.13	0.86	0.66	1.0	0.11
	0.25	0.87	0.64	1.0	0.13	0.80	0.57	1.0	0.14	0.86	0.65	1.0	0.11
	0.10	0.87	0.32	1.0	0.31	0.80	0.33	1.0	0.27	0.86	0.37	1.0	0.26
	0.05	0.87	0.25	1.0	0.36	0.80	0.25	1.0	0.33	0.86	0.29	1.0	0.32
	0.025	0.87	0.19	1.0	0.43	0.80	0.19	1.0	0.39	0.86	0.23	1.0	0.37
	0.010	0.87	0.14	1.0	0.49	0.80	0.14	1.5	0.46	0.86	0.16	1.0	0.45
	0.005	0.87	0.11	1.0	0.54	0.80	0.11	1.5	0.52	0.85	0.12	1.0	0.51
	0.0005	0.87	0.05	1.0	0.70	0.80	0.05	2.0	0.65	0.83	0.06	2.0	0.67
20	0.40	0.88	0.64	1.0	0.14	0.82	0.58	1.0	0.14	0.87	0.68	1.0	0.10
	0.25	0.88	0.63	1.0	0.14	0.82	0.58	1.0	0.15	0.87	0.67	1.0	0.11
	0.10	0.88	0.31	1.0	0.32	0.82	0.31	1.0	0.29	0.87	0.37	1.0	0.26
	0.05	0.88	0.23	1.0	0.39	0.82	0.24	1.0	0.35	0.87	0.29	1.0	0.33
	0.025	0.88	0.19	1.0	0.44	0.82	0.19	1.0	0.41	0.87	0.23	1.0	0.39
	0.010	0.88	0.14	1.0	0.51	0.82	0.13	1.0	0.48	0.87	0.16	1.0	0.46
	0.005	0.88	0.11	1.0	0.56	0.82	0.10	1.0	0.54	0.87	0.13	1.0	0.52
	0.0005	0.88	0.05	1.0	0.70	0.81	0.04	1.0	0.68	0.86	0.06	1.0	0.66
30	0.40	0.88	0.65	1.0	0.13	0.82	0.57	1.0	0.14	0.87	0.67	1.0	0.11
	0.25	0.88	0.64	1.0	0.14	0.82	0.57	1.0	0.15	0.87	0.66	1.0	0.11
	0.10	0.88	0.31	1.0	0.32	0.82	0.29	1.0	0.31	0.87	0.37	1.0	0.27
	0.05	0.88	0.24	1.0	0.39	0.82	0.23	1.0	0.37	0.87	0.28	1.0	0.33
	0.025	0.88	0.19	1.0	0.44	0.82	0.18	1.0	0.42	0.87	0.21	1.0	0.39
	0.010	0.88	0.13	1.0	0.51	0.82	0.13	1.0	0.50	0.87	0.15	1.0	0.46
	0.005	0.88	0.11	1.0	0.57	0.82	0.10	1.0	0.54	0.87	0.12	1.0	0.51
	0.0005	0.88	0.05	1.0	0.70	0.82	0.05	1.0	0.67	0.86	0.06	1.0	0.66
40	0.40	0.90	0.64	1.0	0.15	0.79	0.58	1.0	0.12	0.85	0.67	1.0	0.11
	0.25	0.90	0.64	1.0	0.15	0.79	0.58	1.0	0.13	0.85	0.66	1.0	0.12
	0.10	0.90	0.30	1.0	0.35	0.79	0.30	1.0	0.28	0.85	0.36	1.0	0.27
	0.05	0.90	0.22	1.0	0.41	0.79	0.23	1.0	0.33	0.85	0.28	1.0	0.33
	0.025	0.90	0.17	1.0	0.46	0.79	0.19	1.0	0.39	0.85	0.22	1.0	0.38
	0.010	0.90	0.13	1.0	0.53	0.79	0.13	1.0	0.45	0.85	0.15	1.0	0.45
	0.005	0.90	0.10	1.0	0.58	0.79	0.10	1.0	0.51	0.85	0.13	1.0	0.51
	0.0005	0.90	0.05	1.0	0.71	0.79	0.05	1.0	0.64	0.85	0.06	1.0	0.66

P_{PT}	p	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
50	0.40	0.90	0.64	1.0	0.15	0.82	0.58	1.0	0.13	0.83	0.66	1.0	0.09
	0.25	0.90	0.63	1.0	0.15	0.82	0.57	1.0	0.14	0.83	0.66	1.0	0.10
	0.10	0.90	0.29	1.0	0.35	0.82	0.29	1.0	0.29	0.83	0.38	1.0	0.26
	0.05	0.90	0.23	1.0	0.40	0.82	0.23	1.0	0.35	0.83	0.29	1.0	0.31
	0.025	0.90	0.18	1.0	0.46	0.82	0.18	1.0	0.40	0.83	0.22	1.0	0.37
	0.010	0.90	0.13	1.0	0.53	0.82	0.13	1.0	0.47	0.83	0.16	1.0	0.45
	0.005	0.90	0.10	1.0	0.58	0.82	0.10	1.0	0.52	0.83	0.13	1.0	0.50
	0.0005	0.90	0.05	1.0	0.72	0.82	0.05	1.0	0.65	0.83	0.06	1.0	0.66
60	0.40	0.90	0.65	1.0	0.14	0.78	0.57	1.0	0.11	0.84	0.67	1.0	0.10
	0.25	0.90	0.65	1.0	0.15	0.78	0.56	1.0	0.12	0.84	0.66	1.0	0.10
	0.10	0.90	0.32	1.0	0.32	0.78	0.31	1.0	0.27	0.84	0.37	1.0	0.26
	0.05	0.90	0.24	1.0	0.38	0.78	0.23	1.0	0.32	0.84	0.29	1.0 5	0.32
	0.025	0.90	0.19	1.0	0.44	0.78	0.18	1.0	0.38	0.84	0.23	1.0	0.37
	0.010	0.90	0.14	1.0	0.51	0.78	0.13	1.0	0.46	0.84	0.16	1.0	0.44
	0.005	0.90	0.11	1.0	0.56	0.78	0.10	1.0	0.51	0.84	0.12	1.0	0.50
	0.0005	0.89	0.05	1.0	0.69	0.78	0.05	1.0	0.64	0.84	0.06	1.0	0.65
70	0.40	0.88	0.64	1.0	0.14	0.79	0.60	1.0	0.13	0.85	0.66	1.0	0.10
	0.25	0.88	0.63	1.0	0.14	0.79	0.58	1.0	0.13	0.85	0.65	1.0	0.11
	0.10	0.88	0.30	1.0	0.34	0.79	0.32	1.0	0.27	0.85	0.35	1.0	0.27
	0.05	0.88	0.23	1.0	0.41	0.79	0.25	1.0	0.34	0.85	0.28	1.0	0.32
	0.025	0.88	0.18	1.0	0.46	0.79	0.19	1.0	0.39	0.85	0.22	1.0	0.38
	0.010	0.88	0.13	1.0	0.53	0.79	0.14	1.0	0.46	0.85	0.16	1.0	0.46
	0.005	0.88	0.10	1.0	0.59	0.79	0.10	1.0	0.51	0.85	0.12	1.0	0.51
	0.0005	0.88	0.05	1.0	0.71	0.79	0.05	1.0	0.65	0.85	0.06	1.0	0.66
80	0.40	0.88	0.65	1.0	0.14	0.77	0.58	1.0	0.10	0.84	0.67	1.0	0.09
	0.25	0.88	0.64	1.0	0.14	0.77	0.58	1.0	0.10	0.84	0.66	1.0	0.10
	0.10	0.88	0.33	1.0	0.32	0.77	0.32	1.0	0.25	0.84	0.39	1.0	0.26
	0.05	0.88	0.26	1.0	0.37	0.77	0.26	1.0	0.29	0.84	0.30	1.0	0.31
	0.025	0.88	0.21	1.0	0.43	0.77	0.20	1.0	0.34	0.84	0.24	1.0	0.36
	0.010	0.88	0.15	1.0	0.50	0.77	0.15	1.0	0.41	0.84	0.17	1.0	0.44
	0.005	0.88	0.11	1.0	0.54	0.77	0.11	1.0	0.48	0.84	0.13	1.0	0.50
	0.0005	0.88	0.05	1.0	0.69	0.77	0.06	1.0	0.60	0.83	0.06	1.0	0.64
90	0.40	0.89	0.63	1.0	0.14	0.77	0.58	1.0	0.11	0.87	0.67	1.0	0.10
	0.25	0.89	0.62	1.0	0.15	0.77	0.57	1.0	0.12	0.87	0.65	1.0	0.10
	0.10	0.89	0.30	1.0	0.34	0.77	0.32	1.0	0.28	0.87	0.36	1.0	0.27
	0.05	0.89	0.23	1.0	0.40	0.77	0.25	1.0	0.33	0.87	0.28	1.0	0.33
	0.025	0.89	0.18	1.0	0.45	0.77	0.20	1.0	0.37	0.87	0.23	1.0	0.38
	0.010	0.89	0.13	1.0	0.52	0.77	0.14	1.0	0.45	0.87	0.16	1.0	0.46
	0.005	0.89	0.11	1.0	0.57	0.77	0.11	1.0	0.50	0.87	0.12	1.0	0.52
	0.0005	0.89	0.05	1.0	0.69	0.77	0.06	1.0	0.63	0.86	0.06	1.0	0.65
100	0.40	0.87	0.65	1.0	0.13	0.80	0.58	1.0	0.11	0.82	0.67	1.0	0.08
	0.25	0.87	0.64	1.0	0.13	0.80	0.58	1.0	0.12	0.82	0.66	1.0	0.09
	0.10	0.87	0.35	1.0	0.29	0.80	0.32	1.0	0.27	0.82	0.39	1.0	0.24
	0.05	0.87	0.27	1.0	0.35	0.80	0.25	1.0	0.32	0.82	0.30	1.0	0.29
	0.025	0.87	0.22	1.0	0.40	0.80	0.20	1.0	0.38	0.82	0.23	1.0	0.35
	0.010	0.87	0.16	1.0	0.46	0.80	0.14	1.0	0.47	0.82	0.17	1.0	0.43
	0.005	0.87	0.13	1.0	0.50	0.80	0.10	1.0	0.52	0.82	0.12	1.0	0.48
	0.0005	0.87	0.06	1.0	0.68	0.80	0.05	1.0	0.65	0.82	0.06	1.0	0.63

Table C.7: The median TPR, FPR, Latency (LT), and MCC of the Quartile-based classifier using a range of k values in detecting the variants of the P_{GR} from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}.

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	5 TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
5	0.0	0.95	0.40	1.5	0.30	0.94	0.41	1.0	0.27	0.95	0.43	1.0	0.27
	0.5	0.93	0.22	2.0	0.43	0.92	0.22	2.0	0.41	0.93	0.23	2.0	0.40
	1.0	0.85	0.16	3.0	0.45	0.78	0.16	2.5	0.43	0.83	0.17	2.5	0.42
	1.5	0.82	0.10	4.0	0.50	1.00	0.10	3.0	0.58	0.77	0.11	3.0	0.45
	2.0	0.80	0.09	5.0	0.52	0.71	0.09	3.5	0.51	0.75	0.09	4.0	0.48
	2.5	0.78	0.07	6.0	0.55	0.71	0.08	3.5	0.53	0.72	0.08	4.0	0.50
	3.0	0.78	0.07	6.0	0.57	0.70	0.07	4.5	0.53	0.70	0.07	5.5	0.50
10	0.0	0.97	0.39	1.5	0.31	0.97	0.41	1.0	0.30	0.97	0.41	2.0	0.30
	0.5	0.97	0.21	2.0	0.44	0.94	0.21	2.0	0.45	0.93	0.21	2.0	0.43
	1.0	0.88	0.15	2.0	0.48	0.80	0.16	2.5	0.44	0.85	0.16	3.0	0.43
	1.5	0.83	0.10	2.0	0.55	0.74	0.10	3.0	0.49	0.80	0.11	3.0	0.50
	2.0	0.83	0.09	2.5	0.57	0.74	0.08	3.5	0.52	0.78	0.09	3.0	0.52
	2.5	0.83	0.08	3.0	0.59	0.74	0.07	4.0	0.54	0.78	0.08	3.0	0.54
	3.0	0.83	0.07	3.5	0.60	0.74	0.06	4.0	0.55	0.78	0.08	4.0	0.55
20	0.0	0.98	0.38	1.0	0.32	0.98	0.40	2.0	0.30	0.98	0.41	1.0	0.31
	0.5	0.98	0.20	1.5	0.47	0.97	0.20	2.0	0.46	0.98	0.21	2.0	0.45
	1.0	0.90	0.15	2.0	0.49	0.80	0.14	2.0	0.42	0.88	0.15	2.0	0.48
	1.5	0.98	0.20	1.5	0.47	0.97	0.20	2.0	0.46	0.98	0.21	2.0	0.45
	2.0	0.85	0.08	2.0	0.59	0.74	0.08	2.0	0.51	0.82	0.08	3.0	0.56
	2.5	0.85	0.07	2.0	0.61	0.74	0.07	2.0	0.54	0.82	0.07	3.0	0.59
	3.0	0.85	0.07	2.0	0.62	0.73	0.06	2.5	0.55	0.82	0.07	3.0	0.60
30	0.0	0.99	0.39	1.0	0.32	1.00	0.40	1.0	0.32	0.99	0.40	1.0	0.31
	0.5	0.98	0.20	1.0	0.47	0.98	0.20	1.0	0.47	0.98	0.21	1.5	0.46
	1.0	0.91	0.15	1.0	0.50	0.82	0.15	2.0	0.44	0.90	0.15	1.5	0.48
	1.5	0.88	0.10	2.0	0.56	0.75	0.10	2.0	0.50	0.82	0.10	2.0	0.52
	2.0	0.87	0.09	2.0	0.59	0.75	0.08	2.0	0.52	0.82	0.08	2.0	0.55
	2.5	0.87	0.08	2.0	0.61	0.74	0.08	2.0	0.54	0.82	0.07	2.0	0.59
	3.0	0.87	0.07	2.0	0.62	0.74	0.06	2.0	0.55	0.82	0.06	2.0	0.61
40	0.0	1.00	0.39	1.0	0.32	1.00	0.41	1.0	0.32	1.00	0.39	1.0	0.32
	0.5	0.98	0.21	1.0	0.47	1.00	0.20	1.0	0.48	1.00	0.20	1.0	0.46
	1.0	0.93	0.15	1.0	0.49	0.85	0.15	1.0	0.47	0.87	0.15	1.0	0.47
	1.5	0.88	0.10	1.0	0.55	0.77	0.09	1.5	0.52	0.83	0.10	1.0	0.53
	2.0	0.88	0.09	1.0	0.57	0.77	0.08	1.5	0.54	0.82	0.09	1.0	0.55
	2.5	0.87	0.08	2.0	0.60	0.76	0.07	2.0	0.56	0.81	0.08	1.5	0.57
	3.0	0.87	0.07	2.0	0.61	0.76	0.07	2.0	0.57	0.80	0.07	2.0	0.59

P_{PT}	k	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
50	0.0	1.00	0.37	1.0	0.34	1.00	0.39	1.0	0.31	1.00	0.40	1.0	0.31
	0.5	1.00	0.19	1.0	0.49	0.99	0.20	1.0	0.47	0.98	0.21	1.0	0.45
	1.0	0.93	0.14	1.0	0.52	0.83	0.14	1.0	0.45	0.85	0.16	1.5	0.44
	1.5	0.89	0.10	1.0	0.58	0.77	0.10	2.0	0.53	0.79	0.10	1.5	0.48
	2.0	0.89	0.09	1.0	0.60	0.77	0.08	2.0	0.55	0.78	0.09	2.0	0.52
	2.5	0.88	0.07	2.0	0.62	0.77	0.07	2.0	0.58	0.78	0.07	2.0	0.55
	3.0	0.88	0.06	2.0	0.63	0.77	0.06	2.0	0.59	0.77	0.07	2.0	0.56
60	0.0	1.00	0.38	1.0	0.33	1.00	0.39	1.0	0.32	1.00	0.41	1.0	0.31
	0.5	1.00	0.21	1.0	0.47	1.00	0.21	1.0	0.47	1.00	0.22	1.0	0.45
	1.0	0.92	0.16	1.0	0.49	0.80	0.15	1.0	0.43	0.88	0.16	1.0	0.46
	1.5	0.87	0.10	1.0	0.56	0.68	0.10	1.0	0.48	0.82	0.10	1.0	0.50
	2.0	0.87	0.09	1.0	0.59	0.68	0.08	1.0	0.50	0.82	0.08	1.0	0.52
	2.5	0.86	0.08	1.0	0.61	0.68	0.07	1.5	0.52	0.82	0.07	1.0	0.55
	3.0	0.86	0.07	1.0	0.63	0.68	0.06	2.0	0.54	0.82	0.07	1.0	0.57
70	0.0	1.00	0.39	1.0	0.32	1.00	0.40	1.0	0.31	1.00	0.41	1.0	0.31
	0.5	1.00	0.21	1.0	0.47	1.00	0.20	1.0	0.48	1.00	0.22	1.0	0.44
	1.0	0.93	0.16	1.0	0.50	0.80	0.16	1.0	0.45	0.88	0.17	1.0	0.45
	1.5	0.89	0.10	1.0	0.57	0.73	0.10	1.0	0.50	0.80	0.11	1.0	0.49
	2.0	0.89	0.09	1.5	0.59	0.73	0.08	1.0	0.53	0.80	0.10	1.0	0.52
	2.5	0.89	0.08	2.0	0.62	0.73	0.07	1.5	0.55	0.80	0.08	1.0	0.54
	3.0	0.89	0.08	2.0	0.63	0.73	0.06	2.0	0.56	0.80	0.08	1.0	0.55
80	0.0	1.00	0.39	1.0	0.32	1.00	0.40	1.0	0.31	1.00	0.42	1.0	0.31
	0.5	1.00	0.21	1.0	0.47	1.00	0.21	1.0	0.46	1.00	0.23	1.0	0.45
	1.0	0.90	0.16	1.0	0.47	0.78	0.15	1.0	0.40	0.87	0.18	1.0	0.45
	1.5	0.80	0.10	1.0	0.51	0.70	0.10	1.0	0.42	0.80	0.12	1.0	0.50
	2.0	0.80	0.09	1.0	0.54	0.70	0.08	1.0	0.44	0.80	0.10	1.0	0.53
	2.5	0.80	0.08	2.0	0.56	0.70	0.07	1.0	0.47	0.79	0.09	1.0	0.55
	3.0	0.80	0.08	2.0	0.57	0.70	0.07	1.0	0.48	0.79	0.08	1.0	0.576
90	0.0	1.00	0.39	1.0	0.32	1.00	0.41	1.0	0.31	1.00	0.40	1.0	0.31
	0.5	1.00	0.21	1.0	0.47	1.00	0.21	1.0	0.47	1.00	0.22	1.0	0.45
	1.0	0.91	0.15	1.0	0.49	0.81	0.16	1.0	0.43	0.85	0.16	1.0	0.43
	1.5	0.85	0.10	1.0	0.55	0.73	0.10	1.0	0.48	0.79	0.11	1.0	0.50
	2.0	0.85	0.09	1.0	0.57	0.73	0.09	1.0	0.50	0.79	0.09	1.0	0.51
	2.5	0.84	0.08	1.0	0.60	0.73	0.07	1.0	0.52	0.79	0.08	1.0	0.53
	3.0	0.84	0.08	1.0	0.61	0.73	0.07	1.0	0.54	0.78	0.08	1.0	0.54
100	0.0	1.00	0.41	1.0	0.31	1.00	0.41	1.0	0.31	1.00	0.42	1.0	0.30
	0.5	1.00	0.22	1.0	0.45	1.00	0.21	1.0	0.47	1.00	0.22	1.0	0.45
	1.0	0.86	0.16	1.0	0.45	0.85	0.15	1.0	0.46	0.85	0.16	1.0	0.44
	1.5	0.81	0.11	1.0	0.52	0.77	0.10	1.0	0.49	0.76	0.11	1.0	0.48
	2.0	0.81	0.09	1.0	0.55	0.77	0.08	1.0	0.53	0.76	0.10	1.0	0.50
	2.5	0.80	0.08	1.0	0.55	0.77	0.07	1.0	0.55	0.75	0.08	1.0	0.53
	3.0	0.80	0.08	1.0	0.57	0.77	0.06	1.0	0.56	0.75	0.08	1.0	0.55

Table C.8: The median TPR, FPR, Latency (LT), and MCC of the Q-Test classifier using a range of α values in detecting the variants of the P_{GR} from 5 $\times 10^{-5}$ m.s^{-2} to 100 $\times 10^{-5}$ m.s^{-2}.

P_{PT}	α	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
5	80	0.92	0.18	3.0	0.47	0.93	0.20	2.0	0.43	0.88	0.18	3.0	0.43
	90	0.89	0.13	3.5	0.51	0.88	0.15	2.0	0.47	0.83	0.13	4.0	0.48
	95	0.85	0.11	3.5	0.53	0.83	0.13	4.0	0.48	0.78	0.11	6.0	0.49
	96	0.85	0.10	3.5	0.54	0.81	0.13	5.0	0.48	0.75	0.11	7.0	0.48
	99	0.76	0.08	5.0	0.52	0.70	0.11	5.5	0.44	0.68	0.08	10.0	0.48
10	80	0.96	0.15	3.0	0.53	0.97	0.19	2.0	0.46	0.95	0.18	2.5	0.48
	90	0.93	0.11	3.0	0.58	0.93	0.15	3.0	0.50	0.93	0.13	3.0	0.54
	95	0.92	0.09	4.0	0.61	0.89	0.12	3.5	0.51	0.87	0.11	4.0	0.55
	96	0.90	0.09	4.0	0.61	0.87	0.12	3.5	0.51	0.87	0.10	5.0	0.56
	99	0.84	0.07	4.5	0.61	0.74	0.11	4.0	0.49	0.78	0.08	5.5	0.55
20	80	0.98	0.15	2.0	0.53	0.98	0.18	2.0	0.48	0.97	0.18	2.0	0.50
	90	0.97	0.11	2.0	0.61	0.95	0.13	2.0	0.54	0.95	0.13	3.0	0.57
	95	0.93	0.09	2.0	0.62	0.92	0.11	3.0	0.55	0.93	0.10	3.0	0.60
	96	0.93	0.08	2.0	0.63	0.89	0.10	3.0	0.55	0.92	0.10	3.0	0.60
	99	0.88	0.07	3.5	0.63	0.78	0.10	3.5	0.49	0.83	0.07	3.5	0.60
30	80	0.98	0.16	2.0	0.53	0.98	0.18	2.0	0.49	0.98	0.17	1.5	0.52
	90	0.98	0.12	2.0	0.58	0.98	0.13	2.0	0.54	0.97	0.13	2.0	0.57
	95	0.96	0.09	2.0	0.62	0.92	0.11	2.0	0.56	0.93	0.10	2.0	0.60
	96	0.95	0.09	2.0	0.62	0.92	0.10	2.0	0.56	0.93	0.10	2.0	0.61
	99	0.88	0.07	3.0	0.63	0.80	0.09	3.0	0.55	0.87	0.08	3.0	0.60
40	80	0.98	0.15	2.0	0.54	0.98	0.18	2.0	0.49	0.98	0.17	1.5	0.52
	90	0.98	0.11	2.0	0.60	0.97	0.13	2.0	0.55	0.97	0.13	2.0	0.58
	95	0.97	0.09	2.0	0.64	0.92	0.11	2.0	0.55	0.93	0.10	2.0	0.61
	96	0.96	0.09	2.0	0.64	0.91	0.11	2.0	0.55	0.93	0.10	2.0	0.61
	99	0.92	0.07	2.0	0.66	0.82	0.10	2.0	0.53	0.85	0.07	2.0	0.60
50	80	1.00	0.15	1.0	0.54	1.00	0.16	1.0	0.50	0.98	0.18	1.5	0.51
	90	0.98	0.11	2.0	0.59	0.98	0.12	1.0	0.56	0.97	0.13	2.0	0.56
	95	0.97	0.09	2.0	0.63	0.93	0.10	1.5	0.57	0.93	0.10	2.0	0.59
	96	0.95	0.09	2.0	0.64	0.92	0.10	1.5	0.57	0.90	0.10	2.0	0.59
	99	0.92	0.07	2.0	0.64	0.81	0.09	2.0	0.53	0.83	0.08	2.0	0.58
60	80	0.98	0.15	1.0	0.54	0.98	0.17	1.0	0.48	0.99	0.18	1.0	0.51
	90	0.98	0.11	1.0	0.61	0.97	0.13	1.0	0.53	0.98	0.13	1.0	0.57
	95	0.97	0.09	2.0	0.62	0.93	0.11	1.0	0.54	0.93	0.11	1.0	0.58
	96	0.95	0.09	2.0	0.63	0.90	0.11	1.0	0.55	0.93	0.10	1.5	0.59
	99	0.90	0.07	2.0	0.64	0.78	0.10	2.0	0.50	0.85	0.08	2.0	0.59

P_{PT}	α	CST				V_{OPR}				V_{ODS}			
		TPR	FPR	LT	MCC	TPR	FPR	LT	MCC	TPR	FPR	LT	MCC
70	80	1.00	0.14	1.0	0.55	1.00	0.19	1.0	0.49	1.00	0.17	1.0	0.51
	90	0.98	0.10	1.0	0.63	0.98	0.15	2.0	0.54	0.98	0.12	1.0	0.59
	95	0.98	0.09	1.5	0.65	0.92	0.12	2.0	0.55	0.94	0.10	1.0	0.61
	96	0.97	0.08	1.5	0.66	0.90	0.12	2.0	0.54	0.93	0.09	1.5	0.61
	99	0.93	0.06	2.0	0.68	0.80	0.10	2.0	0.50	0.85	0.07	2.0	0.59
80	80	1.00	0.16	1.0	0.53	1.00	0.19	1.0	0.47	1.00	0.18	1.0	0.52
	90	0.99	0.12	1.0	0.59	0.98	0.14	1.0	0.54	0.98	0.14	1.0	0.57
	95	0.97	0.10	1.5	0.61	0.93	0.11	1.0	0.54	0.95	0.11	1.0	0.60
	96	0.95	0.10	1.5	0.61	0.91	0.11	1.0	0.54	0.93	0.10	1.0	0.60
	99	0.89	0.08	2.0	0.61	0.74	0.10	2.0	0.46	0.83	0.08	2.0	0.60
90	80	1.00	0.15	1.0	0.53	1.00	0.18	1.0	0.50	1.00	0.17	1.0	0.51
	90	0.98	0.11	1.0	0.59	0.98	0.14	1.0	0.54	0.97	0.13	1.0	0.57
	95	0.97	0.10	1.0	0.61	0.93	0.12	1.0	0.55	0.93	0.10	1.0	0.60
	96	0.95	0.10	1.0	0.62	0.93	0.12	1.0	0.55	0.93	0.10	1.0	0.60
	99	0.90	0.08	2.0	0.61	0.80	0.10	2.0	0.52	0.83	0.08	1.5	0.58
100	80	1.00	0.17	1.0	0.52	1.00	0.18	1.0	0.50	1.00	0.18	1.0	0.51
	90	1.00	0.12	1.0	0.60	0.97	0.13	1.0	0.54	0.98	0.13	1.0	0.57
	95	0.97	0.10	1.0	0.63	0.94	0.11	1.0	0.57	0.93	0.10	1.0	0.60
	96	0.95	0.09	1.0	0.63	0.92	0.10	1.0	0.56	0.93	0.10	1.0	0.60
	99	0.87	0.07	1.0	0.61	0.83	0.10	2.0	0.54	0.85	0.08	1.0	0.61