Immune-inspired fault diagnosis for a robotic system

Ran Bi

A thesis for the degree of Doctor of Philosophy

The University of York

Department of Electronics

January 2012

Abstract

To achieve fully autonomous systems, *fault tolerance* is often employed. *Fault tolerance* is the ability to continue operation in the presence of faults. *Fault diagnosis* is an essential component of *fault tolerance*, especially for autonomous robotics. It is the process of determining as much information as possible about the fault, especially the origin of the fault. However, a real time fault diagnosis for resource limited robotic systems has proposed a new set of challenges, such as its complexity and efficiency, which traditional methods will find difficult to meet. This has led the work to seek inspiration from the immune system, where an effective and efficient fault diagnosis solution has been provided for thousands of years. This thesis presents a novel immune-inspired on-line fault diagnosis algorithm for robotic systems and includes the first application of that Artificial Immune System to robot fault diagnosis.

Contents

List of Tables

List of Figures

Acknowledgements

This thesis is dedicated to my the other half, Miao, for her tremendous love and support which has enabled me to pursue my dreams.

I would have never made this far, were it not for the support of some fantastic people: I would like to start by acknowledging the help and support I received from the Intelligent Systems Research Group at University of York. I would like to thank Dr. Martin Trefzer and Dr. James Walker for their tips on Linux and Python. Thank goes to Dr. Liu Yang, James Hilder, Omer Qadir, Pitiwut Teerakittikul, Dr. Maizura Mokhtar and Dr. Tuze Kuyucu for countless informal chats. I would like to thank all the guys and girl from our office, Luis Rodrguez, Liu Xiaohu, Piero Conca, Colin Bonney, Antonio Zamorano and Yuan Zhang for a quiet and pleasant study environment. And I would also like to thank the SYMBRION project for those project meetings in Europe.

And finally, I would like to thank my supervisors Prof. Andy Tyrrell and Prof. Jon Timmis for their encouragement, support, direction and attention over the past four years.

Author's declaration

I declare that this thesis entitled *"Immune inspired fault diagnosis for a robotic system"* is the result of my own research except as cited in the references. Selected aspects of the research described in this thesis, as well as some earlier work has been documented and published in [28].

Chapter 1

Introduction

Contents

To achieve fully autonomous systems, *fault tolerance* is often employed. *Fault tolerance* is the ability to continue operation in the presence of faults. Many applications of mobile robots require the robots to be able to continuously and autonomously function for a period of time, such as space programmes (e.g.[3][4]), and deep water exploration[5][6]. Autonomous functioning often means that a robot must function on its own with little or no human involvement. Because of the application requirement, such as space exploration, often the human commands would be less effective due to unknown environment changes or time delay. However, experience has shown that even carefully designed and tested robots may encounter faults [32]. One of the reasons for this is that components degrade over time. Therefore, to achieve autonomy, a mobile robot needs to be able to continue functioning in the presence of faults.

Fault diagnosis is an essential component of *fault tolerance*, especially for autonomous robotics. It is the process of determining as much information as possible about the fault, especially the origin of the fault. There have been proposed many methods for fault diagnosis for the past half a century, such as statistical methods[63][34][78] and model based methods[69][94]. However, a real time fault diagnosis for resource limited robotic systems implies a new set of challenges, such as complexity and efficiency, which traditional methods will find difficult to meet. A resource limited robot is often small and mobile, which is often equipped with less computational power to achieve its long term task. It is often too expensive computationally or impossible to remodel or retrain in real time. Hence, a real time fault diagnosis for robotic system has to be lightweight in terms of computational consumption and also delivers reasonable result. Many robots often work in unknown or partially unknown environments and therefore some learning in the diagnosis system would be beneficial. This has led this work to seek inspiration from the immune system, where an effective and efficient fault diagnosis solution has been provided for thousands of years.

The analogy between fault tolerance and the immune system was first expressed in [23]. The notion of diagnosis within the immune system was proposed in "danger theory"[64], where the Antigen Presenting Cell (APC) was emphasised specially dendritic cell plays vital role in triggering an immune response. APC recognises the signalling molecules released by the death of a cell. This has led other research[50] to abstract a dendritic cell model and derive an algorithm for anomaly detection, named the Dendritic Cell Algorithm (DCA). Based upon DCA, certain aspects of immunology were added, proved and formed the work of this thesis.

This thesis presents a novel immune inspired on-line fault diagnosis algorithm for robotic systems.

1.1 Contribution

To better understand the contributions of this thesis, it is perhaps best to first define a hypothesis by which the contributions can be tested. For this thesis the hypothesis can be defined as:

> *"An immune-inspired system can be successfully deployed in a resource constrained robotic system to diagnose the cause of faults, in an on-line manner and accurately with reasonable response time."*

In order to validate this hypothesis, this thesis presents the following work:

- *Challenges:* Fault diagnosis is introduced and the unique challenges for fault diagnosis for robotic systems are outlined.
- *A novel algorithm:* Immunology is revisited and additional plausible abstractions are described and argued. A novel immune inspired fault diagnosis algorithm is then presented.
- *Analysis:* A detailed qualitative and quantitative analysis of the new algorithm and its parameters are presented which identifies that this new algorithm is capable of real time diagnosis for robotic system.
- *A comparison:* A quantitative and statistical comparison between the new algorithm, Artificial Neural Network (ANN) and the Support Vector Machine (SVM) shows that the new algorithm is an effective diagnosis system in comparison to existing techniques and yet provides the benefits of being simpler to use with less computational requirement.

As described above this thesis proposes a novel immune-inspired fault diagnosis algorithm. This algorithm further extends the abstraction of dendritic cell model from [50] and is then applied to fault diagnosis. This thesis includes the first application of an AIS to robot fault diagnosis, which has since become an active area of research.

1.2 Thesis structure

This thesis is organised as following:

Chapter 2 introduces the field of fault diagnosis. As this is a very large topic area covering many different techniques, this chapter focuses only on those methods that are relevant to this thesis. The challenges for robot fault diagnosis is also presented.

Chapter 3 introduces the immune system fault diagnosis and Artificial Immune System. The the original Dendritic Cell Algorithm (the original DCA)[50] and the conceptual framework are also introduced.

Chapter 4 presents a novel fault diagnosis algorithm extended from the abstracted model in the original DCA[50]. This work includes justifications of the additional features from the original DCA and describes the implementation of these as a novel fault diagnosis algorithm.

Chapter 5 presents the result of two different approaches to implement the new algorithm. More importantly, it shows that the modifications from the the original DCA improves the algorithm's performance.

Chapter 6 looks in detail into the effects of the parameters of the new algorithm and suggests how these parameters affect the outcome of the algorithm.

Chapter 7 compares the performance of ANN, SVM and the new algorithm in fault diagnosis for robotic system.

Chapter 8 provides a summary of the work presented in the thesis and outlines the conclusions that can be drawn, with ideas for future work also being presented.

Chapter 2

Fault diagnosis

Contents

Figures

2.1 Introduction

The aim of this chapter is to present the challenges faced when developing fault diagnosis for a robotic system. *Fault diagnosis* is an essential component of *fault tolerance*, especially for autonomous robotics. *Fault tolerance* involves *error detection, diagnosis* and *recovery*, which afford the ability of a system to continue operation in the presence of *faults*[62]. Comprehensive reviews on fault diagnosis can be found in [59][90][91][89].While a variety definitions of around the term *fault tolerance* have been suggested, this thesis will use the set of definitions suggested by [62] and [59] as following:

Fault is an unpermitted deviation of at least one characteristic property of the system from the acceptable, usual, standard condition.

Fault tolerance allows a system to continue operation in the presence of faults.

Error detection detects an erroneous state within the system.

Fault diagnosis determines as much information as possible about the fault, such as the fault origin, magnitude, location and time of existence.

Recovery allows a system to overcome the fault.

There are many types of faults, in which the most common one would be *stuck-at fault*. *Stuck-at fault* can occur when components degrade over time or by environment. Some faults are grouped by the frequency of their existence, such as *periodic fault*, *permanent fault* and *transient* *fault.* A *periodic fault* occurs periodically, and is normally caused by rotation, such as wheel spin or gear rotation. A *permanent fault* is one off action, which once present will stay permanently. A *transient fault* is also one off action, however, only of short duration. *Transient faults* can be caused by sudden changes, and because it only lasts a few seconds or milliseconds usually it is expected a system will not react to it. If it appears repeatedly, it would be considered as a *periodic fault.* The robotic system used in this work is a *real time* system[96]. A *real time* system is that a system where any information processing activity within it has to respond to externally generated input stimuli within a finite and specified period. Therefore, the fault diagnostic system discussed here has to be a *real time* system.

Having explained engineering problems, in real time faults diagnosis for a robotic system, this chapter starts with a general description of how fault diagnosis is achieved and why it is only interested in classification methods in this study. It is followed by an introduction of fault diagnosis methods, in which Artificial Neural Network (ANN), Support Vector Machine (SVM) and K-nearest neighbour methods will be introduced. At the end, the challenges faced by this work will be summarised.

2.2 Engineering fault diagnosis

Fault diagnosis is an inverse process of fault propagation[59]. A fault in general influences events (effects), where events can manifest as an irregular behaviour of a robot due to the fault (cause). Events then influence error detection to flag the system as faulty. The irregular behaviour (event) is often observable or can be calculated from the deviation from the "normal" values. However, the cause of the event (fault) is often determined by systems internal physical properties. The underlying physical laws, are mostly not known in an analytical form, or are too complicated for calculations. For example, a robot is running round in a small circle when there are no obstacles in sight, whereas it should be walking forward in straight line. This is irregular behaviour and can be observed and calculated. However, there might be faults causing the problem, such as a stuck-at fault on one of the front sensors at a low value, thus, the robot is wrongly sensing an obstacle near by; or, it could be a wheel broken, in which case the robot can not move properly. The observed behaviour or the state of the system could be the same, but the cause or causes would be different. This presents a challenge, as the fault diagnosis implies the inversion of the causality. One cannot expect to reconstruct the cause-effect chain solely from measured data, because the causality is not reversible or the reversibility is ambiguous. [59] If one can ascertain the causality, fault diagnosis will become trivial. Otherwise, classification methods (in machine learning) are often applied and will be described in the next section.

Before introducing any learning methods, it is worth mentioning the term, *prior knowledge*, as there are many model based fault diagnosis approaches using prior knowledge widely used in engineering systems. Prior knowledge[75] refers to all available information about the problem. This could be user's experience; the cause-effect relations; or physical laws. Using prior knowledge, there were many examples of model based faults diagnosis which were reviewed in [60] and [91]. From a modelling perspective, fault diagnosis is achieved by acquiring accurate process models from prior knowledge. The model can be an equation or a set of equations representing the system. The approaches [48][46] reviewed in [60] were sharing a common ground, where a specific fault is targeted and the environment localised, although some were presented in a dynamic environment. The quality of a model and its diagnostic performance depends on prior knowledge. If this is only partial or insufficient, then a model based approach will not be appropriate. On the other hand, classification methods do not assume any forms of model and rely only on historic process data (training data). One could argue training data is a form of prior knowledge and indeed, the quality of it will also affect the diagnostic performance.

Partial knowledge of a robotic system used in this work is insufficient to create a model to achieve fault diagnosis. There are so many possibilities and the search space is effectively infinite. There are infinite combinations of the system state, as there are many sensors, actuators and other information which form the state of the system. It is not possible to calculate or model a fault, therefore, classification methods are considered here. They will be introduced in the next section and compared in Chapter 7.

2.3 Fault diagnosis methods

A survey of diagnosis methods[59] is shown in Figure 2.1. The methods that will be introduced are Artificial Neural Network (ANN), Support Vector Machine (SVM) and K-nearest neighbour. They are all classification methods and fall into artificial intelligence, approximation and density based method, respectively. These methods were selected to compare with the method proposed in this thesis, as they represent the classification category for fault diagnosis. All of them are well known and widely used in machine learning.

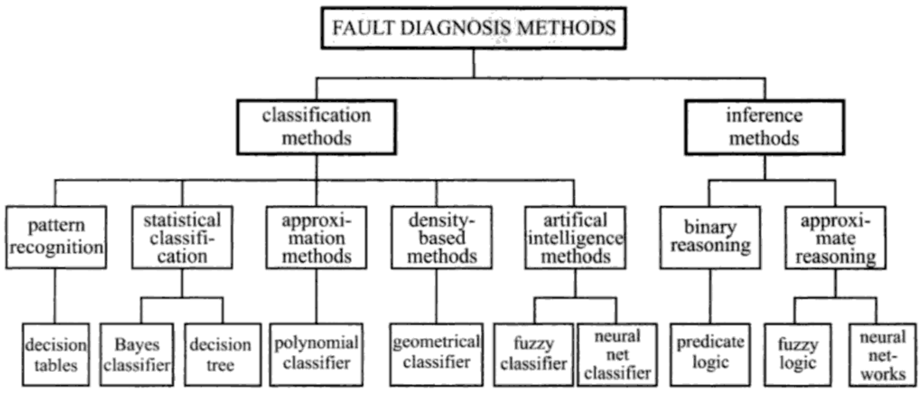


Figure 2.1: Fault diagnosis methods[59]

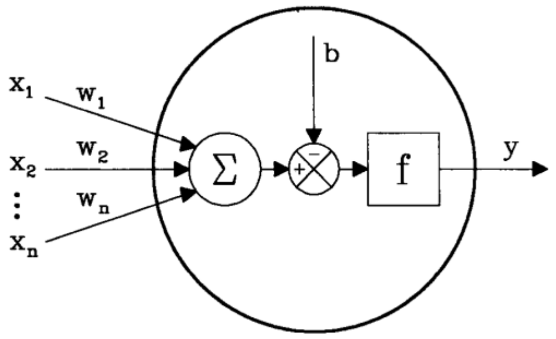


Figure 2.2: Schematic diagram of a processing element[22]

2.3.1 Artificial Neural network

There are many types of Artificial Neural networks (ANNs). Since the first neural model by [65] there have been developed hundreds of different models considered as ANNs. The differences in them might be the functions, the accepted values, the topology, the learning algorithms, etc. Also there are many hybrid models where each neuron has more properties than the ones are introduced here. This section presents only an ANN which is a multilayer perceptron network and learns using the backpropagation algorithm [70] for learning the appropriate weights, since it is one of the most common models used in ANNs, and many others are based on it.

The basic element of an ANN is the artificial neuron know as a "Pro-

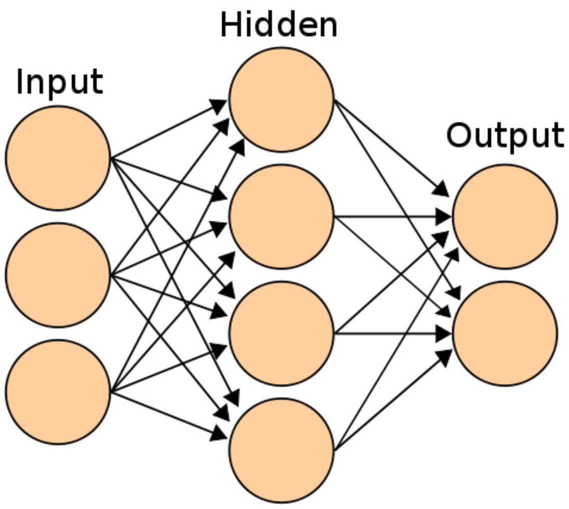


Figure 2.3: A feed forward ANN with 3 layers[10]

cessing Element" (PE). The schematic of a PE is shown in Figure 2.2. The PE output y is given by

$$y = s[(\sum_i w_i \cdot x_i) - b], \tag{2.1}$$

where x_i are the PE inputs and w_i are their weight, b is the offset (bias) and s is the transfer function. There are linear, threshold and sigmoid transfer functions. The transfer function of a neuron is chosen to have a number of properties which either enhance or simplify the network containing the neuron. [11] For instance, any multilayer perceptron using a linear transfer function has an equivalent single-layer network; a non-linear function is therefore necessary to gain the advantages of a multi-layer network. In general, if the output from ANN is required to be continuous, it is sigmoid:

$$s(h) = 1/[1 + exp(-h)]. \tag{2.2}$$

An ANN consists of a set of these PEs interconnected and organised in layers. The most common setting used is the feed-forward network.

It is made up of three or more layers, shown in Figure 2.3. Each layer consists of a number of nodes (PEs) and the interconnections are only between nodes of adjacent layers. The layer connected to inputs is called input layer; the layer connected to outputs is called output layer; the rest is called hidden layer. The number of nodes required for input and output layers depends on a specific application. The choice of hidden layer and hidden node number is debatable without an agreed guidance[47].

The ANN is capable of learning. The network is trained to produce the desired outputs on the basis of the inputs supplied to it. This is carried out through a learning algorithm, where the network's weights and offset will be modified towards the desired outputs until some 'stop' conditions. A stop condition normally refers to convergence. Once learning is complete, the network would not only produce the desired output for known inputs but also reasonably pleasing output for unknown inputs. This is under the assumption that the data trained is a representation of the whole data and the learning algorithm would "train" the network[22].

One of the commonly used learning algorithm is back-propagation[56]. It evolves over three phases:

1. Output formation: with an randomised initial weight and offset distribution ($k = 0$), the network $y_i(k)$ outputs, at the k-th step, are generated for a given set of inputs.

2. Error calculation:

   - For the output layer PE's:

   $$\varepsilon_i(k) = y_i(k) \cdot [1 - y_i(k)] \cdot [y_i^*(k) - y_i(k)] \tag{2.3}$$

   with $y_i^*(k)$ being the desired network outputs at the k-th step.

- For the PE's of other layers:

$$\varepsilon_i(k) = y_i(k) \cdot [1 - y_i(k)] \cdot [\sum_j w_{ij}(k) \cdot \varepsilon_j(k)] \tag{2.4}$$

  with $\varepsilon_j(k)$ being the errors at the k-th step for PEs of the immediately adjacent layer.

3. Backward error propagation: the errors thus determined are used to readjust the values of interconnections weights and the offset of each node according to

$$\begin{aligned} w_{ij}(k+1) &= w_{ij}(k) + \alpha \cdot \varepsilon_i(k) \cdot y_i(k) \\ b_{ij}(k+1) &= b_i(k) + \alpha \cdot \varepsilon_i(k) \end{aligned}$$

   where the coefficient α represents the learning speed.

Iterating through (1), (2) and (3), new input and output sets are supplied to the network by each iteration. Once the weight and offset value distribution reaches with the minimised errors in (2) and (3). In this condition, the network has learned and can be used on test data.

The ANN has been applied to fault diagnosis. Examples can be found in [26][98][92][73]. To some extent, the ANN has shown its capability in a dynamic environment. However, a large problem of the ANN is the difficult extrapolation[1] behaviour, as the data set in diagnostic applications are not always complete, argued by the author[59]. This creates a need for a diagnosis system to work outside the trained domain. This is especially problematic for ANN. A simple experiment of ANN within the robotic environment will be presented and compared in Chapter 7.

[1]In mathematics, *extrapolation* is the process of constructing new data points.[1]

2.3.2 Support Vector Machine

Support Vector Machine (SVM) is an example of two-class linear classifiers. It has been widely used in machine learning community, due to its high accuracy and the ability to deal with high-dimensional data. Some of its use on fault diagnosis can be found in [97][35][95]. Although SVM was first introduced[29] as linear classifiers, non-linear problems can be solved by applying different kernels, such as Polynomial and Gaussian. In this section, a description of linear SVM from [25] will be given and the effects of different kernels can be found in Appendix Figure G.1, G.2 and G.3.

The key concept required for defining a linear classifier is the dot product between two vectors, defined as $w^T x = \sum_i w_i x_i$. A linear classifier is based on a linear discriminant function of the form

$$f(x) = w^T x + b. \tag{2.5}$$

where w is the weight vector, and b is the bias. Consider the case $b = 0$ first. The set of points x such that $w^T x = 0$ are all points that are perpendicular to w and go through the origin. That is a line in two dimensions, a plane in three dimensions and more generally, a hyperplane. The bias b separates the hyperplane away from the origin. The hyperplane

$$x : f(x) = w^T x + b = 0 \tag{2.6}$$

divides the space into two. The boundary between regions classified as two classes is called the decision boundary.

If a clear distinctive hyperplane can not be drawn, so data cannot be separated into different classes, the notion of soft margin can be introduced. The soft margin method will choose a hyperplane that splits the examples as cleanly as possible, while still maximizing the distance to the nearest cleanly split data. It allows some data to be misclassified. A smaller value of soft margin allows for ignoring points close to the boundary and increases the margin[25], shown in Appendix G.

However, the imbalance data sets present a challenge for learning algorithms as noted in [33]. SVM is no exception. A good strategy for this, is to classify data belonging to majority classes. This will be used in Chapter 7, where significant imbalance can be found in the data sets, because most of the time the robot was not in contact with any obstacles. This creates duplicate data entries with all the sensors returning the maximum values. To improve SVM, the duplicate data entries had to be removed.

2.3.3 K-nearest neighbour

The k-nearest neighbour algorithm (k-NN)[12] is a method for classifying objects based on closest training examples in the feature space. It might be the simplest method in machine learning. The training data entries are vectors in a multidimensional feature space, each with a class label. The algorithm is trained by storing the feature vectors and class labels of the training data. When classifying test data, an unlabelled vector is assigned a label where k (user defined) training samples are nearest to it.

K-nearest neighbour is not considered to be suitable for this work, as data samples are imbalanced and some have one too many class relations. The basic "majority voting" classification is that in which the more frequent examples tend to dominate the prediction of the new vector, as they tend to come up in the k nearest neighbours when the neighbours are computed due to their large number.[36] To illustrate this issue, a print out of k-NN with test data is shown in Appendix I.

2.4 Summary

This chapter sets out to introduce the fault diagnosis. A general approach for fault diagnosis is described as the inverse process of fault propagation. It is a process of finding the cause (fault) of the effect (irregular behaviour). Then, the notion of prior knowledge was introduced as the available information on the causality related to the fault. However, for a robotic system the prior knowledge is often partial or inadequate to use model based methods. Hence, classification methods were considered and introduced. It is apparent that all the above presented a few challenges and they are summarised as below:

2.4.1 Challenges

Robotic challenges

- The robotics system used for this work is autonomous and real time, the fault diagnosis process has to be on-line and there is only limited resource, as in computational power. These have set the basic requirements for the fault diagnosis system: it has to be real time system, where it has to produce an output within a time limit; It has to respond quickly and accurately, otherwise it would not be of much use and it has to consume as little computational power as possible, such as memory and cpu time.

- The prior knowledge is partial. This implies the fault diagnosis system has to be able to cope with unseen faults or data

- The robotics environment is dynamic. This implies the fault diagnosis system has to be adaptable to changes, such as environment or the robot's internal changes (objective/ task changes).

Fault diagnosis challenges

- Data size and quality. A typical challenge for classification methods is the size of available data sets and how well they represent the whole system. Insufficient data will lead a classification method to perform undesirably. This will be further explored in Chapter 7, where a comparison is presented.

- The distinction between different faults. If faults are not distinctive, but somehow independent, it would cause any diagnosis system to become confused, illustrated by one example of a robot running round in circle illustrated previously.

The challenges stated above have suggested that "in a resource constrained robotic system to diagnose the cause of the fault, in an on-line manner and accurately" will not be an easy task. As always, if an engineering problem can not be easily solved, one will seek inspirations from biology, where there has been a solution for thousands of years, such as the immune system.

Chapter 3

Immune system and AIS

Contents

Figures

3.1 Introduction

The immune system is a complex biological system. It protects the body from infectious agents and the damage they cause, with a variety of interacting cellular and molecular elements. Its complexity and profound ability to protect is still not fully understood. However, properties emerging from the immune system have caught the attention of many engineers, in relation to topics such as real time capability, efficiency and self regulating. The immune system is constantly monitoring the states of the body, recognising any “non-self” invaders and trying to eliminate them. It is also adaptable to change, is able to remember what it has ‘seen’ and to react more quickly next time to a similar occurrence, for example, the use of vaccination. The immune system is efficient as it utilises the cells and molecules to combat an infection, but only when necessary. Once the threat is eliminated, the immune system will regulate itself to prevent any overreaction. The immune system indeed provides a rich inspirational ground, where many aspects are desirable to an engineering system.

Artificial Immune System (AIS) is a discipline inspired from the immune system. Many algorithms have been developed with some success, such as negative selection, clonal algorithm and immune networks. However, many of them have been said to be “reasoning by metaphor”[76]. They have drifted away from their original inspiration and failed to capture the richness of the immune system. These include simple models of clonal selection and immune networks [39][40][82][68], and negative selection algorithms [31][49][81]. Authors in [76] also states that bio-inspired computational algorithms usually proceed directly from a (naive) biological model to an algorithm, with little analytical framing of the representations properties. Such reasoning by metaphor is a troubling aspect of these algorithms. Without the application of suitable analysis techniques to the simplified representations of biological systems, algorithms derived from these representations rely only on the (often weak) analogy to the biological system to support their use. Therefore, to develop a good AIS, the authors[76] proposed the conceptual framework and promote an interdisciplinary approach, which will be described in Section 3.4. The

the original Dendritic Cell Algorithm (the original DCA) is a product of such an approach. It is also the foundation of the work presented in this thesis.

This chapter begins with a brief description of the immune system and illustrates why it suggests an immune inspired method for fault diagnosis. It is followed by a description of the the original DCA, however, there will not be much detail as the abstract model of Dendritic Cell (DC) is adopted by Diagnostic Dendritic Cell Algorithm (D-DCA) and will be described in Chapter 4. At the end of this chapter, the conceptual framework will also be introduced.

3.2 Immune fault diagnosis

Immune system is a defence system protecting a host from infections by foreign microbes or pathogens, such as bacteria, viruses. The immune system consists of two parts, innate and adaptive . Once the pathogens breach the skin barrier, the immune system performs four main tasks:

- Recognition

  Involve a white blood cells provide immediate response and lymphocytes identify the pathogens.

- Effector functions

  To contain the infected cells and provide counter-attack response if possible, such as antibodies, complement system and T killer cells.

- Regulation

  Self regulation features of the immune system, which prevent reaction to host cells and tissues.

- Memory

Once the host has been exposed to a type of pathogens, the immune system will provide an immediate and stronger response for reinfection from the exact or similar types. Some of the memory could be life-long lasting and some are relatively short.

The innate part is considered as the first layer of an immune system. Once there is an infection, innate immunity eliminates the pathogens within a short period of time. If the pathogens are not cleared, then often an inflammation is recruited and the pathogens are taken by antigen presenting cells for further analysis and ready for activation of adaptive immune response. Dendritic cells play a very important role between the innate and adaptive. They ingest debris from dead cells, extract antigens from them and present antigens to activate T cells. How they process is proposed in the "Danger Theory"[64].

The "Danger Theory" was proposed by Polly Matzinger, in [64]. It was based on "self" and "non-self" model, but added another layer of cells and signals, proposing that Antigen Presenting Cells (APC) are activated by danger/stress signal from injured cells. The Danger theory argues the discrimination of "self" and "non-self" within the immune system is not the only reason for the initiation of an immune response. The trigger of a response is not the recognition of the "foreignness" of the invader, but the "danger/stress". The Danger Theory opens a new viewpoint of immunology. The "Danger Theory" has blurred the distinction between the adaptive and innate of immune system. Most importantly, it provides a sound explanation where "self" and "non-self" model cannot explain, such as transplant and puberty.

The "Danger Theory" emphasises the APC specially dendritic cell plays a vital role in triggering an immune response. APC recognises the signalling molecules released by the death of a cell. This is critical in initiating an immune response. There are programmed cell death called apoptosis and "danger" death called necrotic.

The dendritic cell is the immune solution for "diagnosis", where it exacts the feature from the debris of a dead cell and associates with the signal. Then it presents the antigen to T cells. The "diagnosis" is done collectively by cooperation of dendritic cell and T cells. This has inspired the author to present the work in this thesis.

3.3 Artificial Immune System for fault diagnosis

Artificial Immune System

Artificial immune systems as a discipline lies within the jurisdiction of biologically inspired computing. Unlike other bio-inspired algorithms, such as genetic algorithms and neural networks, an AIS refers to any algorithm inspired by the immune system and not to a specific algorithm or technique. AIS algorithms typically fall into one of four groups: negative selection[45], clonal selection[41], immune networks[27] and danger theory[52]. Comprehensive reviews on AIS can be found in [54][85][37].

In [83], the author has argued that AIS has slowly drifted away from the more biologically appealing models and attention to biological detail, with the focus on a more engineering-oriented approach. Many algorithms have been said to be "reasoning by metaphor"[76]. These include simple models of clonal selection, immune networks and negative selection algorithms mentioned above. The author[83] further explained that, for example, the CLONALG lacks any notion of interaction of B-cells with T-cells, MHC or cytokines. In addition, the large number of parameters associated with the algorithm, whilst well understood, make the algorithm less appealing from a computational perspective. aiNET does not employ the immune network theory to a great extent. Only suppression between B-cells is employed, whereas in the immune network theory, there is suppression and stimulation between cells. Negative selection, the simple random search strategy employed, combined with using a binary representation, makes algorithm computationally so expensive that it is almost unusable in a real world setting[77].

However, most recent work from [52] has started to address this imbalance. For example, the authors[52] investigate novel ideas from immunology such as "danger theory" [64], with application to computer security. Those authors propose to observe the biological system by undertaking new experiments to identify key signals involved in dendritic cells. This has followed the conceptual framework proposed in [76], although, it was not refer to. In order to well develop an AIS, the authors [76] suggest the conceptual framework, which will be explained in Section 3.4.

The original Dendritic Cell Algorithm

The purpose of the original Dendritic Cell Algorithm (the original DCA)[50] is to correlate disparate data-streams in the form of antigen and signals and to label groups of identical antigen as normal or anomalous. The the original DCA is formed as part of the new algorithm proposed in the thesis, namely "Core-DCA" described in Section 4.3. As mentioned above, the original DCA is a product of collaboration between biologist and computer scientists. The the original DCA has been applied mainly to computer security, such as anomaly detection[51][53][52]. There were also some attempts[52] to use it to classify benchmark data sets, such as

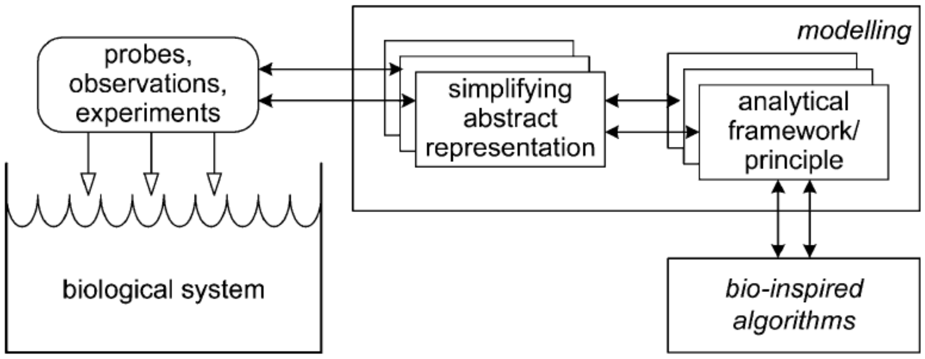

modelling
probes, observations, experiments
simplifying abstract representation
analytical framework/ principle
biological system
bio-inspired algorithms

Figure 3.1: The conceptual framework[76]

standard Breast Cancer machine learning data-set. However, the author found that the order of data fed to the original DCA had effects on its performance. Later on [50], the author stated that the original DCA is not suitable for a static data set, but suitable for a real time problem. The author [50] also claimed the "no training" and "prior" knowledge is required and fail to suggest that how one can obtain the signals and antigen associations, which have to be supplied to the the original DCA. This might be the reason why the the original DCA is not currently developed and studied widely. Nevertheless, the the original DCA's key characteristics can be summarised as light weight in terms of computational consumption, efficiency and accuracy [50]. This suggests that the the original DCA might be suitable for the robotics system use.

3.4 Conceptual Framework

In the paper[76], the authors propose that bio-inspired algorithms, such as AIS, are best developed in a more principled way. To clarify this, the authors suggested that many AIS developed had drifted away from the immunological inspiration and failed to capture the complexity and richness that the immune system offers. The authors suggest the conceptual framework, shown in Figure 3.1, for developing bio-inspired algorithm with bias for its engineering need. This should avoid the reasoning by metaphor approach often seen in bio-inspired computing. The conceptual framework promotes an interdisciplinary approach, involving the design of AIS through a series of observational and modelling stages in order to identify the key characteristics of immunological process.

The first stage of the conceptual framework is to probe the biology by observations and experiments, in order to provide a partial view of the complex biological system. This view is then used to build and validate abstract models of the biology. These models can be both mathematical and/or computational, and are open to validation techniques not available to the actual biological system. The iteration of validation of the models and probing the biology, would be beneficial to both the biologists and the computer scientist to construct a sound algorithm.

In the same paper[76], the authors further applied the conceptual framework at a higher level to the bio-inspired computational domains using the same structure. The authors examined and compared the separate conceptual, mathematical and computational frameworks, to develop more integrated and generic frameworks, and to expose essential differences. In the same way probing the biology, the author asked "meta-questions" to understand the insight of systems/algorithms. The questions addressed notions such as *openness, diversity, interaction, structure*, and *scale*. By continuously asking those questions, it will influence the development of an algorithm. Development of the algorithm, Diagnostic Dendritic Cell Algorithm (D-DCA), proposed in this thesis was applied to the conceptual framework. Although, D-DCA is not a product of close collaboration with biologists, the the original DCA is. During development of the D-DCA, the "meta-questions" have been asked to justify any modification. They will be answered in later Chapter 4. By doing so, it helps to identify the key characteristics of the algorithm.

3.5 Summary

In this chapter, a natural immune "fault diagnosis" solution has been introduced, which is because of cooperation of the Dendritic Cell (DC) and T cells; an Artificial Immune algorithm, the the original Dendritic Cell Algorithm (the original DCA), inspired from the DC has also been introduced, although briefly; and the conceptual framework has been introduced to avoid drifting away from the original inspiration in order to develop a good algorithm. These have been encouraging for one to develop an algorithm inspired from the immune system to achieve fault diagnosis, the Diagnostic Dendritic Cell Algorithm (D-DCA), which will be presented in Chapter 4.

Chapter 4

Diagnostic Dendritic Cell Algorithm

Contents

Tables

Figures

4.1 Introduction

So far, the challenges were outlined in Chapter 2 and an immune solution for fault diagnosis has been introduced, that is the Dendritic Cell (DC) in Chapter 3. Subsequently, it is apparent one needs to abstract from the inspiration and develop an algorithm to handle the challenges.

This chapter proposes an immune inspired algorithm for fault diagnosis, the Diagnostic Dendritic Cell Algorithm (D-DCA). The development of D-DCA is base upon the the original Dendritic Cell Algorithm (the original DCA) [50]. The abstraction of DC functions, including its internal parameters and signalling is based on the work of [50]. However, the the original DCA was tightly integrated within the software environment[1], which was not suitable for our robotic environment. Therefore, there was the need for reimplementation.

During the development of the D-DCA, it was found that the production of association between the input signals (“danger”, “safe” and “PAMP”[2]) and the “items” (the things that one is trying to classify or diagnose whether “faulty” or not) was excluded from the the original DCA. The the original DCA relied on other system to provide such association. However, from early experiments using D-DCA, the quality of the association had direct impact on its performance. The necessity for including this within the D-DCA will be explained and argued, in

[1]Libtissue[87]

[2]Pathogen-associated molecular pattern

Section 4.3.

During the development of the D-DCA, it was also found that the certain aspects of apoptosis and necrosis were not taken into account of the original DCA, which were the foundation of the “Danger” theory [64] that the the original DCA was inspired from. The apoptosis and necrosis will be explained and their necessity will be argued in Section 4.2.2. Later in Chapter 5, it will be proved that adding this feature will robust the D-DCA’s performance.

This chapter begins by describing the inspiration taken from biology and how it is mapped to D-DCA. The D-DCA will then be presented. At the end, to identify the key characteristics, the D-DCA will be questioned in the forms of the “meta-questions” proposed within the conceptual framework.

4.2 Biology inspiration

In this section, a description of immune cells and processes will be presented. It will be followed by their abstractions and interpretations in D-DCA. There will be a few biology terms used, such as CSM, IL10 and IL12. But their biological meanings are beyond this work.

4.2.1 Dendritic cell

Biological Dendritic cell

Dendritic Cells (DCs) are antigen presenting cells[67], whose purposes are the cleaning, processing and presenting antigen to T-cells. DCs have three states, immature, semi-mature and mature. However, an immature DC can only become one of semi-mature or mature states. Immature DCs migrate through the bloodstream from bone marrow to enter tissues.

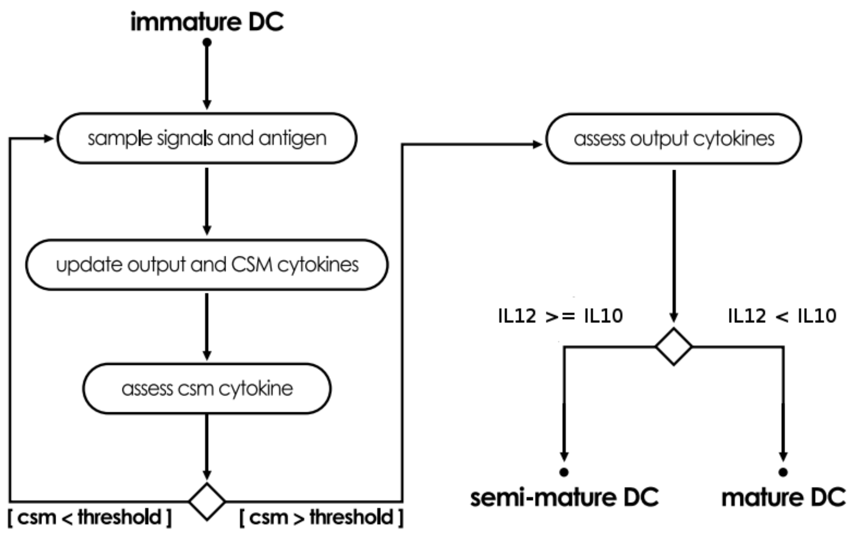


Figure 4.1: DC model activity diagram, modified from [50]

They continually ingest large amounts of the extracellular fluid. They then process what they have taken up and if they have ingested enough, they migrate to a lymph node. Once inside the lymph node, they turn into one of mature states, depending on what signal they have been exposed to.[64] If they have been exposed to significant “danger” signals, they will turn into mature state; If they have been exposed to significant “safe” signals, they will become semi-mature state. The mature DC will activate the T killer cells with specific antigen which is presented to them. The T killer cells then proliferate and migrate to tissues to kill the pathogens or infected cells with that specific antigen. The semi-mature DC will activate the T regulatory cells, which have the function of suppressing the activation and proliferation of specific T killer cells when a T cell is active or is activated, it means that its number will be increased.

Abstraction of Dendritic cell

The abstracted model of DC is adopted from [50], shown in Figure 4.1. It begins by initializing an immature DC. The immature DC then samples signals and antigens pair and updates the CSM by one for each sample. IL10 will be increased by one, if it samples a "danger" signal. IL12 will be increased by one, if it samples a "safe" signal. If the CSM is less than the *migration threshold*, the DC stays as immature and continues to sample. Otherwise, it undergoes migration. Before maturation, IL10 and IL12 will be compared. If IL10 is greater than IL12, then the DC will become a mature one. Otherwise, it will become a semi-mature DC. Each DC contains a list of antigens that it has sampled. Once matured, the mature DC will activate T killer cells with those antigens. The semi-mature DC will activate T regulatory cells.

4.2.2 Apoptosis and Necrosis

Biological Apoptosis and Necrosis

Apoptosis is a process of programmed cell death[67]. It derives from a Greek word meaning the falling of leaves from the trees, and is a general means of regulating the number of cells in the body. Every day the bone marrow produces millions of new cells and this production must be balanced by an equal loss.

When a cell undergoes apoptosis, typically [61], a cell shrinks and pulls away from its neighbours. Then blebs[3] appear on the surface, and chromatin condenses at the edges of the nucleus. The nucleus, and then the cell itself, breaks up, packs itself and is contained as cell fragments. These are ingested by other cells in the vicinity. Apoptosis also releases

[3]A bleb is an irregular bulge.[13]

"safe" signal. It happens quietly and cleanly.

Apoptosis can be initiated through many pathways, however, the interesting one is called *intrinsic* apoptosis[14]. One of the reasons for intrinsic apoptosis is the loss of cell survival factors, or other types of severe cell stress. It happens naturally and commonly within our body, and all cells will undergo such process.

On the contrary, *necrosis* is explosive, messy and releases "danger" signal. Necrosis can be induced by a number of external sources, including injury, infection, and inflammation. Due to the sudden busting of the cell, it releases harmful chemicals to surrounding tissues.

Abstraction of Apoptosis

Previously, it was mentioned that certain aspects of apoptosis and necrosis were not taken into account within the the original DCA. It refers to *intrinsic* apoptosis. Every cell undergoes programmed cell death. So do DCs and T cells. The author [50] used CSM and Migration threshold to control the life span of a DC. Once a DC becomes semi-mature/mature, it gets deleted from the population and a new immature DC will be created. The author proposes here that each cell should be given a separate parameter, namely "life". When a cell is created, it has to be initialised how long it will live. "Life" will decrease as time goes on. Once "life" has reached zero, the cell will be destroyed regardless of its type or state. This will be referred as "with death" approach, in Chapter 5.

In D-DCA, there is a population of DCs continuously sampling the system's status. With this "life" feature, it creates a time window for D-DCA. It allows the D-DCA to keep a record of the past. For a DC, the longer "life" it has, the further it will be able to look back. This enables the D-DCA to "learn" on the fly and adapt to changes in real time.

It could be argued here that adding another parameter and creating a time window is no different to having a variable/fixed time window for sampling. It is known the size of the time window will affect most system's performance. It would just be another extra parameter for the system to tune. The author here argues differently, by adding "life", it is the opposite to having another parameter. Each DC in the population has a different "life" span. Some have longer and they have sampled further back in time. Some have shorter and they have sampled less further back in time. The overall effect on the whole population would be to preserve a "normal" distributed[4] time window of the past. Although, "life" controls how wide the distribution is 0spread out, in Chapter 6, it will be proved that the variation of "life" affects the D-DCA's performance insignificantly.

4.2.3 T killer and T reg

Biological T killer and T reg

The main function of T killer cells is to initiate the infected cell to undergo apoptosis. Once a T killer cell is active, it proliferates and migrates to tissues. Through a series of complicated signalling and binding, it send the death signal to the infected cells. Then, macrophages (including DC) take care of the cleaning.

The main function of T regulatory cells is to suppress the activation of T killer cells. As their name suggests, they regulate the proliferation of T killer cells. This will help to cool down an immune response (massive increase of cell proliferation and cell death), when the pathogen is cleared. Also, it helps to prevent autoimmunity. There are significant numbers of T reg cells with the signature of the host. So this prevents any T killer

[4]May not be exactly normal distributed

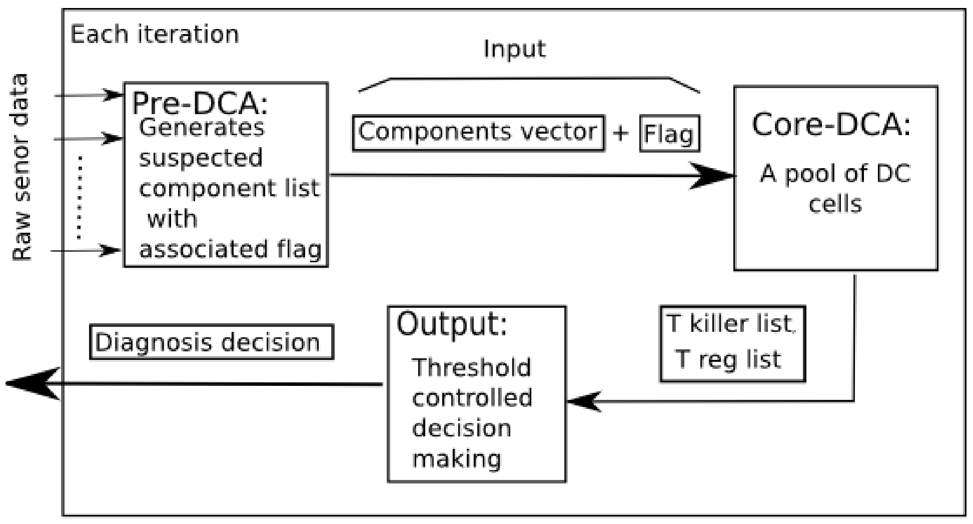


Figure 4.2: Diagnostic-DCA overview

cells as having an immune response.

Abstraction of T killer and T reg

The abstraction of T killer and T reg cells is at the analogy level, in D-DCA. There is no detailed abstract model and signalling. A sensor diagnosed as "faulty" is because the number of T killer cells is greater than T reg cells of such a sensor. The activation of a T cell is increasing its number. However, each T cell will be initialised a "life" and it will be destroyed when there is no "life" left.

4.3 D-DCA

This section proposes the D-DCA, which is an on-line fault diagnosis algorithm for robots. D-DCA is a real time version of the the original DCA, where it is modified be fit a robotic environment with additional features.

D-DCA is a population base algorithm. There is a pool of DC cells within a D-DCA. The memory consumption for D-DCA is the allocated memory for those cells. There is no complex arithmetic operation in D-DCA, except addition and subtraction. In this way, it can be modified to fit any resources limited system. Later in Chapter 7, it will be illustrated that the number of DC is insignificant to D-DCA's performance.

As the name suggests, the D-DCA only diagnoses faults, pinpointing the cause or causes of an erroneous behaviour in a robot. It is assumed that an error detection system exists that provides reasonable performance in the detection of such errors. The output of error detection is simply an "anomaly"/"normal" flag. The D-DCA consists of three blocks: *Pre-DCA*, *Core-DCA* and *Output*.

The ability of diagnosis is achieved by a combination of these three. In Figure 4.2, the *Pre-DCA* takes raw sensor data and shortlists suspect components (components vector) associated with the flag provided by the error detection system. This shortlist may contain noise, and can not be used for direct diagnosis. These components with their flag, named as inputs, will then be fed into the population of DC cells (*Core-DCA*), DC cells sample an input and store the shortlist, which effectively creates a memory containing what is happening within the system. The *Core-DCA* then creates a noise reduced list of "faulty"/"not faulty" components (T killer/reg list) based on the semi-mature/mature DC cells. By using thresholds, the *Output* identifies the "real faulty" component and makes a diagnosis decision.

Mapping between biology terms to D-DCA is shown in Table 4.1 and the key data objects of D-DCA are the following:

Components list = {IR sensor 0, IR sensor 1 ... IR sensor7, leftwheel, rightwheel, location} The components list is a list of IDs of components which could become faulty.

Table 4.1: Biology terms mapping to D-DCA

Biology	D-DCA
Antigen	Component
"Danger" or "safe" signal	Flag
a DC	a DC
T killer cell	T killer cell
T reg cell	T reg cell
Paired antigens and signal (release by cell undergoing apoptosis and necrosis)	Input

Flag = {anomaly, normal} Flag contains "anomaly" and "normal", which are inherited from the error detection system. In the Pre-DCA phase, flag is also a part of the Pre-DCA output.

Input = {Possible faulty components list, flag}, where *Possible faulty components* ⊆ *Components list*

A DC cell:

- State ∈ {immature, semi-mature, mature}
- CSM (Co-stimulus molecule) ∈ {0,1,2...} The DC output signals include a costimulation signal (CSM) which shows that the cell is prepared for antigen presentation and two context signals, the mature and semi-mature output signals.
- IL10 ∈ {0,1,2...} semi-mature output signals.
- IL12 ∈ {0,1,2...} mature output signals.
- MT (Migration threshold) ∈ {1,2,3..} The threshold value which determines whether a DC cell needs to move onto next state. Effectively, it controls how many antigen each DC cell can sample. By sampling an antigen, internal parameter co-stimulatory molecule will be increased by 1. If the co-stimulatory molecule of a DC cell

is greater than the migration threshold, then the DC cell will mature into one of mature states. Otherwise, the DC cell stays on immature state and can sample more antigens.

- Antigenlist: A list of $Input$. Once a DC has sampled an $Input$, its contents (*components* and $flag$) will be stored in this list.

A T cell:

- Component $\in$ Components list

A T killer/reg list: List of $Tcell$s

Pre-DCA

The purpose of *Pre-DCA* is pre categorising. For the D-DCA to work, it requires the weak association between "signal" and "item". In D-DCA, the "signal" is the output from error detection, which is an "anomaly"/"normal" flag. The components are the possible faulty components of a robot, such as IR sensors, wheels and location. The *Pre-DCA* block is fed by raw data, such as IR sensors proximity, wheel speed and location coordinates, and returns a list of component IDs with an associated flag, which is termed as *Input*.

The method used for *Pre-DCA* is a rule based system. The rules are, for example, to associate with an "anomaly", "Is this IR sensor value significantly different from nearby IR sensors'?"; "Has this component value had a large variation?". To associate with "normal", "Is this component value within standard deviation range for the past period?"; "Is this IR sensor value close to nearby IR sensors'?". However, the method used can be any other technique, as long it can extract the features to associate with the signals. One would argue that if the *Pre-DCA* has associated the "anomaly"/"normal" with the possible faulty components, the diagnosis has been done. That is indeed true, if *Pre-DCA* could provide the exact match between signals and components. As mentioned earlier, the pre categorising for DCA would only provide a weak association, which would be insufficient to diagnose by itself. That is one of the reasons for using a rule-based system here. Trying to write rules to fit a dynamic system is often impossible, due to the dynamics. The output from *Pre-DCA* block is often noisy. The suspect faulty components list often contains non-faulty components.

The following setting has been used: the signals have two categories "anomaly" and "normal" ("danger" and "safe"). It is because for robot system there might not be "know bad" which can be categorised as "PAMP" signal, which was used in the the original DCA.

```
Core-DCA
    Input: Input $componentlist, Signal$
    Output: List of T cells $Tkillerlist$
    begin
        select $N$ DCs from the pool to sample;
        for $n \in N$ do
            DC[n].CSM++;
            DC[n].insert ($componentlist$);
            if $Flag$ == "anomaly" then
                DC[n].IL10++
            end
            else
                DC[n].IL12++
            end
            if $CSM$ >= $Migration$ $threshed$ then
                if $IL10$ >= $IL12$ then
                    DC[n].state = Mature;
                    $Tkillerlist$ = $Tkillerlist$ + DC[n].activateTcell()
                end
                else
                    $Tkillerlist$ = $Tkillerlist$ - DC[n].activateTcell()
                end
            end
        end
    end
```

Algorithm 1: Core-DCA

The *Core-DCA* is a population of DC cells. Depending on what input a DC has sampled, it changes its internal variables and activates corresponding T cells. The pseudo code for *Core-DCA* is shown in Algorithm 1. For each iteration, a number of DC cells are selected to sample an

Input, which is a list of components and the flag from the error detection system. It will return a list of active T killer cells list. For each selected DC, the internal variable CSM increases by 1 and the list of components is inserted into the DC's *Antigenlist.* Depending on the signal ("anomaly/normal"), if "anomaly", *IL10* increases by 1. Otherwise, *IL12* increases by 1. After sampling, if a DC has sampled enough inputs (CSM>=MT), then this DC could activate T cells. This means that all the *components* in the *AntigenList* will be used to create corresponding T cells. If a DC reaches a "mature" state (IL10>=IL12), then the DC activates T killer cells. Otherwise, the DC activates T reg cells. A T reg cell suppresses T killer cell activation, which means reducing the quantity of T killer cells. One T reg reduces one T killer. For example, if a DC has an "antigenlist" of "[IR sensor 1, IR sensor 3],[IR sensor1, leftwheel]" and it has reached a "mature" state. Then, two IR sensor 1, one IR sensor 3 and one leftwheel T killer cells will be created. If there is another DC cell that has an "antigenlist" of "[IR sensor 1, IR sensor 3],[rightwheel, leftwheel]", but, it reaches a "semi-mature" state. Then, the activate T killer cells list would become one (2-1) IR sensor 1, minus one rightwheel and no T killer cells for other components. A negative number of T killer cells means that there are more T reg than T killer cells of its type. The T killer cell list will be passed to the *output* block to make a diagnosis decision.

```
Input: $TkillerList$
Output: $DecisionVector$
begin
    for $i \in Allcomponents$ do
        if ($TkillerList[i]$ >= quantity threshold) or
        ($TkillerList[i]/max(TkillerList)$ >= percentage threshold)
        then
            $DecisionVector[i] = 1$
        end
        else
            $DecisionVector[i] = 0$
        end
    end
    return $DecisionVector$
end
```

Algorithm 2: Pseudo code of Diagnosis decision

Output

The *output* block takes a list of T killer cells and returns a vector of diagnosis decision. The width of diagnosis decision vector is the number of the components. The pseudo code for *output* block is given in Algorithm 2. The decision on each component is based on how many T killer cells exist for that component. If there are more T killer cells than the quantity threshold or the percentage T killer cell of one component (with respect to the maximum T killer cell of all components) is larger than the percentage threshold, then this component will be diagnosed as “faulty”. Otherwise, it will be diagnosed as “non-faulty”, where “1” indicates faulty and “0” indicates non faulty

4.4 Investigating D-DCA

Having presented the D-DCA, the key characteristics of it will be discussed in this section. To identify the characteristics, the conceptual framwork is applied by asking the "meta-questions", such as diversity, interaction and scale.

4.4.1 Diversity

Diversity is present on a number of levels: the presence of multiple types of immune cell provides a layer of heterogeneity; although from the same population, each DC acting independently provides another; all DCs have different life span at any time and are randomly selected to sample. This means there are no (less chance for) two identical DCs existing at a given time. These diversities provide two key advantages: it allows the D-DCA to tolerate noise, such as an one off event or sudden irregular sensor data in very short time (*transient fault*). It also allows the D-DCA to process the diverse signals from a number of sources (sensors) and diagnose with multiple outputs (multiple faults diagnosis).

4.4.2 Interaction

There are two loops of interaction within D-DCA, T killer and T reg loop. To illustrate this, one could assume such a robotic system with sensor A, B and C. There is an error detection which can flag "anomaly" or not by monitoring the behaviour of the robot. The D-DCA is used for fault diagnosis and a recovery system is in place. And sensor B is faulty at present.

T killer loop

The error detection system indicates that there is a fault present, as the faulty sensor causes the robot to behave abnormally. Once flagged as "anomaly", the DCs interact with the robot by sampling sensor data. Once sampled enough, the DCs become mature DCs and then activate the correspond T killer cells (of sensor B). Once active, the number of T killer cells will be increased. This happens continuously and repeatedly. The longer the fault presents, the more T killer cells will be produced. With the faulty sensor identified, the recovery system would somehow compensate the effect of sensor B. This will change the behaviour of the robot and lead the error detection to flag as "normal". In this way, there will be less DCs becoming mature and less active T killer cells. Eventually, T killer cells (of sensor B) will die out, due to each cell having limited life. Then, sensor B will no longer be indicated as "faulty". However, purely relying on cell's death will not be quick enough to stop the identification of sensor B as "faulty", as cells death takes time. This would cause mis-diagnosis and increase false positive. To shut down the immune response (diagnosis as "faulty") quickly, there is the T reg loop, once there is no fault present.

T reg loop

While there is no fault present, the error detection system indicates the system as "normal". The DCs are sampling the sensor data. Once sampled enough, the DCs will become semi-mature DCs and then activate the T reg cells (of all sensors A, B and C). Once active, the number of T reg cells will be increased. The D-DCA is implemented as the diagnosis decision made by the number of T killer cells of one sensor and the number of T reg cells of that sensor. The more T reg cells of a sensor will let the D-DCA diagnose such sensor as “non-faulty”. And if there is more T killer cells, then the D-DCA will diagnose such sensor as “faulty”.

The two loops are happening simultaneously and the cooperation of these two interactions helps the D-DCA to diagnose quickly and reduce the false positive. They allow the D-DCA to perform more robustly and efficiently.

4.4.3 Scale

The question asked here is “how big the DC population has to be in order to perform well? And at what cost?”. The second question is easy to answer. It is apparent that the more number of DCs the more memory it will consume. The more randomly selected DC to sample, the more operations it will require. And, possibly, one would assume that the “more” the “better” its performance will be. However, the limits of the DC population and the number randomly selected to sample for each iteration are not easily recognised. To answer this, further parameter analysis will be presented in Chapter 6. It reveals that it is not the “more” the “better”. The selections of the number of DC population and the number of selected to sample are actually insignificant to the D-DCA’s performance. In order to achieve a reasonable performance, the D-DCA will not require “more” computational power. This certainly suggests that the D-DCA is suitable for resource limited robotic system.

4.5 Summary

An immune inspired algorithm, the Diagnostic Dendritic Cell Algorithm (D-DCA), was presented in this chapter. The D-DCA was designed for a resource limited robotic system to diagnose faults in real time. By applying the conceptual framework while developing the D-DCA, several "meta-questions" were asked, addressing *diversity, interaction* and *scale*. A number of key characteristics have been identified, such as tolerance to noise, multiple faults diagnosis, robustness and efficiency.

To illustrate the feasibility of D-DCA on a robotics system, a series of tests has been conducted and presented in Chapter 5. A further study of its parameters is presented in Chapter 6. A comparison of performance with Artificial Neural Network (ANN) and Support Vector Machine (SVM) will then be presented in Chapter 7.

Chapter 5

Feasibility analysis

Contents

Tables

Figures

5.1 Introduction

In this chapter, the performance of Diagnostic Dendritic Cell Algorithm (D-DCA) will be assessed , proposed in Chapter 4. The performance of the two different D-DCA approaches, “No death” and “with death”, will be compared. The “No death” approach is developed as a prototype D-DCA, which is without cells dying feature. Once a cell is created, there is no fixed life span associate with it. For example, a Dendritic Cell (DC) will be destroyed after it presents its contents and becomes either semi-mature or mature; A T cell will not die at all. It is soon discovered its limitation on certain type of fault, namely *permanent fault* described in Section 2.1. The “with death” is implemented with an additional “life” feature, described in Section 4.2.2.

One of the major problems associated with a classifier’s performance is the data set that it is tested on. Performance is dependent on data set. For example, if the data is imbalanced and has significant one to many related entries, then the performance will not be so good, using simple geometric classification, i.e. K-nearest neighbour. The imbalanced data will cause such method to dominate the prediction of the new vector, as they tend to come up in the k nearest neighbours when the neighbours are computed due to their large number.[36] The data used in the experiments is generated from a simulated robot environment, namely *Stage* described in Appendix A. Since the "No death" approach can only diagnose *permanent fault*, the experiments presented in this chapter are focused on permanent stuck at fault. A detailed description of the data set is in Section 5.2.

Parameter setting is also a significant factor on performance. Variation on parameter often affects performance, depending on how sensitive one parameter is. The parameters in this study for both approaches have been chosen as initial experiments suggested, as this chapter is focused on feasibility rather than parameter analysis. The initial experiments were focused on to implement a light weight and fast response algorithm. A further study on parameters will be presented in Chapter 6. However, it is necessary to include what parameters are used in this study, which is described in Section 5.2.

For each data set, both approaches will be applied to produce a set of diagnosis decisions. Each experiment will be performed twenty times. The decisions will be compared against with desire outcome (actual fault) and computed a confusion matrix. The result will be plotted as a Receiver Operating Characteristic (ROC) curve, shown in Section 5.2. Then, the discussion of the results will be in Section 5.3.

5.2 Experiments and results

The goal of this exercise is to test feasibility of D-DCA and to compare performances of "No death" and "with death". In this section, how the experiments were conducted is explained. It begins by describing the data sets and parameters used in this exercise. Then, the result will be presented in ROC plots and discussed at the end.

Data sets

To test feasibility, 8 scenarios were designed. All of which were stuck-at-fault. The stuck-at-fault is one of most common faults found in mechanical systems. The 8 scenarios were selected because they cover the stuck-at-fault range that was interested. The fault was injected to the simulation environment, namely *Stage*, where it only affected the "faulty" sensor and left the rest intact.

In *Player* and *Stage*, the simulated robot is equipped with eight IR sensors, all equally spaced around the robot. IR sensor 2 was picked randomly for fault injection. The 8 scenarios are as following:

1. IR sensor 2 stuck at value 5% of Max sensing value (0.15 metre is the maximum.) started at position 1 (original start position)

2. IR sensor 2 stuck at value 5% of Max sensing value started at position 2 (random selected)

3. IR sensor 2 stuck at value 50% of Max sensing value started at position 1

4. IR sensor 2 stuck at value 50% of Max sensing value started at position 2

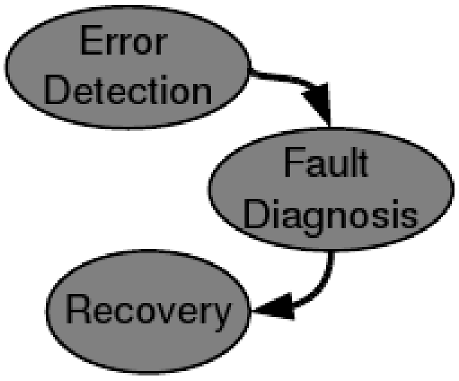


Figure 5.1: Fault tolerance

5. IR sensor 2 stuck at value 85% of Max sensing value started at position 1
6. IR sensor 2 stuck at value 85% of Max sensing value started at position 2
7. IR sensor 2 stuck at value 100% of Max sensing value started at position 1
8. IR sensor 2 stuck at value 100% of Max sensing value started at position 2

The data set was then generated from *Stage* and fed to D-DCA. For each scenario, data was recorded for 1000 time steps (about 2.8 minutes). "Fault" was introduced at time step 200 and presented until the end. For each time step, 8 IR sensor values, left/right wheel speed and location X/Y were recorded to form an input vector, shown in Table 5.1. Their value range is shown in Table 5.2.

An illustration of fault tolerance is shown in Figure 5.1. The output from an "Error detection" is a "faulty" or "non-faulty" flag, which indicates if there is fault within the system or not, respectively. The D-DCA is only responsible for fault diagnosis. A "faulty" label was also included in the input vector, where '1' indicated that at the current time

IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7
0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
Left	Right	X	Y	faulty			
0.0225	0.0225	1.75	0.70	1			

Table 5.1: A typical input vector

IR 0-7	$0 \sim 0.15(m)$
Left/Right wheel speed	$0 \sim 0.0225(m/s)$
X, Y	$-3 \sim 3(m)$ where $(0, 0)$ is starting position

Table 5.2: Input value range

step, there was a fault within the system. This label was the output from "Error detection" described in Section 4.3. Since faults were artificially injected, it was known when and where the faults were. However, one could not assume there was a perfect "Error detection", which can indicate correctly every fault within the system. Therefore, the "Error detection" mechanism with 80% accuracy was introduced. This means that if there is a fault, there is 80% chance this "Error detection" will indicate the system as faulty; if there is not a fault, it will indicate the system as non-faulty by 80% chance.

An input vector was fed to D-DCA and then an output would be produced at each time step. An output (shown in Table 5.3) from D-DCA was a vector of binary decisions on each component, where '1' was faulty and '0' was not faulty. The components for these experiments were sensors 0-7, left wheel, right wheel and location. This vector, shown in Table 5.3, could be interpreted as none of the components were faulty but IR sensor 2.

IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7	Left	Right	location
0	0	0	1	0	0	0	0	0	0	0

Table 5.3: A typical output vector

Parameter	Value
Migration threshold	20
Number of DC cells	20
Number of random select cells	5
Max life of cells (With death) (time steps)	50
Fault injected at time step	200
Quantity T killer threshold (No Death)	-800 to 2000
Quantity T killer threshold (With Death)	-100 to 200
Gradient T killer threshold (No Death)	0
Percentage threshold (With Death)	0%

Table 5.4: Parameters for feasibility experiments

Parameters

The parameters for this feasibility test were chosen after a few preliminary experiments. At this stage, there was no reason why one should use one set rather than the other. Later in Chapter 6, the findings for which parameters would affect the performance more than the others will be presented.

In order to produce a ROC curve, described in Appendix B, it is needed to alter at least one parameter. Quantity T killer threshold was chosen and its range is shown in Table 5.4. This threshold is one of the thresholds which controls diagnosis decision directly. If there are more T killer cells of one component than this threshold value, the component is considered as "faulty". The parameter sweeping ranges were chosen to give a full curve on ROC plot.

Result

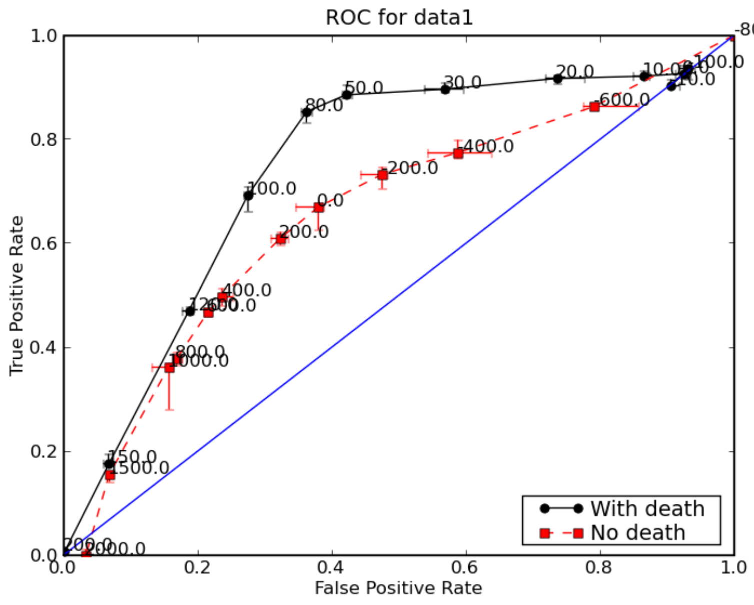
To assess performance, ROC curve was chosen. The author[30] recommends ROC in comparison to overall accuracy for single number evaluation of machine learning algorithms. The benefits are: Firstly, it is a visual representation and can be understand intuitively; Secondly, it plots True Positive Rate (TPR) (benefit) against False Positive Rate (FPR) (cost). This allows us to compare performances from both perspectives.

Both "No death" and "with death" were applied to the 8 scenarios data. Each experiment was performed 20 times for each data. A confusion matrix[1] was then computed using the outcome of diagnosis against the actual fault. From the confusion matrix, TPR and FPR were calculated and plotted on Figure 5.2. Each data point on the figure is the median value and the error bars are the upper and lower quartile of those 20 times.

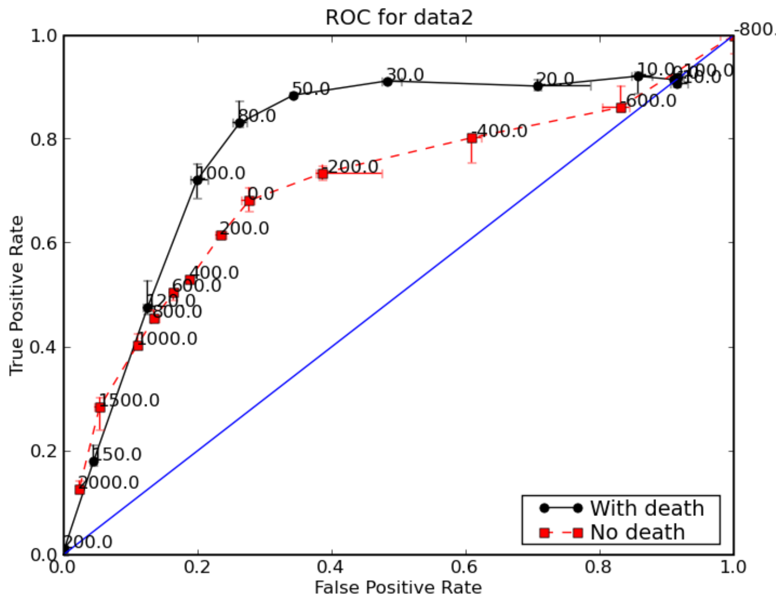
Real time performance

The real time aspect of the D-DCA can be found in how long it takes to correctly diagnose. Figure 5.3 shows a typical output from "with death" approach with a stuck-at fault on sensor 2 from time step of 200. Figure 5.3 consists of 3 sub-figures, top, bottom left and bottom right. Top figure shows the net T killer cells for each component over time (T killer number minus T reg number; the larger number the more "faulty" the component is.); bottom left shows the diagnosis decision on each component over time (Y axis indicates different components), where black indicates the D-DCA diagnose this component as "faulty" and red indicates the actual fault was injected; bottom right is a histogram on the diagnosis decision on the bottom left.

[1]Confusion matrix is described in Appendix B.1.

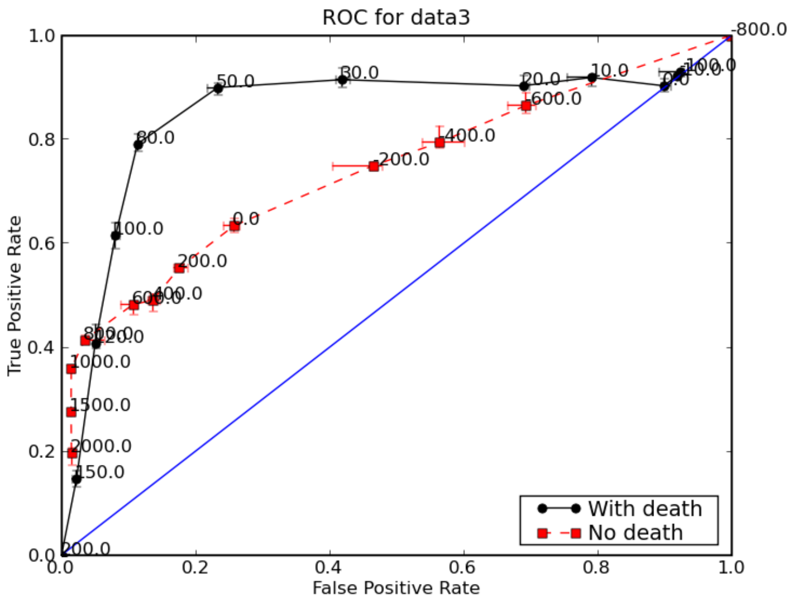


(a) Scenario 1

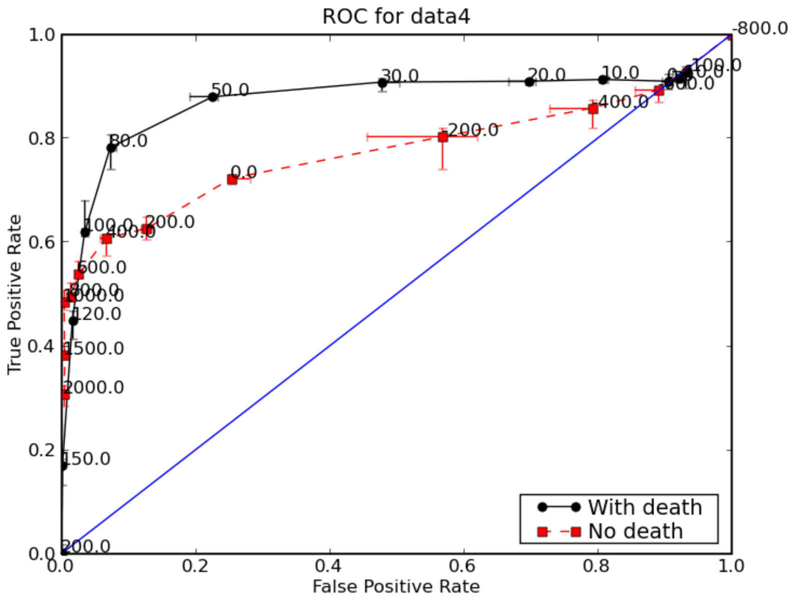


(b) Scenario 2

Figure 5.2: ROC curve for 8 different scenarios

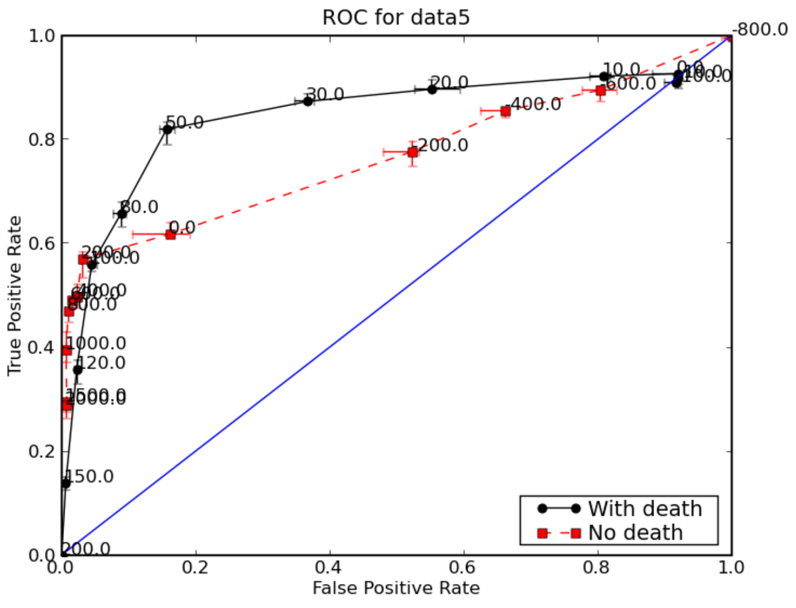


(c) Scenario 3

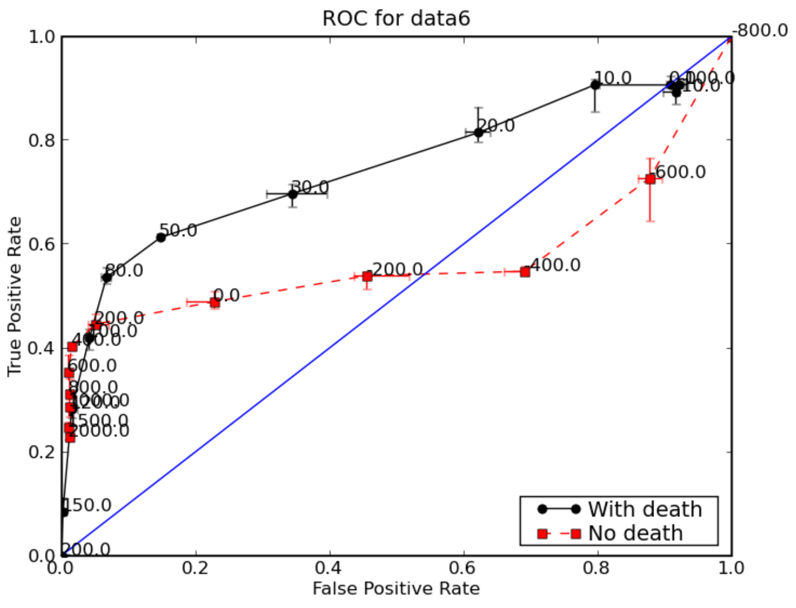


(d) Scenario 4

Figure 5.2: ROC curve for 8 different scenarios

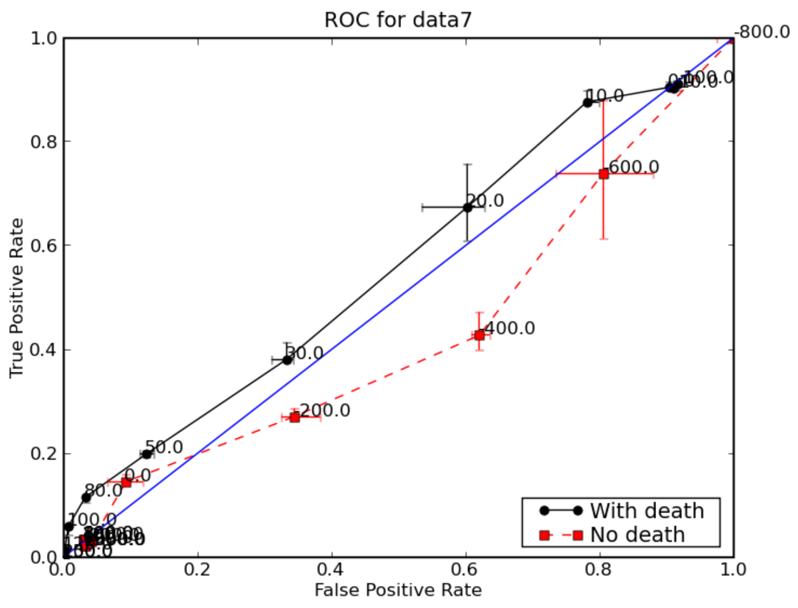


(e) Scenario 5

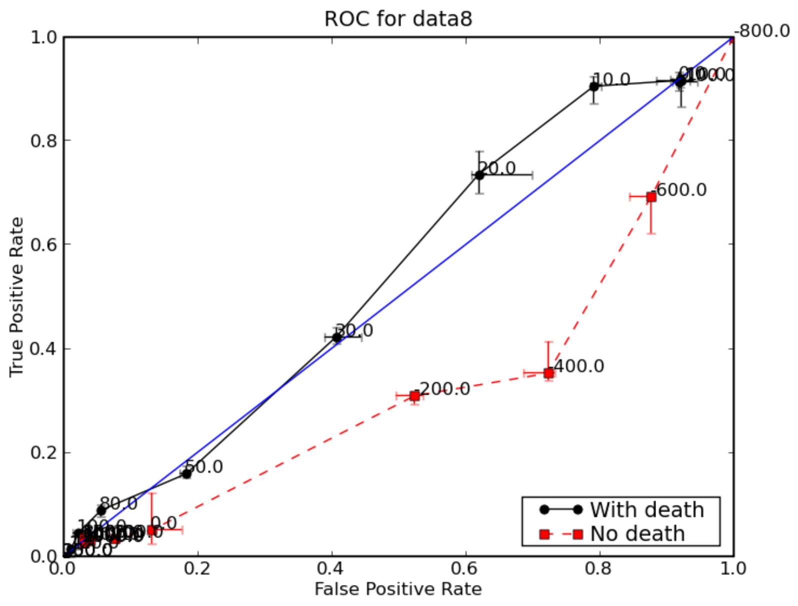


(f) Scenario 6

Figure 5.2: ROC curve for 8 different scenarios



(g) Scenario 7



(h) Scenario 8

Figure 5.2: ROC curves for 8 different scenarios: Each sub-figure shows the comparison of performances between "With death" and "No death" approaches. The performances are plotted as True Positive Rate (TPR) over False Positive Rate (FPR). The parameters used are shown in Table 5.4. For both curves, the TPR and FPR are calculated between the actual injected fault data and the diagnosis decision, from time step 200 to 800. Each data point on the graph is the median of 20 runs of experiments and the error bars are the upper and lower quartiles. The label near the points is the threshold used in the experiments.

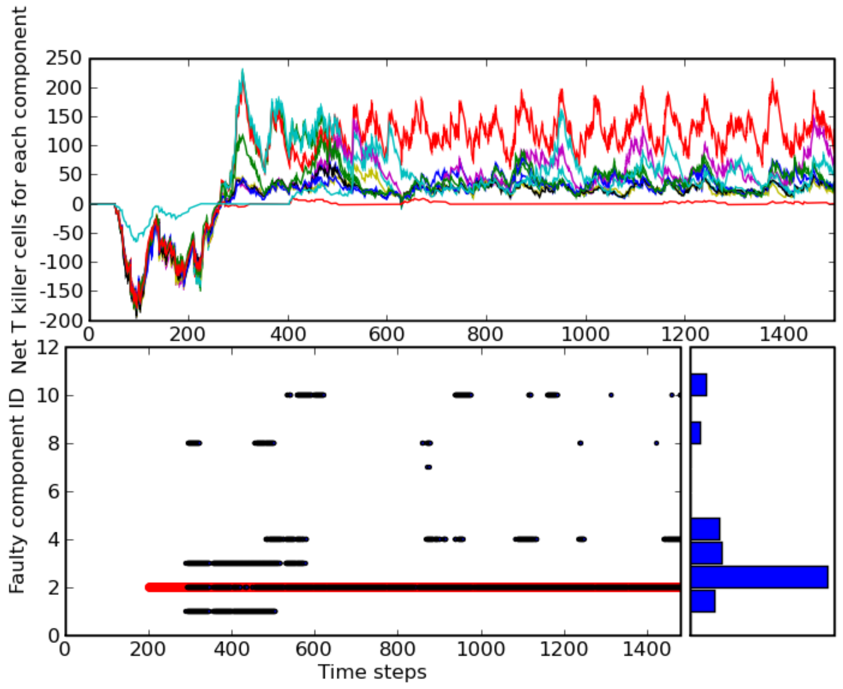


Figure 5.3: A typical output from with death approach

The bottom left figure in Figure 5.3 suggests that there is a typical 80 time steps (10 seconds) delay for D-DCA to correctly diagnose (differences between red and black on sensor 2). This delay is acceptable for the robotic system used in the experiment. After time step 500, it shows that the diagnostic decision is more accurate (less false positives).

5.3 Discussion

On a ROC plot, TPR (benefit) and FPR (cost) are plotted. One measurement of "goodness" is to use the area under the cure [44]. However, the performance which has higher "benefit" and lower "cost" is interesting. The performance is the closer to point (1, 0) the better, described in Appendix B. In this work, it is used the distance to (1, 0) as the "goodness" measurement on a ROC plot. In the context of the robotic system, it is considered that a performance is above 75% TPR and less than 20%

FPR as a “good” one. With an 80% accurate “Error detection”, the best of TPR would be 80% and 20% of FPR is acceptable for the robotic application.

The results of this study indicate that D-DCA is feasible to diagnose fault for the simulated robot. For scenarios 1, 2, 3, 4 and 5, shown in Figures 5.2a, 5.2b, 5.2c, 5.2d, and 5.2e, it achieved more than 80% TPR with less than 20% FPR for scenarios 3,4 and 5. However, one could argue FPR should be as low as possible, where 20% FPR might be too high. One should note that when designing the experiment, the 80% accuracy for “Error detection” was introduced. Therefore, 80% TPR would be what it was aiming for. Also, at this stage, the algorithm was not optimised or deeper studied. The result was an illustration that D-DCA was feasible for fault diagnosis for a simulated robot.

However, D-DCA could not diagnose with a reasonable TPR for scenario 6, shown in Figure 5.2f, as the performance of D-DCA was not above 75% TPR while the FPR was less than 20%. For scenarios 7 and 8, D-DCA poorly performed similarly as a random classifier. A possible explanation for this might be the difference between the data sets. Both scenario 5 and 6 had been stuck at 85% of maximum sensor value. However, arguably, D-DCA performed better in scenario 5 than 6. The only difference on data set was the starting position. Having analysed the simulation in more detail, it was understood that in scenario 6 the robot walked around in the arena avoiding fewer obstacles than in scenario 5. This means that the IR sensors would have a larger value (close to maximum) for most of time, in scenario 6. So the larger value on sensor would be diagnosed as “normal” (not faulty) by D-DCA. This might be the reason why D-DCA performed worse in scenario 6. It was similar for scenarios 7 and 8. It might suggest that there is a weakness of D-DCA.

That is if an input feature is considered as "normal" most of time, it might be miss classified by D-DCA as "non-faulty", due to the fact that diagnosis decision is made by a majority voting rule in D-DCA. In this instance, the large sensor value was such an input feature.

The results of this study indicate that the "With death" outperformed the "No death" approach. In Figures 5.2a, 5.2b, 5.2c, 5.2d, and 5.2e, "With death" performed better (closer to (1,0)) with certain parameter settings. Its performance's variation of 20 runs was less than "No death" approach, shown in Figure 5.2 as in error bar.

From the results, they further support the idea that a classifier's performance is dependable on data set. With Quantity T killer threshold set to 80, the "with death" approach performed about (0.82 (TPR), 0.37 (FPR)) in scenario 1; (0.79, 0.1) in scenario 3; (0.65, 0.1) in scenario 5 and only (0.1, 0.05) in scenario 7. This had shown that the same algorithm, with the same parameter setting, but with different data set, could perform differently.

From the results, they further support the idea that parameters affects the performance. Although, it was not discussed the parameters used in this exercise deeply, the performance varied from "not working" to "good", within the Quantity T killer threshold range (shown in Table 5.4). In fact, the lower this threshold was the higher the FPR. By lowering the threshold, it would be easier for a component T cell to outnumber it. Thus, D-DCA diagnosed such a component as "faulty". On the contrary, a component would not be easily diagnosed as "faulty", by increasing the threshold. So there would be less FPR, but at the mean time, TPR would also be reduced. This effect was shown in Figure 5.2, where those data points, with high Quantity T killer threshold, appeared close to (0, 0). For most "good" ones in Figure 5.2, 80 Quantity T killer threshold would be a balanced choice for the "with death" approach. However, it is insufficient to conclude this. In fact, it only was the case of these 8 scenarios with Quantity T killer threshold range shown in Table 5.4. A further study with more focus on parameter analysis is therefore suggested and is presented later in Chapter 6.

5.4 Summary

This study set out to determine the feasibility of D-DCA and compare with "No death" and "with death" approaches. It started with describing the data sets and parameters set used for this exercise. Then, the results on the feasibility of D-DCA for a simulated robot were presented. It had been discussed that the D-DCA performance could be affected by data set and parameters.

The following conclusions can be drawn from this study. The results have shown D-DCA can be used as fault diagnosis with reasonable success. However, it is also suggested that D-DCA can diagnose incorrectly for certain data sets, where data is labelled as "normal" most of the time. The results clearly have shown the "with death" is better than the "No death" approach. Therefore, any D-DCA beyond this point will only refer to the "with death" approach. This exercise will serve as a base for future studies and more importantly, it has shown ROC could be useful to assess performance.

However, with limited data sets (8 scenarios only on permanent stuck-at fault), caution must be applied, as the findings might not be transferable to other data sets. Thus, a wider variety data sets, such as different types of faults, and faults on different sensors, has to be considered in any future study.

One of the more significant findings to emerge from this study, although preliminary, is that the variation of parameter affects the D-DCA performance. Thus, further experimental investigations are needed to analyse the parameters and are presented in Chapter 6.

Chapter 6

Parameters sensitivity analysis

Contents

Tables

Figures

6.1 Introduction

Parameters sensitivity analysis is an important component in the validation of an algorithm, and plays a key role in understanding how variation of parameter affects the algorithm's performance. It also provides vital information on optimization of an algorithm. If parameters are hypersensitive then they could be said to be "critical". In this instance, the parameters may be so finely tuned to the data that a slight change of values could instigate chaotic behaviour, thus making it very difficult to select. On the contrary, the parameters of an algorithm might show resilience to change, therefore making the algorithm robust and parameter selection a simple task. An example of this can be found in AIRS [93].

However, there has been little discussion about the effect of parameter variation on the behaviour of some immune inspired algorithms. It is often glossed over or not revealed at all, with authors empirically defining parameters that work for their particular data set with little discussion on how such parameters could affect performance. Examples of this can be found from early work on AIS, such as aiNet [38]. This might potentially discourage one to adopt and further develop it. On the contrary, some have addressed the parameters sensitivity. The author in [50] proposed the the original Dendritic Cell Algorithm (the original DCA) and addressed its parameters' sensitivity. In this, the author had concluded some of the parameters are sensitive on performance, specially "number of DC cells in the pool" parameter. Since Diagnostic Dendritic Cell Algorithm (D-DCA) is a reimplementation with modification of the the original DCA. D-DCA has inherited some of the original DCA's parameters. It is necessary to verify the findings in [50] if they apply to D-DCA and also provide a full parameters sensitivity analysis for D-DCA.

The aim of this study is to evaluate and validate the sensitivity of parameters. These are: Migration threshold (MT), Number of DC cells (DC), Number of random select cells (NS), Max life of cells (Life), Quantity T killer threshold (Tk) and Percentage threshold (Per). Within those, MT, DC and NS are inherited from the original DCA; Life is unique to D-DCA; Tk and Per are created for making diagnosis decision and unique to D-DCA.

This chapter begins by describing those parameters under the test. It then moves on to explain how the experiments have been conducted and present the results. Then, it explains how the findings are validated and discussed at the end.

6.2 Parameters in question

- Migration threshold (MT): The threshold value which determines whether a DC cell needs to move onto next state. Effectively, it controls how many antigen each DC cell can sample. By sampling an antigen, internal parameter co-stimulatory molecule will be increased by 1. If the co-stimulatory molecule of a DC cell is greater than the migration threshold, then the DC cell will mature into one of mature states. Otherwise, the DC cell stays on immature state and can sample more antigens.

- Number of DC cells (DC): The number of DC cell which there are in the pool when the algorithm is initialised.

- Number of random select cells (NS): How many DC cells which will be selected to sample an antigen at each iteration.

- Max life of cells (Life): How long a DC cell will live. (Only applies to "with death" approach.)

- Quantity T killer threshold (Tk): The threshold which determines the diagnosis decision on a component. If there are more T killer cells of a component than this threshold value, then this component will be diagnose as "faulty". Otherwise, as "non-faulty".

- Percentage threshold (Per): The threshold which determines the diagnosis decision on a component. If there are more T killer cells of a component than this threshold percentage of maximum of all component's T cells, then this component will also be diagnose as "faulty". Otherwise, as "non-faulty". For example, if there are 10 components, Percentage threshold is 90%, component 1 has 100 T killer cells and component 2 has 92 T killer cells, then both

Table 6.1: Preliminary findings for variation of parameters on performance. *$\uparrow$ means increasing. $\downarrow$ means decreasing. *Slightly* means less than 20% variation. *Significantly* means more than 50% variation..0000000

Parameter	Affects
MT	Affects are not conclusive, but beyond the range of [1,500], the results would not be relevant in this study.
DC	DC $\uparrow$: True Positive Rate (TPR) slightly $\uparrow$ and False Positive Rate (FPR) slightly $\uparrow$. Also, memory consumption $\uparrow$. Suitable range is [1,1000].
NS	NS $\uparrow$:TPR slightly $\uparrow$, FPR slightly $\downarrow$ and memory consumption significantly $\uparrow$. Suitable range is [1,1000].
Life	Life $\uparrow$: Memory consumption significantly $\uparrow$ Suitable range is [1,100].
Tk	Tk $\uparrow$: TPR slightly $\downarrow$ and FPR $\downarrow$. Suitable range is [0,500].
Per	Per $\uparrow$: TPR slightly $\uparrow$ and FPR $\uparrow$. Suitable range is [0,100%]

component 1 and 2 will be diagnose as "faulty" if "Error detection" indicates there is a fault present.

The preliminary experiments were conducted to get an overview of the affect on each individual parameter and to determine parameter's range of interest. By varying one parameter at a time, the findings are shown in Table 6.1.

6.3 Experiments and results

To evaluate and validate the sensitivity of parameters, this section begins by describing the data sets used in this study. The three sets of experiments were conducted will be explained, Wilcoxon signed rank test (described in Appendix D), Latin Hypercube sampling test (described in Appendix C) and D-DCA parameter evaluation. Then, the results in respective order will be presented and the findings will be discussed at the end.

Data sets

It was mentioned in Section 5.4 that a variety of fault types need to be included in the data sets, in order to demonstrate the D-DCA performance. There were 8 data sets generated from *Stage*, shown in Table 6.2. All of them are stuck-at fault. Data 1, 2, 3, 4 and 5 are *periodic faults*; Data 6, 7 and 8 are *permanent faults.* The parameters for data sets, such as component, stuck value, period and duration were randomly assigned.

The data set was then generated from *Stage* and fed to D-DCA. For each scenario, data was recorded for 4000 time steps (about 6.5 minutes). For each time step, 8 IR sensor values, left/right wheel speed and location X/Y were recorded to form an input vector, shown in Table 5.1. And their value range is shown in Table 5.2. A "faulty" label was also included in the input vector, where '1' indicated that at the current time step, there was a fault within the system. An input vector was fed to D-DCA and then an output would be produced at each time step.

Wilcoxon signed rank test results

The aim of this test is to explore the D-DCA's performance on individual parameter variation and to determine the significance of each. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples [15]. The distributions of D-DCA performances were not normal so non-parametric tests were run. The null hypotheses are shown in Table 6.3.

	faulty component	stuck at value	period (time steps)	duration (time steps)
data-1	IR sensor 0	63%	75	48
data-2	IR sensor 2	27%	778	742
data-3	IR sensor 4	45%	878	481
data-4	IR sensor 6	81%	186	181
data-5	IR sensor 7	38%	841	515
	faulty component	stuck at value	fault starts at (time steps)	
data-6	IR sensor 2	51%	1718	
data-7	IR sensor 6	34%	3951	
data-8	IR sensor 7	32%	2056	

Table 6.2: Parameter analysis data set

Table 6.3: Table of null hypotheses for Wilcoxon tests

Null hypothesis	Description
H1	Changing the MT will have no observable effect on the resultant D-DCA's performance.
H2	Changing the DC will have no observable effect on the resultant D-DCA's performance.
H3	Changing the NS will have no observable effect on the resultant D-DCA's performance.
H4	Changing the Life will have no observable effect on the resultant D-DCA's performance.
H5	Changing the Tk will have no observable effect on the resultant D-DCA's performance.
H6	Changing the Per will have no observable effect on the resultant D-DCA's performance.

Table 6.4: Parameter ranges for Wilcoxon and Latin Hypercube Test 1

Parameter	Range
MT	[1,500]
DC	[1,1000]
NS	[1,1000]
Life	[1,100]
Tk	[0,500]
Per	[0,100%]

Table 6.5: Base line parameters setting

Parameter	Value
MT	20
DC	100
NS	200
Life	100
Tk	120
Per	95%

The parameter ranges is shown in Table 6.4, which was suggested from previous experiments. 25 samples were selected for each parameter randomly, but with a continuous uniform [1] distribution. While varying one parameter, the other parameters were used as shown in Table 6.5.

The D-DCA with a parameters setting was performed on each data 20 times. The TPR and FPR were calculated for each experiment. To represent the performance of a parameters setting, the median was selected. The 25 medians of TPR and FPR (the performance) and the parameter values (the one was varied) were then used to perform Wilcoxon signed rank test. The results of minimum critical value for individual parameter are shown in Table 6.6.

To reject a null hypothesis, the minimum critical value has to be less than 68 for a two-tailed test with 99% confidence interval. This can

[1] The continuous uniform distribution is a family of probability distributions such that for each member of the family, all intervals of the same length on the distribution's support are equally probable. [16]

Table 6.6: Wilcoxon signed rank test 1

H_0	data-1	data-2	data-3	data-4
MT	0.0	0.0	3.0	0.0
DC	0.0	1.0	3.0	1.0
NS	0.0	0.0	1.0	1.0
Life	0.0	47.0	33.0	96.0
Tk	0.0	0.0	10.0	0.0
Per	1.0	129.5	9.5	30.0
H_0	data-5	data-6	data-7	data-8
MT	1.5	5.0	0.0	6.0
DC	2.0	3.0	0.0	5.0
NS	1.0	1.0	0.0	2.0
Life	80.5	30.5	137.5	0.0
Tk	6.0	10.0	0.0	25.0
Per	32.0	11.0	128.0	0.0

be found in Wilcoxon critical values table, shown in Appendix Table D.1. The highlighted cells in Table 6.6 were those where the hypotheses stood, where there were no differences between the parameters and the performances. Additional sets of parameters test were also conducted and the result can be found in Appendix E.

In addition to individual parameter hypotheses, a set of tests on one parameter against another were conducted. This was trying to illustrate the significance one parameter over another. The results are shown in Table 6.7.

Latin Hypercube sampling results

For sensitivity analysis, a sampling method has to be employed. The ultimate method is a full sweep through all the parameters, but for most cases, it is impossible. D-DCA's parameters are natural numbers. Even with the range of interest, it is impossible to compute a full sweep. Another method is random sampling. However, random sampling is the preferred technique when sufficiently large samples are possible, in order

Table 6.7: Wilcoxon signed-rank test result

H_0	data-1	data-2	data-3	data-4
DC - Tk	0.0	5.5	8.0	47.5
DC - Per	5.0	0.0	0.0	0.0
DC - MT	74.0	30.0	6.5	45.0
DC - NS	16.5	46.0	48.5	26.0
DC - Life	0.0	0.0	0.0	86.5
Tk - Per	7.0	0.0	0.0	0.0
Tk - MT	0.0	0.0	0.0	18.0
Tk - NS	3.0	3.5	26.0	3.0
Tk - Life	3.5	0.0	7.0	79.0
Per - MT	12.0	0.0	0.0	0.0
Per - NS	5.0	0.0	0.0	0.0
Per - Life	7.0	1.5	0.0	0.0
MT - NS	21.0	12.0	0.0	0.0
MT - Life	5.5	0.0	0.0	41.5
NS - Life	7.0	0.0	1.5	16.5

H_0	data-5	data-6	data-7	data-8
DC - Tk	5.0	0.0	29.5	0.0
DC - Per	0.0	0.0	3.0	0.0
DC - MT	4.5	17.5	1.0	31.0
DC - NS	30.5	24.5	53.5	55.0
DC - Life	1.0	5.0	3.0	104.0
Tk - Per	0.0	0.0	23.5	0.0
Tk - MT	0.0	0.0	0.0	0.0
Tk - NS	33.0	6.0	31.5	4.0
Tk - Life	20.5	12.0	10.5	101.5
Per - MT	0.0	0.0	0.0	0.0
Per - NS	0.0	0.0	0.0	0.0
Per - Life	3.0	5.5	84.5	27.5
MT - NS	1.0	0.0	1.0	55.0
MT - Life	0.0	0.0	0.0	55.5
NS - Life	12.0	8.0	0.0	79.5

to achieve higher accuracy. For this study, neither sampling methods can be used. To overcome this, Latin Hypercube sampling[2] was used. It is used because large samples are not computationally practicable and the estimation of very high quantiles (above 0.99) is not required. In caparison with random sampling, with the same workload, Latin Hypercube is proven more desirable. [72]

The experiment procedure is as following. Firstly, Latin Hypercube for D-DCA's parameter setting was generated. 1000 samples were generated for each test. Once the parameter settings were determined, D-DCA with each setting was applied to the data sets (described at beginning of this section). Each experiment was performed 20 times and the median was chosen to represent the performance of that parameter setting. Finally, those medians were used to calculate the correlation coefficient [17] of covariance [8]. Covariance indicates the level to which two variables vary together. The correlation coefficient is a value between -1 and 1 inclusive. The larger the absolute value of correlation coefficient the closer the two variables are varying together. In this study, the larger the (absolute of) coefficient means that one parameter is more sensitive on D-DCA's performance.

To determine whether a parameter is more sensitive on D-DCA's performance than the other, 3 sets of Latin Hypercube sampling were designed. Test 1 and 2 were two differently generated Latin Hypercubes. The aim of these two tests was to illustrate the individual parameter sensitivity and to show the consistency of the results. Test 3 was conducted using DC/NS ratio as one parameter, shown in Table 6.8. To give a more vivid presentation of the results form this study, a box plot was draw, for each parameter. Each box plot shows the correlation coefficients, ob-

[2]A detailed description how Latin Hypercube was generated can be found in Appendix C.

Table 6.8: Parameter ranges for Latin Hypercube Test 3

Parameter	Range
MT	1,500
DC / NS	1/1000 - 1000/1
Life	1,100
Tk	0,500
Per	0,100%

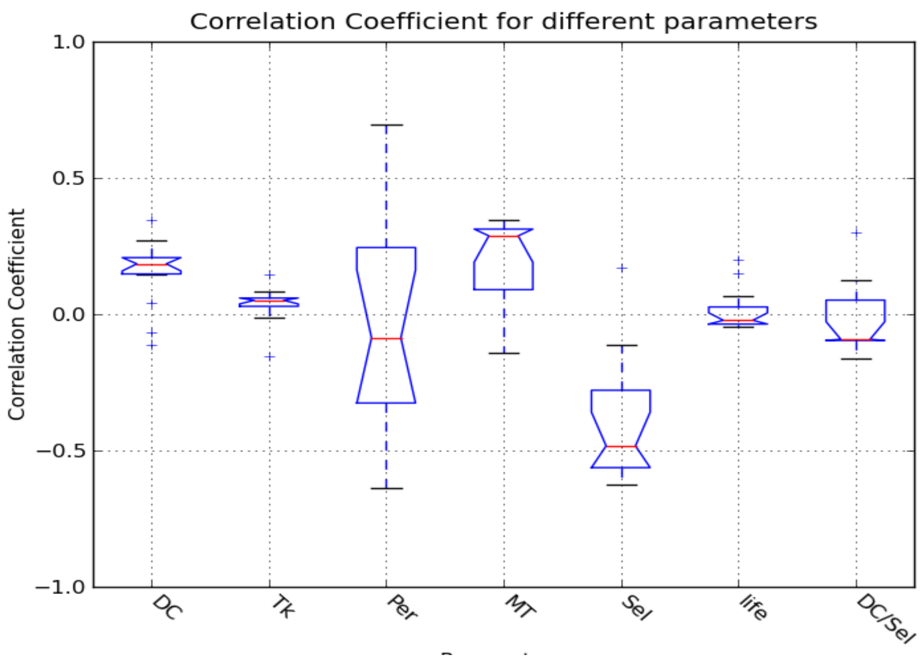


Figure 6.1: Test 1-01 Correlation Coefficient

tained from the Latin Hypercube test. Test 1, 2 and 3 results are shown in Figure 6.1, 6.2 and 6.3 respectively. Each test was also run multiple times to ensure the consistency within each test and more results can be found in Appendix C.

D-DCA parameters evaluation

To verify the the findings from the sensitivity analysis, a set of tests with different DC and NS parameters were performed. 3 sets of evaluations were conducted. E1 was varying NS with the parameters used and are

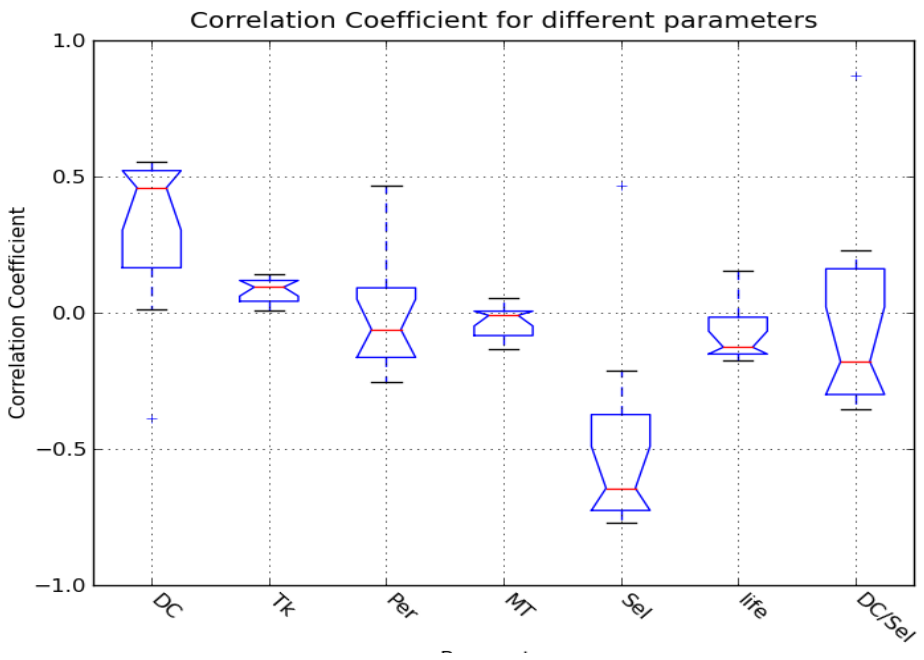


Figure 6.2: Test 2-01 Correlation Coefficient

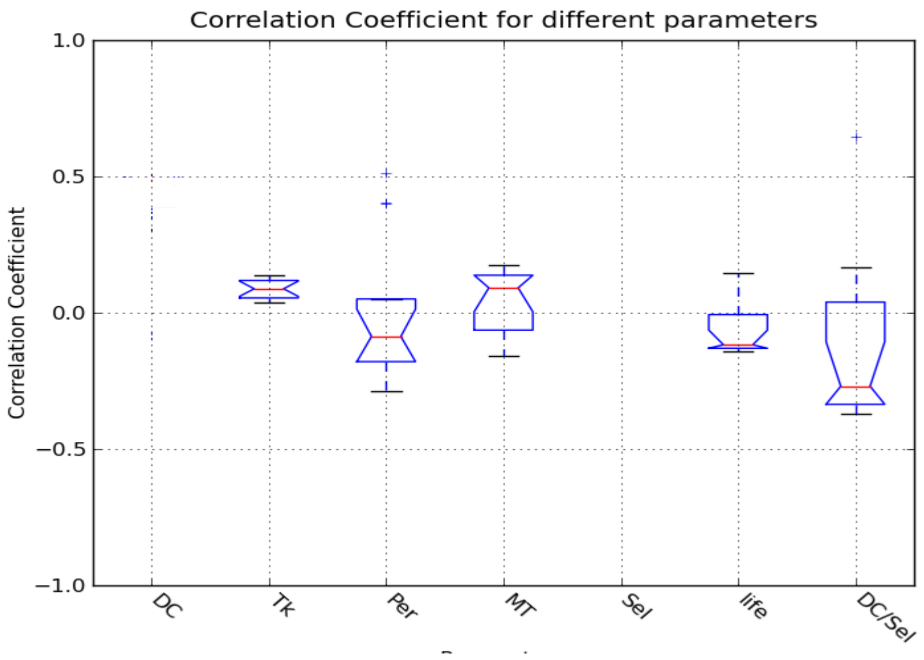


Figure 6.3: Test 3-01 Correlation Coefficient

Table 6.9: Parameters for E1

Parameter	Value
MT	20
DC	100
NS	10,50,100,500,1000
Life	100
Tk	120
Per	95%

NS	data-1	data-2	data-3	data-4
10	(0.0,0.11)	(0.72,0.14)	(0.27,0.05)	(0.06,0.13)
50	(0.01,0.12)	(0.78,0.18)	(0.6,0.06)	(0.1,0.19)
100	(0.01,0.12)	(0.79,0.18)	(0.63,0.07)	(0.13,0.21)
500	(0.0,0.12)	(0.79,0.19)	(0.65,0.07)	(0.12,0.19)
1000	(0.0,0.11)	(0.71,0.24)	(0.64,0.08)	(0.04,0.13)
NS	**data-5**	**data-6**	**data-7**	**data-8**
10	(0.34,0.09)	(0.79,0.06)	(0.0,0.0)	(0.53,0.23)
50	(0.63,0.14)	(0.92,0.11)	(0.0,0.0)	(0.86,0.35)
100	(0.67,0.14)	(0.92,0.11)	(0.07,0.0)	(0.89,0.36)
500	(0.69,0.14)	(0.93,0.12)	(0.04,0.0)	(0.89,0.36)
1000	(0.64,0.16)	(0.89,0.13)	(0.0,0.02)	(0.77,0.37)

Table 6.10: Performance verification E1, by varying NS

shown in Table 6.9 and the results are shown in Table 6.10; E2 was varying DC and NS but keeping their ratio as 1:1, with parameters shown in Table 6.11 and the results are shown in Table 6.12; E3 was varying DC and NS but keeping their ratio as 1:2, with parameters shown in Table 6.13 and the results are shown in Table 6.14. E4 was varying NS but keeping their ratio from 10:1 to 1:10 , with parameters shown in Table 6.15 and the results are shown in Table 6.16. All the data in the result tables are the median value of TPR and FPR, where D-DCA has performed 20 times one each data set. Further results of variation on performance are shown in Appendix F.

Table 6.11: Parameters for E2

Parameter	Value
MT	20
DC	10, 50, 100, 500, 1000
NS	10, 50, 100, 500, 1000
Life	100
Tk	120
Per	95%

DC/NS	data-1	data-2	data-3	data-4
10/10	(0.0,0.08)	(0.68,0.15)	(0.57,0.06)	(0.04,0.12)
50/50	(0.01,0.12)	(0.78,0.18)	(0.63,0.07)	(0.11,0.19)
100/100	(0.01,0.12)	(0.78,0.18)	(0.64,0.07)	(0.13,0.2)
500/500	(0.01,0.13)	(0.79,0.19)	(0.64,0.07)	(0.15,0.22)
1000/1000	(0.0,0.13)	(0.79,0.19)	(0.64,0.07)	(0.15,0.22)

DC/NS	data-5	data-6	data-7	data-8
10/10	(0.58,0.11)	(0.81,0.09)	(0.0,0.0)	(0.64,0.24)
50/50	(0.67,0.14)	(0.92,0.11)	(0.02,0.0)	(0.88,0.35)
100/100	(0.67,0.14)	(0.93,0.11)	(0.07,0.0)	(0.89,0.36)
500/500	(0.68,0.14)	(0.93,0.11)	(0.09,0.0)	(0.89,0.36)
1000/1000	(0.68,0.14)	(0.93,0.11)	(0.01,0.0)	(0.89,0.36)

Table 6.12: Performance verification E2

Table 6.13: Parameters for E3

Parameter	Value
MT	20
DC	10, 50, 100, 500, 1000
NS	20, 100, 200, 1000, 2000
Life	100
Tk	120
Per	95%

DC/NS	data-1	data-2	data-3	data-4
10/20	(0.0,0.1)	(0.75,0.17)	(0.61,0.06)	(0.08,0.16)
50/100	(0.01,0.12)	(0.79,0.19)	(0.65,0.07)	(0.12,0.2)
100/200	(0.0,0.12)	(0.79,0.19)	(0.65,0.07)	(0.14,0.21)
500/1000	(0.0,0.12)	(0.79,0.19)	(0.65,0.07)	(0.14,0.22)
1000/2000	(0.0,0.12)	(0.79,0.19)	(0.65,0.07)	(0.15,0.22)
DC/NS	data-5	data-6	data-7	data-8
10/20	(0.66,0.13)	(0.89,0.11)	(0.05,0.0)	(0.75,0.31)
50/100	(0.68,0.14)	(0.93,0.12)	(0.06,0.0)	(0.89,0.36)
100/200	(0.69,0.14)	(0.93,0.12)	(0.11,0.0)	(0.89,0.36)
500/1000	(0.69,0.14)	(0.93,0.12)	(0.06,0.0)	(0.89,0.36)
1000/2000	(0.69,0.14)	(0.93,0.12)	(0.07,0.0)	(0.89,0.36)

Table 6.14: Performance verification E3

6.4 Discussion

Wilcoxon signed rank test

To assess the validity of the null hypotheses, paired Wilcoxon signed rank tests were performed, where the pairs were the individual parameter values and D-DCA performance (TPR and FPR). The Wilcoxon rank test performed uses a confidence interval of 0.99. In order to reject a null hypothesis, the critical value has to be less than 68[3]. The results, as shown in Table 6.6, indicate that H1, H2, H3 and H5, from Table 6.3, can be rejected for their ranges with all the data sets used. H4 and H6 are partially rejected as they can not be rejected for some data sets,

[3]From Wilcoxon critical value table in Appendix E

Table 6.15: Parameters for E4

Parameter	Value
MT	20
DC	100
NS	10, 50, 200, 500, 1000
Life	100
Tk	120
Per	95%

DC/NS	data-1	data-2	data-3	data-4
100/10	(0.0,0.11)	(0.69,0.14)	(0.3,0.05)	(0.06,0.13)
100/50	(0.01,0.13)	(0.77,0.18)	(0.6,0.06)	(0.1,0.19)
100/200	(0.01,0.12)	(0.79,0.19)	(0.65,0.07)	(0.14,0.22)
100/500	(0.0,0.12)	(0.79,0.19)	(0.65,0.07)	(0.11,0.19)
100/1000	(0.0,0.11)	(0.7,0.24)	(0.63,0.07)	(0.04,0.13)

DC/NS	data-5	data-6	data-7	data-8
100/10	(0.31,0.09)	(0.79,0.06)	(0.0,0.0)	(0.51,0.23)
100/50	(0.63,0.14)	(0.92,0.11)	(0.0,0.0)	(0.87,0.35)
100/200	(0.69,0.14)	(0.93,0.12)	(0.15,0.0)	(0.89,0.36)
100/500	(0.69,0.14)	(0.93,0.12)	(0.1,0.0)	(0.89,0.36)
100/1000	(0.64,0.16)	(0.91,0.13)	(0.0,0.02)	(0.78,0.37)

Table 6.16: Performance verification E4

highlighted in Table 6.6, which it is not conclusive as rejected. However, more tests had been conducted which can be found in Appendix E. Here, it shows H5 can only partially rejected. This might be caused by the small number of samples (25) which had been used. Nevertheless, it can be concluded that the following from the results of Wilcoxon tests:

- Changing the MT will effect D-DCA's performance.
- Changing the DC will effect D-DCA's performance.
- Changing the NS will effect D-DCA's performance.

The findings of the this study are consistent with those in [50], where MT, DC and NS are inherited from. A further test was also conducted to address how one parameter effect differs from the other. The results, shown in Table 6.7, indicate that all parameters are different from each other in most cases. However, this is inefficient to determine if one is more sensitive than another. To address this, Latin Hypercube sampling test was conducted.

Latin Hypercube sampling test

To determine if one parameter is more sensitive than another, Latin Hypercube sampling test was conducted. The results are shown in Figures 6.1, 6.2 and 6.3. A figure is a set of box plots, which are the correlation coefficient of covariance. The coefficient is ranged between -1 and 1, where the larger absolute value of it the more sensitive the parameter is. A positive coefficient means that if the two variables under test are linearly related, then they are proportional[8]. Otherwise, if negative, they are inverse proportional. This might mean that if there is a lager positive coefficient of one parameter, then this parameter is more sensitive to the D-DCA's performance and the lager this parameter is the

Correlation/Sensitivity	Negative	Positive
None	0.09 to 0.0	0.0 to 0.09
Small	0.3 to 0.1	0.1 to 0.3
Medium	0.5 to 0.3	0.3 to 0.5
Strong	1.0 to 0.5	0.5 to 1.0

Table 6.17: Suggested correlation coefficient interpretation[8]

better the performance it would affect. In [8], the author has suggested the ranges for how correlated two variables are, shown in Table 6.17.

Test 1

The results from Figure 6.1 suggest the following:

- Although Per is more widely spread out than the others, the interquartile range still indicates that Per is a little sensitive to D-DCA's performance.
- DC and MT are a little sensitive to D-DCA's performance.
- Tk and Life are not sensitive to D-DCA's performance.
- NS is slightly more sensitive than the others.

Test 2

The results from Figure 6.1 suggest the following:

- Tk, Per, MT and Life are not sensitive to D-DCA's performance.
- DC is a little sensitive to D-DCA's performance.
- NS is significantly more sensitive than the others.

Test 3

The results from Figure 6.1 suggest the following:

- The ratio of DC/NS is slightly more sensitive than the others, but still a little.
- Ther rest of parameters are not sensitive.

From Test 1 and 2, surprisingly, only NS was found to be more sensitive than the other parameters with given ranges in Table 6.4. It also suggested that NS was inverse proportional to D-DCA's performance. This finding was unexpected and contradictory to the previous findings, shown in Table 6.1, which suggested they were proportional to each other. A possible explanation for this might be that the NS affects performance dependently with other parameter, although, all parameters are independent. It suggested that the ratio of DC/NS as a new parameter could affect the performance. For this, Test 3 was conducted. The results suggested that the ratio of DC/NS was slightly more sensitive than the others and it was inverse proportional to performance. This will be verified to be the case in the next section.

Evaluation

To evaluate the findings, 4 sets of tests were conducted. E1 was varying NS parameter. The results from E1, shown in Table 6.10, suggest that there are significant changes in performances, by varying NS from 10 to 50. But, no significant changes, once above 50 and below 1000. There is a slight drop in performance, where NS was 1000. This verifies the findings in the preliminary test, shown in Table 6.1, which is that NS is proportional to performance. This also disproves the findings in Latin Hypercube Test 1 and 2, where it suggests that NS is inverse proportional to performance. Without it, however, Test 3 would not be conducted. E2 and E3 were designed to verify the findings in Test 3. By comparing the results from E2 and E3, in Tables 6.12 and 6.14, they suggest that:

- There is no significant difference on performances, if the ratio of DC/NS is constant, neglecting when small parameters were used.
- The performances are almost constant, despite that NS was increased.

To further verify the findings in Test 3, E4 was conducted. The results, shown in Table 6.16, suggest that the performances are almost constant when varying DC/NS as one parameter. This confirms the findings in Test 3, shown in Figure C.8, which is that DC/NS is a little sensitive to the performance. From Table 6.16, it also suggests that DC/NS is inverse proportional to performance. However, at the largest ratio of DC/NS, the performance was slightly reducing rather than improving. Nevertheless, from the results of E2, E3 and E4, it is apparent that the findings in Test 3 was correct.

6.5 Summary

This study has set out to evaluate and validate the sensitivity of Diagnostic Dendritic Cell Algorithm (D-DCA)'s parameters. It started with presenting the parameters under the test, which were Migration threshold (MT), Number of DC cells (DC), Number of random select cells (NS), Max life of cells (Life), Quantity T killer threshold (Tk) and Percentage threshold (Per). The preliminary findings with regards to parameters and performances were shown in Table 6.1. Then, how 3 sets of experiments were conducted was explained , which were Wilcoxon signed rank test, Latin Hypercube test and parameters evaluation test. The results from those tests were presented and discussed.

The following conclusions can be drawn from this study. The parameters of D-DCA, MT, DC, NS, Life, Tk and Per, are not sensitive to the performance individually. However, the ratio of DC/NS is slightly more sensitive than the others, but just a little. This is under the assumption that, all parameters are not in small numbers (below 50). In Table 6.16, the highlighted parameter is a sensible choice for these data sets and it will be used for next chapter for comparison with other methods.

Furthermore, DC and NS define the computational power consumption of the D-DCA, however, they are insignificant to the D-DCA's performance. This has suggested that the D-DCA can perform well without the increase of the computational cost. Hence, the D-DCA is suitable for resource limited robotic systems.

With the effects of parameters analysed, a comparison between Artificial Neural Network (ANN), Support Vector Machine (SVM) and D-DCA will be presented in Chapter 7. In this, why other method, i.e. K-nearest neighbour, was not considered will be explained and why D-DCA would be a better choice for real-time robotic system will be illustrated.

Chapter 7

Comparison analysis

Contents

Tables

7.1 Introduction

Comparison is often desirable when introducing a novel algorithm or method. However, the comparison is often difficult or impossible to undertake. This can be attributed to difficulty in finding comparable methods that can perform the required tasks. When introducing a novel method, it is often designed to solve a particular problem. Therefore, the data used might be unique, often all, if this was not the case, it would probably not be necessary to introduce the method in the first place. The unique data used will cause difficulty in finding a suitable method to compare with. Once a suitable method is found, a suitable quantitative measure is also needed. Once a suitable measure is found, then tuning parameters will be another challenge. With this in mind, the investigation of Diagnostic Dendritic Cell Algorithm (D-DCA)'s performance in comparison with Artificial Neural Network (ANN) and Support Vector Machine (SVM), will be presented in this chapter.

The data set used in this work is unique. This does not mean that they only exist within this work. The uniqueness appears as unbalanced, one-to-many relations and duplicated entries, as discussed in Section 2.4.1. This would cause some methods, for example K-nearest neighbours, to have difficulty in coping. The basic "majority voting" classification is that the classes with the more frequent examples tend to dominate the prediction of the new vector, as they tend to come up in the k nearest neighbours when the neighbours are computed due to their large number.[36] The results are shown in Appendix I confirm this. Here, K-nearest neighbour was able only to classify all data entries into one class, due to unbalanced data. For that reason, ANN and SVM have been chosen for the comparison purpose. ANN is one of the most popular methods in machine learning and has been suggested for fault diagnosis, in [26] and [88]. SVM has a reputation for non-linear and multi-class classification problems, where examples are described in [18]. For these reasons, it has been decided that these are the best methods to adopt for this investigation. To assess performance, Receiver Operating Characteristic (ROC) is used. It has been proved to be a suitable measuring tool for our study in the previous chapters.

This chapter begins by explaining how the experiments that have been conducted and results are presented; then, the findings are discussed and concluded at the end.

7.2 Experiments and results

The aim of this section is to explain the procedures of experiments conducted, and to present the results. It begins by describing the data sets used in this study and explains the necessary modification made. The ANN results, SVM results and the comparison results with D-DCA will be then presented.

7.2.1 Data sets

There were 8 training data sets generated from *Stage* (the same as used in Chapter 6), shown in Table 7.1. All of them were stuck-at fault. Data 1, 2, 3, 4 and 5 were *periodic faults*; Data 6, 7 and 8 were *permanent faults*. The parameters for data sets, such as component, stuck value, period and duration were randomly assigned.

The data set was then generated from *Stage*. For each scenario, data was recorded for 4000 time steps (about 6.5 minutes). For each time step, 8 IR sensor values, left/right wheel speed and location X/Y were recorded to form an input vector, shown in Table 5.1. Their value range is shown in Table 5.2. A "faulty" label was also included in the input vector, where '1' indicated that at the current time step, there was a fault within the system. An input vector was fed to D-DCA and then an output would be produced at each time step.

There were 5 test data sets, shown in Table 7.2, in which, test data 1 and 4 were randomly selected from training data sets. The parameters of 3 other test data sets were randomly generated and the data sets were recorded the same as described above.

To improve the results of ANN and SVM, when training, the data was modified by deleting duplicate data entries. This was because the significant number of duplicate data entries would cause the algorithm to have difficulty to converge in the training phase.

	faulty component	stuck at value	period (time steps)	duration (time steps)
training data-1	IR sensor 0	63%	75	48
training data-2	IR sensor 2	27%	778	742
training data-3	IR sensor 4	45%	878	481
training data-4	IR sensor 6	81%	186	181
training data-5	IR sensor 7	38%	841	515
	faulty component	stuck at value	fault starts at (time steps)	
training data-6	IR sensor 2	51%	1718	
training data-7	IR sensor 6	34%	3951	
training data-8	IR sensor 7	32%	2056	

Table 7.1: Training data set

	faulty component	stuck at value	period (time steps)	duration (time steps)
test data-1	IR sensor 4	45%	878	481
test data-2	IR sensor 7	71%	774	149
test data-3	IR sensor 7	93%	229	153
	faulty component	stuck at value	fault starts at (time steps)	
test data-4	IR sensor 2	51%	1718	
test data-5	IR sensor 7	2%	1519	

Table 7.2: Test data set

7.2.2 ANN results

The Feed Forward ANN used in this study was implemented using PyBrain [74]. There were three ANNs constructed with a different number of hidden layer neurons. All of the ANNs consisted of 3 layers, an input, a hidden and an output layer. Each ANN had 12 inputs, and 11 outputs neurons. All the neurons between adjacent layers were fully connected. The inputs fed into ANN were their raw data recorded from simulation. The 11 outputs corresponded to the components which were IR sensor 0-7, left and right wheels and location.

There was no guidelines for how many one should use. The author [47] has discussed how the number of hidden units affects the bias/variance trade-off. It might depend on such as, the number of training cases, the architecture, regularization, etc. and it was also suggested that one should just try many configurations. During the initial experiments, the author here has tried various settings of hidden neurons as ranged between 2 and 100. The findings from those experiments suggest that the more hidden neurons would not benefit on the training convergence rate and ANN's accuracy. In general, it is suggested the number of hidden neurons would not need to exceed twice of the input neurons. Therefore, although no comprehensive tests will be performed on ANN's parameters, a few settings were chosen to be $\{5, 18, 24\}$.

The ANNs were then trained and optimized with the training data sets, shown in Table 7.1, using back-propagation. Each ANN was trained until convergence. To avoid trapping in local maxima, if a best was found, then the ANN would try 5 (this was set) more training epochs[1] before giving up. The ANN was trained using paired input data and actual fault. The input data was the same as input vector for D-DCA (shown in Table

[1]An epoch is a training run with all training data.

Number of hidden nodes	test data-1	test data-2	test data-3	test data-4	test data-5
	(0.43,0.02)	(0.54,0.01)	(0.0,0.04)	(0.87,0.0)	(0.0,0.06)
	(0.42,0.02)	(0.59,0.01)	(0.0,0.04)	(0.87,0.0)	(0.0,0.06)
	(0.44,0.06)	(1.0,0.02)	(0.52,0.04)	(0.88,0.01)	(0.0,0.06)
18	(0.43,0.06)	(1.0,0.02)	(0.52,0.04)	(0.87,0.01)	(0.0,0.06)
	(0.43,0.02)	(0.55,0.02)	(0.0,0.04)	(0.87,0.0)	(0.0,0.06)
	(0.42,0.02)	(0.53,0.02)	(0.0,0.04)	(0.85,0.0)	(0.0,0.06)
	(0.42,0.02)	(0.57,0.01)	(0.0,0.04)	(0.84,0.0)	(0.0,0.06)
5	(0.4,0.06)	(1.0,0.01)	(0.54,0.04)	(0.85,0.02)	(0.0,0.06)
	(0.43,0.06)	(1.0,0.02)	(0.54,0.04)	(0.86,0.02)	(0.0,0.06)
	(0.43,0.02)	(0.6,0.02)	(0.0,0.04)	(0.85,0.0)	(0.0,0.06)
	(0.43,0.02)	(0.64,0.02)	(0.0,0.04)	(0.87,0.0)	(0.0,0.06)
	(0.4,0.02)	(0.47,0.01)	(0.0,0.03)	(0.88,0.0)	(0.0,0.06)
	(0.46,0.06)	(1.0,0.02)	(0.52,0.04)	(0.9,0.01)	(0.0,0.06)
24	(0.4,0.06)	(1.0,0.01)	(0.53,0.04)	(0.87,0.01)	(0.0,0.06)
	(0.41,0.02)	(0.48,0.01)	(0.0,0.03)	(0.89,0.0)	(0.0,0.06)

Table 7.3: ANN results

5.1) and actual fault was a vector, similar to the output vector of D-DCA shown in Table 5.3, in which, 11 components corresponded to each output neuron. The output from ANN was a vector of 11 components, which were decimal numbers ranged between [0, 1].

Once the ANNs were trained, they were then applied to test data sets and the results were in the form of True Positive Rate (TPR) and False Positive Rate (FPR), shown in Table 7.3, in which, the highlighted were selected as the best performances to compare in a later section. To be consistent, the test data was not modified. For a data set, there were 4000 data entries. Each data entry was fed to the ANN and the largest output of those 11 components was considered as “faulty”, if “Error detection” indicated there was a fault.

7.2.3 SVM results

The SVMs used in this study were implemented using PyML [24]. The SVMs were implemented as multi-class classifiers using one against the rest winner takes all strategy [42], with Linear, Gaussian and Polynomial kernels. The parameters used were suggested by the initial experiments and their effects can be found in Appendix G.

Each SVM was trained and optimized with the training data sets, shown in Table 7.1. Once the SVM was trained, it then applied to the test data sets, shown in Table 7.2. The results were in the form of TPR and FPR, shown in Table 7.4. The highlighted were selected as the best performances to compare in a later section.

7.2.4 DDCA comparison results

The D-DCA was applied to the test data sets using the parameters shown in Table 7.5. For each setting, experiments were performed 20 times and the results are shown in Table 7.6 were the medians of 20 runs. Its performance on the training data sets can be found in Table 6.15 (highlighted). The best performances of ANN's and SVM's are included in Table 7.6, where the highlighted is the best achieved result for each data set.

To determine significance on differences, a set of Wilcoxon signed rank tests were performed. In addition to the 5 test data sets, 20 more test data sets were generated from simulation, all of which were a mix of *permanent fault* and *periodic fault* for IR sensor 0, 2, 4, 6 and 7. The SVMs and ANNs were applied to 25 data sets once, as their performances were consistent without much variation for individual data set previously.

	test data-1	test data-2	test data-3	test data-4	test data-5
Gau-0.1-0.01	(0.46,0.06)	(0.0,0.08)	(0.0,0.1)	(0.31,0.07)	(0.0,0.1)
Gau-0.1-0.1	(0.62,0.05)	(0.09,0.07)	(0.0,0.1)	(1.0,0.04)	(0.0,0.09)
Gau-0.1-1	(0.73,0.03)	(0.0,0.05)	(0.0,0.07)	(1.0,0.03)	(0.0,0.08)
Gau-0.1-10	(1.0,0.01)	(0.42,0.02)	(0.0,0.07)	(1.0,0.01)	(0.0,0.07)
Gau-100-0.01	(0.95,0.01)	(0.42,0.01)	(0.0,0.04)	(1.0,0.01)	(0.89,0.02)
Gau-100-0.1	(0.78,0.03)	(0.0,0.05)	(0.0,0.09)	(0.88,0.04)	(0.0,0.1)
Gau-100-1	(0.52,0.07)	(0.15,0.09)	(0.0,0.1)	(0.89,0.05)	(0.0,0.1)
Gau-100-10	(0.93,0.01)	(0.4,0.04)	(0.0,0.06)	(1.0,0.01)	(0.0,0.08)
Gau-10-0.01	(0.97,0.01)	(0.01,0.01)	(0.0,0.05)	(1.0,0.02)	(0.0,0.07)
Gau-10-0.1	(0.84,0.02)	(0.15,0.04)	(0.0,0.08)	(1.0,0.03)	(0.0,0.1)
Gau-10-1	(0.82,0.04)	(0.17,0.06)	(0.0,0.1)	(1.0,0.03)	(0.0,0.1)
Gau-10-10	(1.0,0.0)	(0.45,0.01)	(0.0,0.04)	(1.0,0.0)	(0.0,0.06)
Gau-1-0.01	(0.6,0.04)	(0.0,0.07)	(0.0,0.1)	(0.56,0.05)	(0.0,0.1)
Gau-1-0.1	(0.94,0.01)	(0.0,0.03)	(0.0,0.08)	(1.0,0.02)	(0.0,0.08)
Gau-1-1	(0.97,0.01)	(0.0,0.03)	(0.0,0.07)	(1.0,0.02)	(0.0,0.09)
Gau-1-10	(1.0,0.0)	(0.47,0.01)	(0.0,0.04)	(1.0,0.0)	(0.0,0.07)
linear-0.1	(0.97,0.01)	(0.25,0.03)	(0.0,0.06)	(0.49,0.04)	(0.0,0.09)
linear-1	(0.95,0.01)	(0.22,0.01)	(0.0,0.04)	(0.83,0.02)	(0.91,0.01)
linear-10	(0.9,0.01)	(0.51,0.01)	(0.0,0.03)	(0.74,0.02)	(0.9,0.01)
linear-100	(0.92,0.01)	(0.63,0.01)	(0.0,0.02)	(0.75,0.02)	(0.91,0.01)
linear-1000	(0.95,0.01)	(0.66,0.01)	(0.0,0.02)	(0.78,0.02)	(0.91,0.01)
pol-0.1-deg-2	(0.98,0.0)	(0.41,0.01)	(0.0,0.04)	(0.88,0.02)	(0.89,0.02)
pol-0.1-deg-3	(0.99,0.0)	(0.24,0.01)	(0.0,0.03)	(0.79,0.02)	(0.0,0.01)
pol-0.1-deg-4	(1.0,0.0)	(0.28,0.01)	(0.0,0.02)	(0.88,0.01)	(0.0,0.01)
pol-0.1-deg-6	(1.0,0.0)	(0.32,0.0)	(0.0,0.01)	(1.0,0.0)	(0.0,0.06)
pol-100-deg-2	(0.99,0.0)	(0.15,0.0)	(0.0,0.02)	(0.86,0.01)	(0.0,0.0)
pol-100-deg-3	(1.0,0.0)	(0.11,0.0)	(0.0,0.01)	(0.97,0.0)	(0.0,0.06)
pol-100-deg-4	(1.0,0.0)	(0.2,0.0)	(0.0,0.01)	(1.0,0.0)	(0.0,0.06)
pol-100-deg-6	(1.0,0.0)	(0.23,0.01)	(0.0,0.01)	(1.0,0.0)	(0.0,0.06)
pol-10-deg-2	(0.98,0.0)	(0.36,0.0)	(0.0,0.02)	(0.8,0.02)	(0.0,0.01)
pol-10-deg-3	(1.0,0.0)	(0.27,0.0)	(0.0,0.02)	(0.88,0.01)	(0.0,0.0)
pol-10-deg-4	(1.0,0.0)	(0.31,0.0)	(0.0,0.01)	(0.99,0.0)	(0.0,0.06)
pol-10-deg-6	(1.0,0.0)	(0.35,0.0)	(0.0,0.01)	(1.0,0.0)	(0.0,0.06)
pol-1-deg-2	(0.97,0.0)	(0.29,0.01)	(0.0,0.02)	(0.77,0.02)	(0.0,0.01)
pol-1-deg-3	(0.99,0.0)	(0.31,0.0)	(0.0,0.02)	(0.83,0.01)	(0.0,0.01)
pol-1-deg-4	(1.0,0.0)	(0.24,0.0)	(0.0,0.02)	(0.89,0.01)	(0.25,0.0)
pol-1-deg-6	(1.0,0.0)	(0.28,0.0)	(0.0,0.01)	(1.0,0.0)	(0.0,0.06)

Table 7.4: SVMs results

Table 7.5: Parameters for DDCA

Parameter	Value
Migration threshold	20
Number of DC cells	100
Number of random select cells	200
Max life of cells (With death) (time steps)	100
Quantity T killer threshold	120
Percentage threshold	95%

	test data-1	test data-2	test data-3	test data-4	test data-5
DDCA	(0.65,0.07)	(0.41,0.02)	(0.07,0.11)	(0.93,0.12)	(0.96,0.13)
ANN 18	(0.44,0.06)	(1.0,0.02)	(0.52,0.04)	(0.88,0.01)	(0.0,0.06)
ANN 5	(0.43,0.06)	(1.0,0.02)	(0.54,0.04)	(0.86,0.02)	(0.0,0.06)
ANN 24	(0.46,0.06)	(1.0,0.02)	(0.52,0.04)	(0.9,0.01)	(0.0,0.06)
SVMs	test data-1	test data-2	test data-3	test data-4	test data-5
Gaun-100-0.01	(0.95,0.01)	(0.42,0.01)	(0.0,0.04)	(1.0,0.01)	(0.89,0.02)
lin-1000	(0.95,0.01)	(0.66,0.01)	(0.0,0.02)	(0.78,0.02)	(0.91,0.01)
pol-0.1-deg-2	(0.98,0.0)	(0.41,0.01)	(0.0,0.04)	(0.88,0.02)	(0.89,0.02)

Table 7.6: DDCA comparison results

The D-DCA was applied to the data sets 20 times and the median were chosen to represent the result for each data set. The test hypotheses which were addressed are shown in Table 7.7. They are whether there was any difference within the performances of each method against D-DCA's. The results of critical values are from Wilcoxon tests with paired each method's performance (TPR and FPR) and D-DCA's, shown in Table 7.8. To illustrate the differences, the performances' TPR of 25 data sets with each method are shown as box plots in Figure 7.1, FPR in Figure 7.2 and the Euclidean distances between the performance and the perfect one (1, 0)[2] in Figure 7.3. The 25 test data sets details can be found in Appendix Table H.1.

[2](TPR = 1, FPR = 0)

[3]A description of distance to (1, 0) as a "goodness" measurement can be found in Appendix B.

Table 7.7: Table of null hypotheses for comparison tests

Null hypothesis	Description
H7	The D-DCA's performance is no different from the ANN's with 5 hidden neurons for given data sets shown in Table 7.2.
H8	The D-DCA's performance is no different from the ANN's with 5 hidden neurons for given data sets shown in Table 7.2.
H9	The D-DCA's performance is no different from the ANN's with 5 hidden neurons for given data sets shown in Table 7.2.
H10	The D-DCA's performance is no different from the SVM's with Gaussian 0.01 kernel and soft margin 100 for given data sets shown in Table 7.2.
H11	D-DCA's performance is no different from the SVM's with Linear kernel and soft margin 1000 for given data sets shown in Table 7.2.
H12	D-DCA's performance is no different from the SVM's with Polynomial degree of 2 kernel and soft margin 0.1 for given data sets shown in Table 7.2.

DDCA against	Critical value		
	TPR	FPR	Distance to (1, 0)[3]
ANN 5	120	24	121
ANN 18	121	25	118
ANN 24	118	26	117
SVM Linear-1000	99	29	99
SVM Polynomial 0.1 deg 2	80	52	91
SVM Gaussian 100-0.01	134	29	134

Table 7.8: DDCA comparison Wilcoxon results

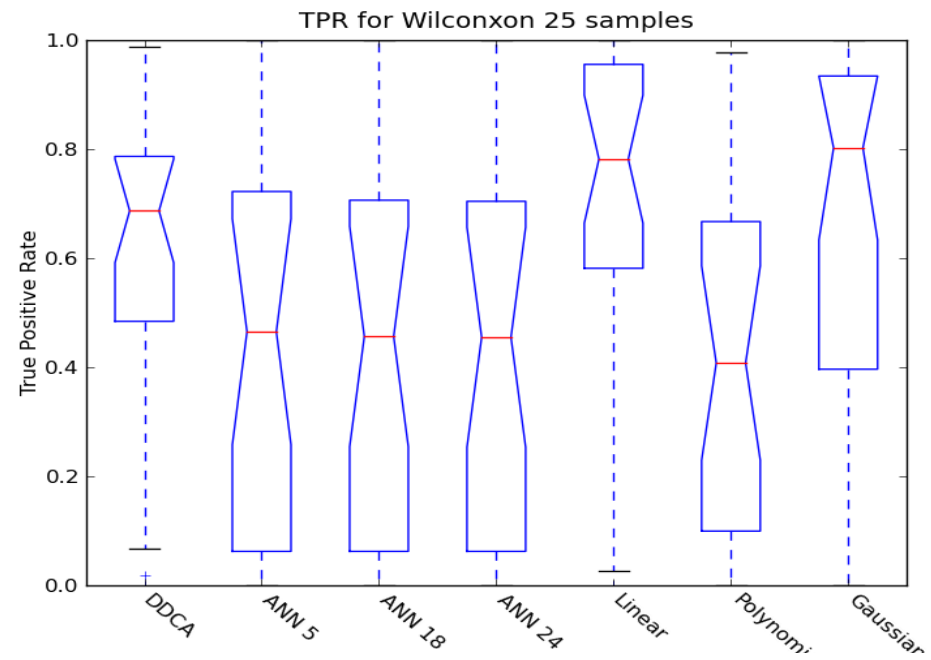


Figure 7.1: TPR result for Wilcoxon 25 samples test

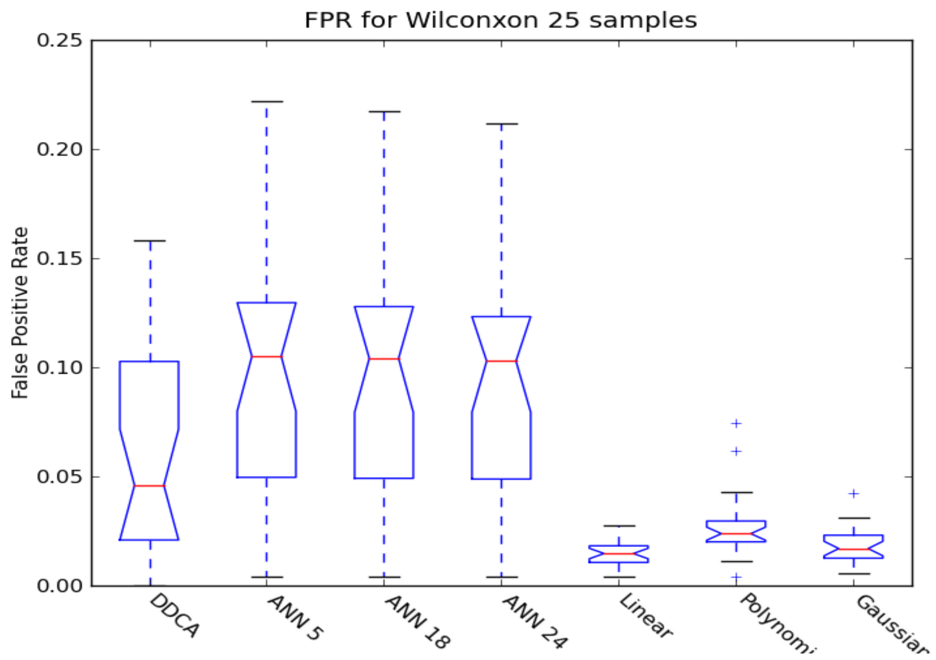


Figure 7.2: FPR result for Wilcoxon 25 samples test

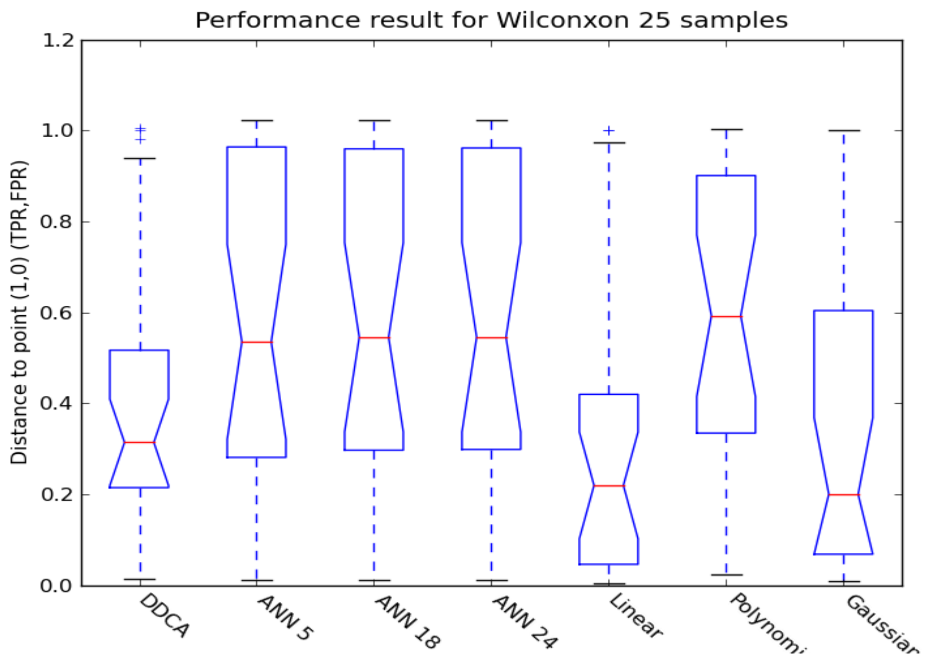


Figure 7.3: Distance to (1, 0) (TPR, FPR) result for Wilcoxon 25 samples test

7.3 Discussion

Data sets

The design of the data sets was randomly chosen and equally fair on all methods. The reason why only 8 training data sets were used, was to present a real world problem, where there might not be much information available. Although the simulation set up was static, there would be near infinite possibilities of input combinations. The availability of information was not guaranteed. One might argue that this is not fair on any learning method. However, this is why on-line diagnosis is rarely done. The dynamic environment will require a learning method to be retrained. This is too expensive for a resource limited robot. In this study, ANNs and SVMs were trained with the training data sets and applied to the test data sets. The test data 1 and 4 were selected from the training data sets to illustrate their performance on training data sets.

It was decided that it is best to delete the duplicated data entries, when training ANN and SVM. From earlier experiments, it was found that the duplicated data entries would cause those methods to have difficulty converging, resulting in poor performances on training data sets, especially for ANNs. However, in removing the duplicate data entries, only gave significantly better performance for SVM, but not for ANN's. This will be explained individually in the next sections.

ANN results

The result, shown in Table 7.3, was that the ANN with different number of hidden neurons applied to the test data sets. It was in the form of TPR and FPR. There were 5 ANNs trained for each parameter setting. Poor results can be found for the test data 1, where only slightly above 40% TPR were achieved for all settings. One of reasons might be that some of the data used was one to many related. That is one input feature can be associated with different classes. This would cause ANN to be confused during training, where it was trying to balance the inputs were fed and the classes association (whether "faulty" or "not faulty"). Thus, poor performance was found for the test data 1.

The result also shows that for the same data set and parameter setting, but different tests, there were significant variation (more than 50%) on performances. This can be found in Table 7.3 for data 2 and 3. This might suggest that the ANNs were trained to a local maxima point which was in favour to one particular data set. Caution might be taken for any future use.

Overall, the result shows that the ANN performed poorly on data set 2, 3 and 5, and better on the data 1 and 4. However, data 1 and 4 were from the training data sets. This might suggest that ANN was not suitable for this work. At least, it should not be used with the data sets used in this work or in this set up.

SVM results

The result, shown in Table 7.4, was that the SVM with different kernels applied to the test data sets. It shows that the SVMs performed solidly on data sets 1 and 4, where perfect classification (1, 0) could be found. This suggests that SVM is a better choice than ANN for this work, if there are sufficient training data.

However, for the test data 2, 3 and 5, the SVMs did not perform so well, with exception for linear kernel on the test data 5. This might be due to the fact that no sufficient data was trained on relevant sensors.

Overall, the result shows that there is no one kernel or parameter setting which would fit for all 5 test data sets. However, the best overall performance is selected from each kernel and will be used in next section for comparison.

Comparison results

The best performance from each ANN and SVM was selected from the experiments, to compare with the D-DCA's median performance of 20 runs, shown in Table 7.6. The highlighted were the best for each data set. As expected, there was no one method which would dominate the performance. However, for the test data set 1 and 4, which were selected from the training data sets, the result indicates that the SVM outperformed the others. For the test data set 2, the ANN was the best, although there were no consistencies within each setting, shown in Table 7.3. For the test data 5, the D-DCA and SVMs had similar performances, where the ANNs had failed. For the test data 3, none of the methods could be considered performed well. Unfortunately, the result was not be able to show one method was better than the other. To further compare, a statistical analysis was used, the Wilcoxon signed rank test.

Table 7.7 lists the hypotheses addressed. They are if there was no difference from the D-DCA's performance with the others. The results from Wilcoxon test are shown in Table 7.8. The single most striking observation to emerge from the comparison data was that no hypotheses could be rejected, with regards to TPR, expecting there was a difference on TPR between SVM polynomial soft margin 0.1 at degree of 2 and D-DCA with confidence interval of 95%[4]. The result of TPR also indicates that all hypotheses can be rejected with a confidence interval of 99%[5]. However, if using the distance to the perfect performance (1, 0) to measure, no significant differences were found between ANN, SVM and D-DCA.

To illustrate the differences, 5 sets of box plots were produced, TPR in Figure 7.1, FPR in Figure 7.2 and the distance to perfect in Figure 7.3. By focusing on the interquartile range from the box plots in Figure 7.1, it shows that the D-DCA has the least variation in TPR, but SVM linear and Gaussian have higher median values. Now, by turning to Figure 7.2, it is apparent that the SVMs have the least FPR. By using the distance to perfect performance to measure, shown in Figure 7.3, it indicates that the SVM linear has the least distance and better performance. Although, the SVM Gaussian has a lower median value, the interquartile range is spreading slight larger than SVM linear and D-DCA. In all figures, they indicate that the D-DCA performed better than the ANNs.

In a short conclusion of the Wilcoxon tests, the evidence from this

[4]The critical value is less than 89 for confidence interval of 95% from Wilcoxon critical value table for 25 samples, shown in Appendix Table D.1

[5]The critical value is less than 68 for confidence interval of 99% from Wilcoxon critical value table for 25 samples, shown in Appendix Table D.1

study suggests that there is no significant difference between the performances, as there were no consistent results for all the performance measurements. However, slight difference can be found in box plots, where they suggest that the SVM linear is better suited for this work and the D-DCA is slightly behind.

7.4 Summary

This chapter set out to compare the performances between the ANN, the SVM and the D-DCA. The data sets used in this study were explained and the set up and procedure of experiments have been described. A series of experiments have been conducted and their results have been presented in this chapter.

The results of the experiments presented in this chapter provide a comparison between the performances of the ANN, the SVM and the D-DCA. The evidence from the results suggests that there is no statistically significant difference between the performances. But, the SVM linear with soft margin 1000 performed better than the others for the given data sets[6]. Its performance is followed by the D-DCA with parameters setting shown in Table 7.5.

However, the evidence also illustrates the limitation of a learning method, where its performance is dependent on the training data, for example ANN. On the contrary, D-DCA requires no training and "learns on the run". The results also indicate the robustness of D-DCA's performance, where it has the least variation in TPR of all given data sets.

In conclusion, the results suggest that ANN, SVM and D-DCA gave approximate the same results.

[6]The 25 data sets used can be found in Appendix Table H.1.

Chapter 8

Conclusion

Contents

8.1 Summary

This thesis began with the research hypothesis:

"An immune-inspired system can be successfully deployed in a resource constrained robotic system to diagnose the cause

of faults, in an on-line manner and accurately with reasonable correct response time."

In order to answer this hypothesis, the challenges for fault diagnosis for robotic systems have been outlined and several plausible methods have been introduced, such as Artificial Neural Network (ANN) and Support Vector Machine (SVM) in Chapter 2. The immune system has been revisited and an immune-inspired solution for fault diagnosis has been introduced, as the cooperation between the dendritic cells and T cells. The the original Dendritic Cell Algorithm (the original DCA) within the field of Artificial Immune System has been introduced, in Chapter 3. This has led to the development of a novel immune inspired fault diagnosis algorithm, named Diagnostic Dendritic Cell Algorithm (D-DCA), has been presented in Chapter 4, which extends and modifies the dendritic cell model from the the original DCA. Then, a comparison between two D-DCA approaches and a sensitivity analysis of the D-DCA's parameters have been undertaken, in Chapter 5 and 6. And finally, a comparison of performances between ANN, SVM and D-DCA has been presented in Chapter 7.

8.2 Contribution

This thesis proposes a novel immune inspired fault diagnosis algorithm. This algorithm further extends the abstraction dendritic cell model from [50] and applied to robot fault diagnosis. This thesis includes the first application of AIS to robot fault diagnosis, which has since become an active area of research[84][28][86]. This section will review the main outcomes of the processes, outlined in Section 1.1 and draw conclusions based on the hypothesis stated.

8.2.1 Challenges

Robotic challenges

- The robotics system used for this work is autonomous and real time, the fault diagnosis process has to be on-line and there is only limited resource, as in computational power. These have set the basic requirements for the fault diagnosis system: it has to be a real time system, in that it has to produce an output within a time limit; it has to respond quickly and accurately, otherwise it would not be of much use; it has to consume as little computational power as possible, such as memory and CPU time.

- The prior knowledge is partial. This implies the fault diagnosis system has to be able to cope with unseen faults or data

- The robotics environment is dynamic. This implies the fault diagnosis system has to be adaptable to changes, such as environment changes or robot's internal changes (objective/ task changes).

Fault diagnosis challenges

- Data size and quality. A typical challenge for classification methods is the size of available data sets and how well they represent the whole system. Insufficient data will lead to a classification method that performs undesirably. This was further explored in Chapter 7, where a comparison was presented.

- The distinction between different faults. If faults are not distinctive, but somehow independent, it would cause any diagnosis system to be confused, for example, a robot running in a circle as illustrated in Chapter 2.

8.2.2 A novel algorithm

An immune inspired algorithm, named the Diagnostic Dendritic Cell Algorithm (D-DCA), was presented in Chapter 4. The D-DCA was designed for a resource limited robotic system to diagnose faults in real time. By applying the conceptual framework while developing the D-DCA, several "meta-questions" were asked, addressing *diversity, interaction* and *scale.* A number of key characteristics have been identified, such as tolerance to noise, multiple faults diagnosis, robustness and efficiency.

However, D-DCA is not a classification algorithm. It should never be used to classify a static data set, although later, classification analysis method (Receiver Operating Characteristic (ROC)) could be used. The sequence of data entries will affect the D-DCA result, as D-DCA is constantly analysing the current states of the system; what is happening now will have more effect then what has happened in the past. Hence, the sequence of data entries will affect its result.

8.2.3 Analysis

The following conclusions can be drawn from the feasibility tests. The results have shown D-DCA can be used as fault diagnosis with reasonable success. However, it is also suggested that D-DCA can diagnose incorrectly for certain data sets, where data is labelled as "normal" most of the time. The results clearly have shown the "with death" is better than the "No death" approach. This exercise served as a base for future studies and more importantly, it has shown ROC could be used as a measuring tool to assess performance.

The following conclusions can be drawn from parameter sensitivity analysis. The parameters of D-DCA, Migration threshold (MT), Number of DC cells (DC), Number of random select cells (NS), Max life of cells (Life), Quantity T killer threshold (Tk) and Percentage threshold (Per), are not sensitive to the performance individually. However, the ratio of DC/NS is slightly more sensitive than the others, but just a little. This is under the assumption that, all parameters are not in small numbers (below 50). In Table 6.16, the highlighted parameter is a sensible choice for these data sets and it was used for a comparison with other methods.

Furthermore, DC and NS define the computational power consumption of the D-DCA, however, they are insignificant to the D-DCA's performance. This has suggested that the D-DCA can perform well without the increase of the computational cost. Hence, the D-DCA is suitable for resource limited robotic systems.

8.2.4 A comparison

The results of the comparison experiments presented in Chapter 7 provide a comparison between the performances of the ANN, the SVM and the D-DCA. The evidence from the results suggests that there is no significant difference between the performances. But, the SVM linear with a soft margin of 1000 gives better performance than the others for the given data sets[1]. Its performance is followed by the D-DCA with parameters setting shown in Table 7.5.

However, the evidence also illustrates the limitations of an learning method, where its performance is dependent on the training data, for example ANN. On the contrary, D-DCA requires no training and "learns" during operation. The results also indicate the robustness of D-DCA's performance, where it has the least variation in TPR of all given data sets.

[1]The 25 data sets used can be found in Appendix Table H.1.

In conclusion, the results suggest that ANN, SVM and D-DCA gave approximate the same results, but the D-DCA is very efficient in terms of implementation and requires no training. Hence, it is more suitable for a real-time system, such as a robotic system.

However, this work is not the 'final word' for the D-DCA and the next section outlines future work on both this algorithm, and for its applications.

8.3 Future work

8.3.1 Towards a truly immune inspired fault diagnosis

The D-DCA is currently only inspired by the processes of the dendritic cells and T cells to achieve diagnosis. Firstly, the diagnosis decision is metaphorically made by an immune response towards an antigen, where there are more T killer cells of this antigen. This abstraction is at a very high level. It excludes the complicated signalling and binding processes, which ensure the robust functioning of the immune system as a whole. Secondly, an immune response does not just involve dendritic cells and T cells. There are many more types of cell, such as antibodies, which provide a long term immunity, and the activation of B cells that produce antibodies. The collaboration between different cells ensures the immune system provides an effective and efficient protection against diverse threats to its host. Finally, the design of the *Pre-DCA* part of the D-DCA is trying to recreate the apotheosis and necrosis processes. Again, it has been implemented by using a rule based system. As argued by [83], the immune system does not work in isolation. Therefore, to design a truly immune inspired fault diagnosis system, more exploration of the natural immune system and abstraction from it to useful artificial systems has to be taken.

8.3.2 Applying to real robots and beyond

The D-DCA is designed for robotics system. There is one particular type of robotics systems in mind, named swarm robotics[71], such as the SYMBRION[19] robots. Here, a swarm consists of hundreds of small robots, where they have the ability to aggregate into one or many symbiotic organisms and collectively interact with the physical world via a variety of sensors and actuators. One of the grand challenges for SYMBRION is “to survive 100 days with 100 robots”, where the robots will require to charge themselves, adapt to changes in the environment and tolerate to faults. Such a system proposes many challenges for fault diagnosis, such as listed in this thesis. It would be interesting to apply the D-DCA to such a complex and dynamic system to diagnose fault.

The D-DCA is not just for robotics systems, although it has been designed for and tested on simulated robotics systems. Its key characteristics, such as low computational consumption and efficiency, might suggest that it would be suitable for any real time embedded systems[55]. For example, one could design a portable fault diagnosis device, which can diagnose faults in real time by monitoring a much larger system’s states, of even to diagnose fault for a part of the system. Such a device would help engineers to understand the fault, i.e. it could act as an autonomous diagnosis device. This device could be used, for example, on a large cruiser ship, where crack on ship body can lead catastrophic consequences and often difficult to pinpoint the location when the ship is sailing. The diagnosis devices can be deployed within the ship in various locations. They act as an early warning system but with low cost. There are many possibilities for D-DCA and thus there is a potential for the D-DCA in a diverse range of applications.

Appendix A

Player Stage

Player/Stage [2] provides Open Source tools that simplify controller development, particularly for multiple robot, distributed-robot, and sensor network systems. The Player/Stage project was originated at the USC(University of Southern California) Robotics Research Lab in 1999 to address an internal need for interfacing and simulation for multi robot system. It has since been adopted, modified and extended by researchers around the world. Player/Stage offers a combination of transparency, flexibility and speed that makes it the most useful robot development environment available.The Player robot server is probably the most widely used robot control interface in the world. Its simulation back ends, Stage and Gazebo, are use.d worldwide.

The project provides the Player robot device server and the Stage multiple robot simulator, plus supporting tools and libraries. Player provides a clean and simple interface to the robot's sensors and actuators. Control programs (e.g obstacle avoidance) talk to Player, read data from sensors, and writes commands to actuators. Stage provides a population of simulated robots and sensors operating in a two-dimensional bit mapped environment. The devices are accessed through Player, as if they were real hardware. Player and Stage communicate using a standard network protocol (TCP/IP).

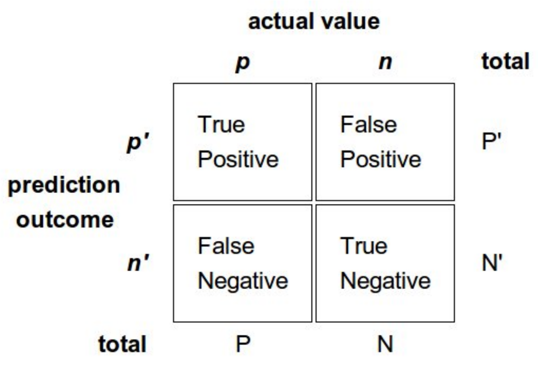
Appendix B

Receiver Operation Characteristic curve

B.1 Confusion matrix

Confusion matrix is used to measure the performance of a classifier. It uses the actual result in comparison with the predicted result and works out the parameters in the confusion matrix, True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN), shown in Figure B.1.

We begin by considering classification problems using only two classes.



		actual value		
		p	*n*	total
prediction outcome	*p'*	True Positive	False Positive	P'
	n'	False Negative	True Negative	N'
	total	P	N	

Figure B.1: Confusion matrix[7]

Formally, each instance is mapped to one element of the set $\{p', n'\}$ (positive and negative classes) as "predicted outcome". A classification model (or classifier) is a mapping of instances to one element of the set {p, n}. Given a classifier and an instance, there are four possible outcomes. If it maps to "p" and "p'", then it counts as a TP; if "p" and "n'", then FN; if "n" and "p'", then FP and if "n" and "n'", then TN. Given a classifier and a set of instances, a two by two confusion matrix can be constructed to represent the disposition of the set of instances.[43]

B.2 ROC

A Receiver Operating Characteristic (ROC), or simply ROC curve, is a graphical plot of statistical measures of performance of a system or binary classifier. It plots true positive rate (TPR) vs false positive rate (FPR), where TPR is the fraction of true positives out of the positives and FPR is the fraction of false positives out of the negatives. For a binary classifier, ROC analysis provides a tool to visually select possible optimal models, as its discrimination threshold is varied. It achieves this by plotting the "benefit (TPR) against the "cost" (FPR). The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battle fields, also known as the signal detection theory, and was soon introduced in psychology to account for perceptual detection of stimuli. ROC analysis since then has been used in medicine, radiology, and other areas for many decades, and it has been introduced relatively recently in other areas like machine learning and data mining.[79] using deterministic

B.3 How to interpret ROC

Figure B.2 is an illustration of how a possible optimal model can be select using ROC cure and how comparison can be made. A ROC space is defined by FPR and TPR as x and y axes respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Each result or one instance of a confusion matrix represents one point in the ROC space.[79]

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% true positives and no false positives. The (0, 1) point is also called a perfect classification. A completely random guess would give a point along a diagonal line (the so-called line of no-discrimination) from the left bottom to the top right corners. An intuitive example of random guessing is a decision by flipping coins (head or tail).

The diagonal divides the ROC space. Points above the diagonal represent good classification results, points below the line poor results. However, the ROC space is symmetrical along the diagonal line. The output of a poor predictor could simply be inverted to obtain points above the line. For example, in Figure B.2, P2 (0.8, 0.2) is at the lower bottom of the space and it could be considered as the same performance of P1 (02, 08), if its predictive decisions is reversed ({p, n} instead of {n, p}). This might be a bit confusing, but within this work there is no comparison between lower and upper ROC space. There is only comparison within the upper space, such as P1 (0.2, 0.8) and P3 (0.4, 0.6). As P1 has lower FPR (0.2) and higher TPR (0.8), P1 performs better than P3. In fact, the distance between point on ROC space and the perfect classification point (0, 1) measures the performance. The shorter the distance the better the performance is. In this case, the distance between the perfect

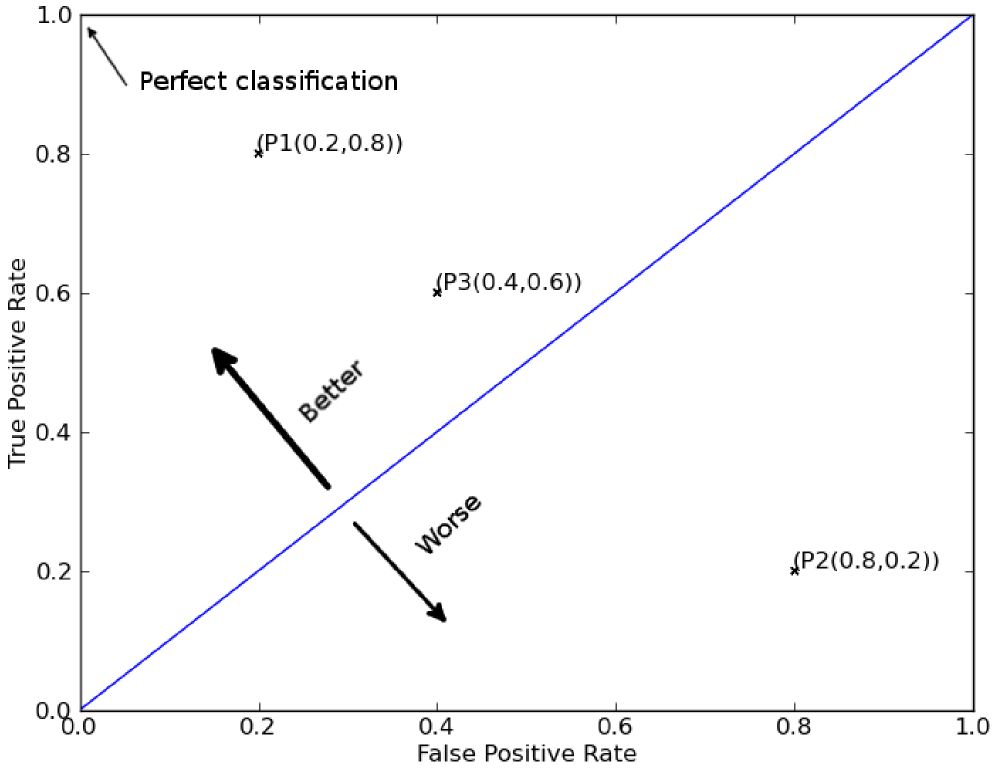


Figure B.2: A ROC example, modified from [7]

and P1 is 0.28, whereas distance for P2 is 0.57. Therefore, P1 performs better than P2. If points are distant to the perfect point as the same, they performs the same theoretically. However, this might be argued differently as some would prefer lower FPR vs TPR, and vice versa.

Appendix C

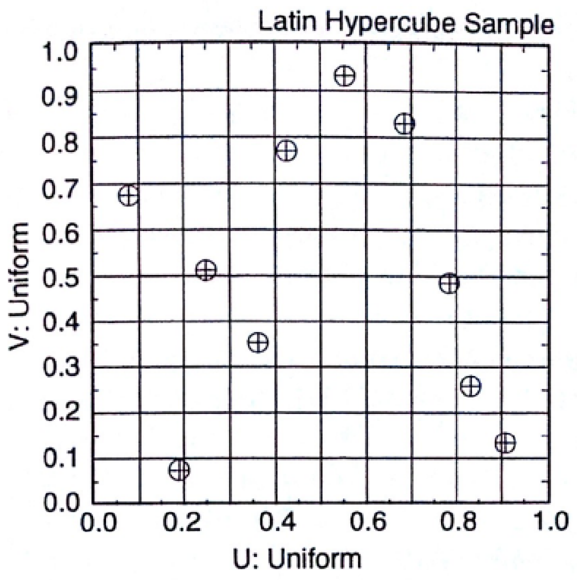
Latin Hypercube and Sensitivity analysis

The following description of Latin hypercube sampling is cited from [72]. Latin hypercube sampling ensures the full coverage of the range of each variable is divided into n_{LHS} intervals of equal probability and one value is selected at random from each interval. The n_{LHS} values thus obtained for x_1 are paired at random and without replacement with the n_{LHS} values obtained for x_2. These n_{LHS}pairs are combined in a random manner without replacement with the n_{LHS} value of x_3 to form n_{LHS} triples. This process is continued until a set of n_{LHS} n_X-tuples is formed. These n_X-tuples are of the form

$$x_k = [x_{k1}, x_{k2}, ..., x_{kn_X}], K = 1, ..., n_{LHS} \tag{C.1}$$

and constitute the Latin hypercube sampling. An illustration is shown in Figure C.1 between a random and Latin Hypercube sampling.

Desirable features of Latin hypercube sampling[80] include unbiased estimates for means and distribution functions and dense stratification across the range of each sampled variable[66]. In particular, uncertainty and sensitivity analysis results obtained with Latin hypercube sampling have been observed to be quite robust even when relatively small samples (i.e., 50 - 200) are used[57][58].



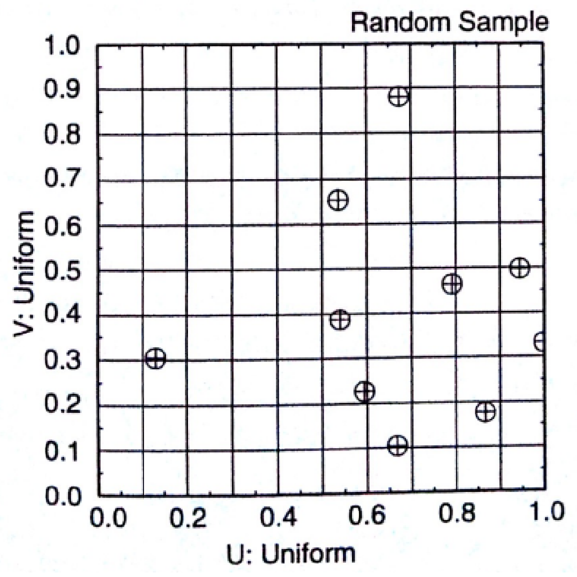


Figure C.1: Examples of Latin hypercube and random sampling to generate a sample of size 10 from variables U and V with U and V uniform on [0,1][72]

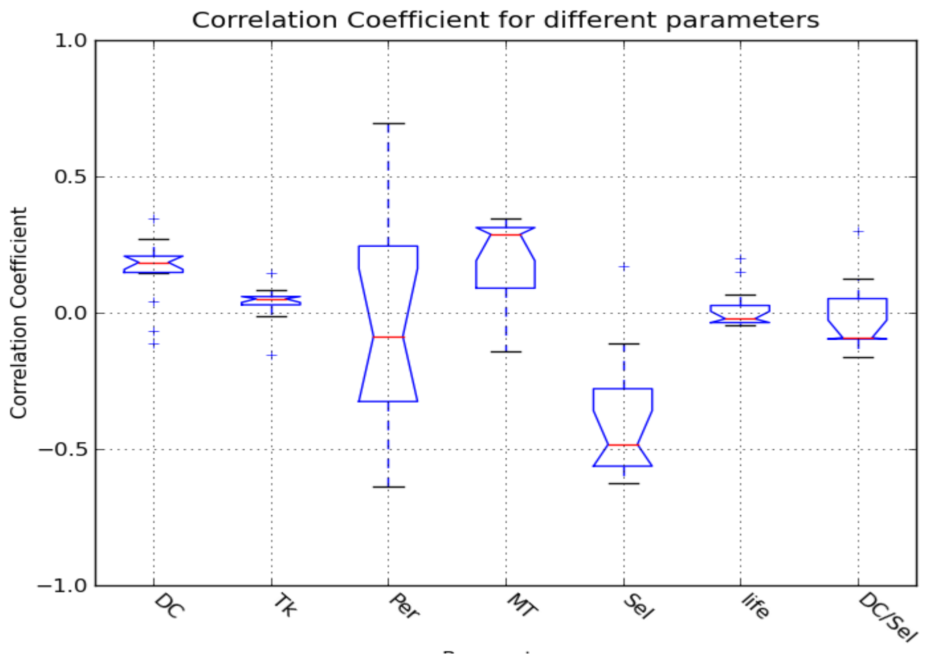


Figure C.2: Test 1-01 Correlation Coefficient

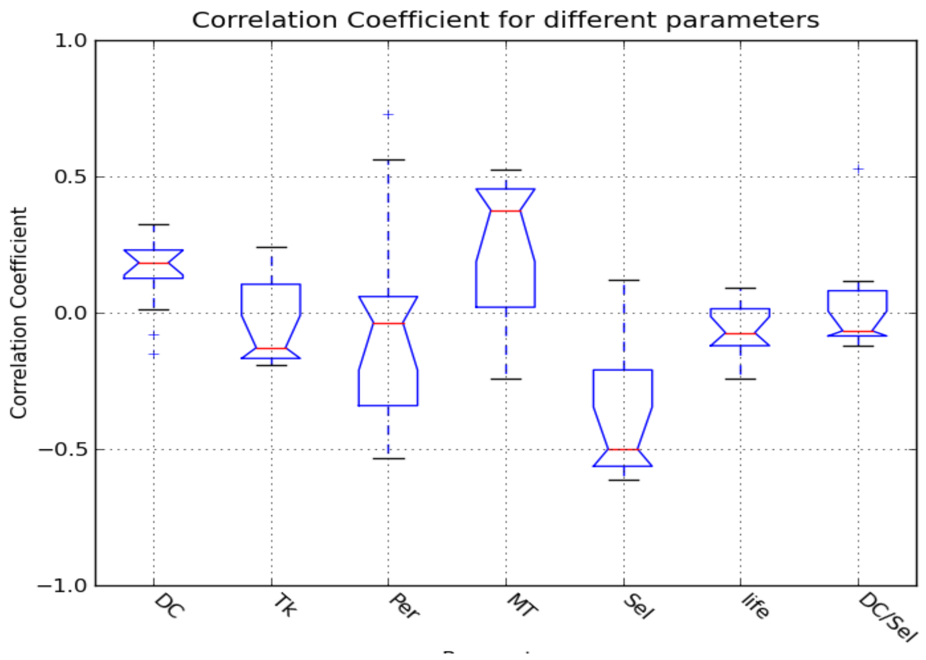


Figure C.3: Test 1-02 Correlation Coefficient

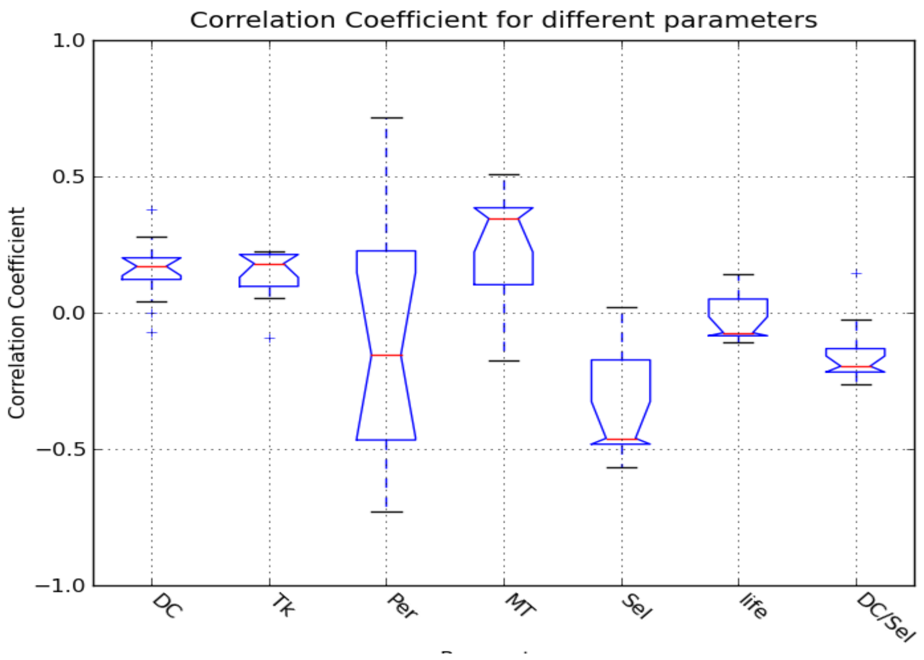


Figure C.4: Test 1-03 Correlation Coefficient

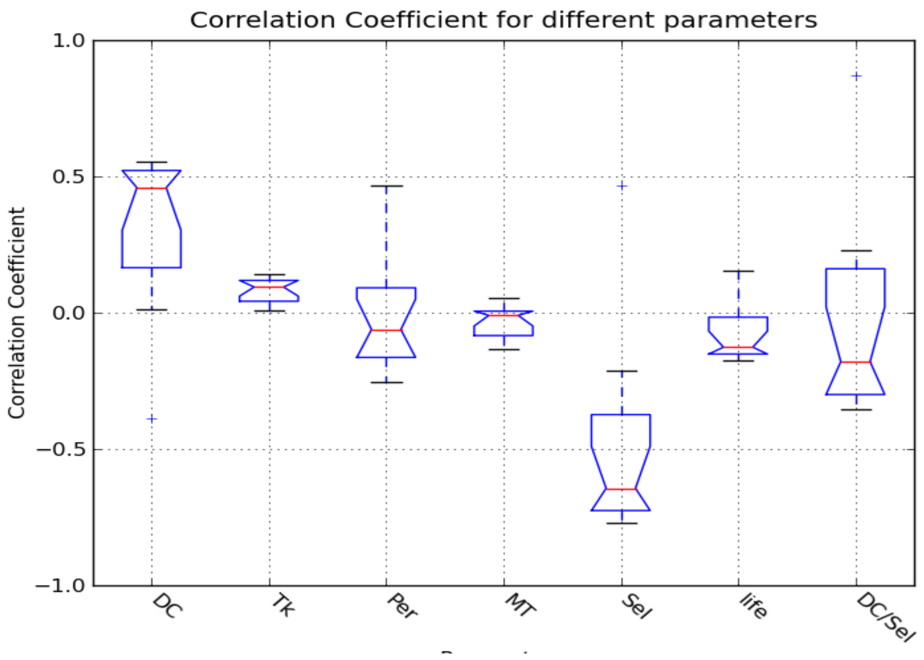


Figure C.5: Test 2-01 Correlation Coefficient

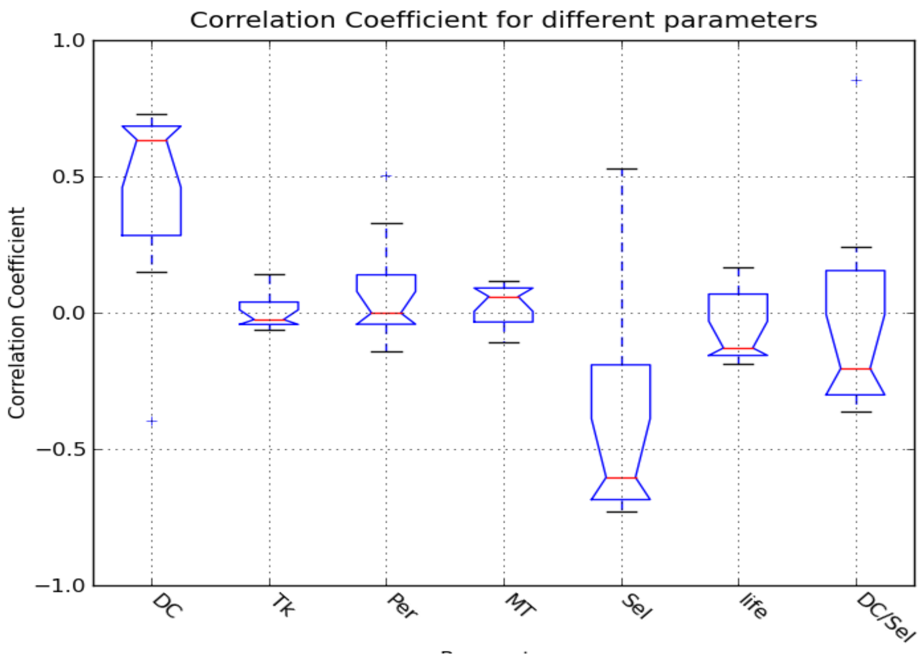


Figure C.6: Test 2-02 Correlation Coefficient

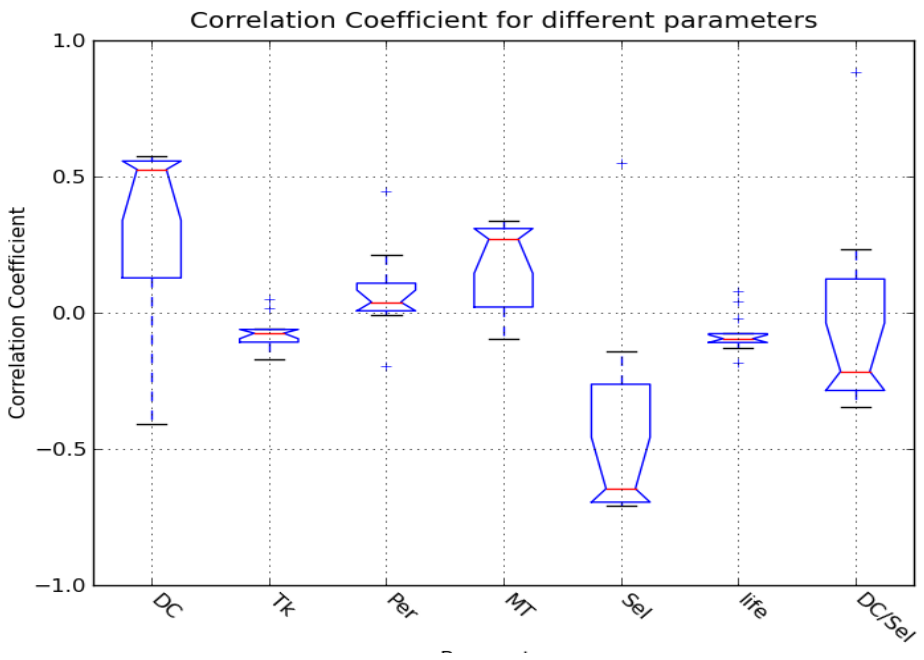


Figure C.7: Test 2-03 Correlation Coefficient

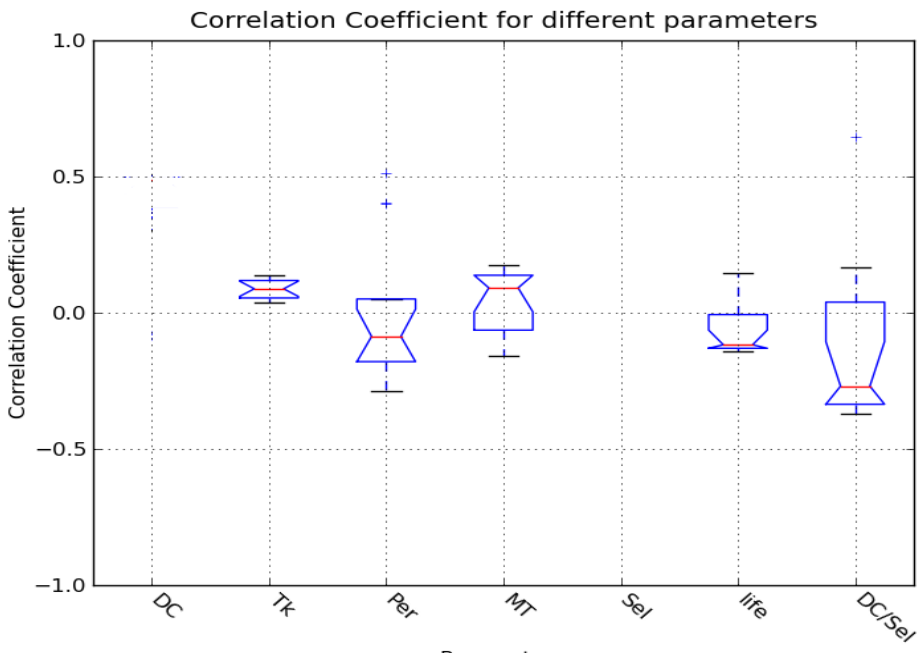


Figure C.8: Test 3-01 Correlation Coefficient

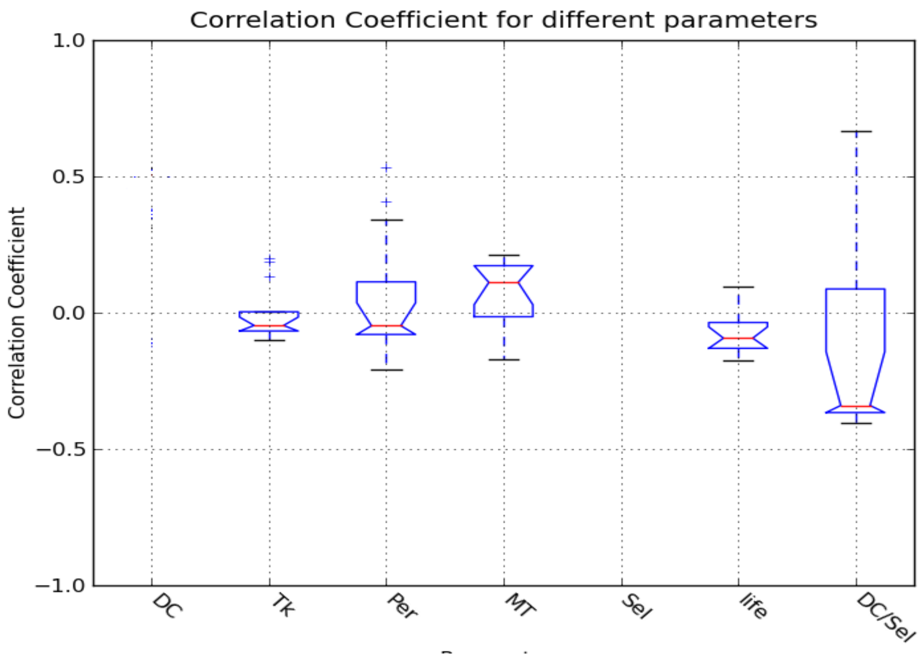


Figure C.9: Test 3-02 Correlation Coefficient

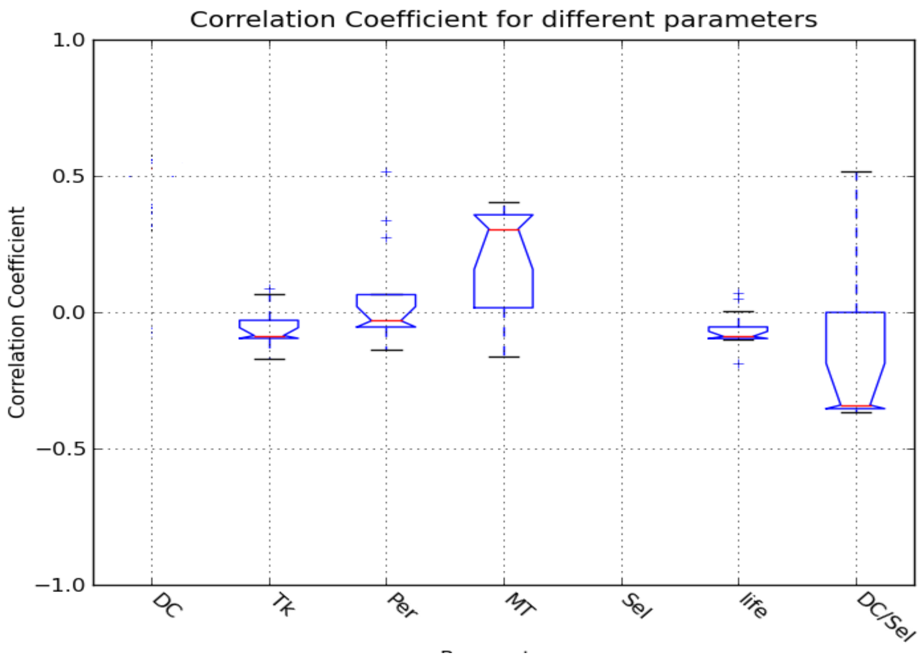


Figure C.10: Test 3-03 Correlation Coefficient

Appendix D

Wilcoxon test

The following description is cited from [20]. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ (i.e. it's a paired difference test).

It can be used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed or the data is on the ordinal scale.

The Wilcoxon function was used in this work is implemented using Python *scipy.stats.wilcoxon* function [21]. In order to reject a null hypothesis for N samples, the result has to be less than the value shown in Table D.1. Otherwise, the null hypothesis stands.

	Two Tailed significance levels:		
N	0.05	0.02	0.01
6	0	-	-
7	2	0	-
8	4	2	0
9	6	3	2
10	8	5	3
11	11	7	5
12	14	10	7
13	17	13	10
14	21	16	13
15	25	20	16
16	30	24	20
17	35	28	23
18	40	33	28
19	46	38	32
20	52	43	38
21	59	49	43
22	66	56	49
23	73	62	55
24	81	69	61
25	89	77	68

Table D.1: Table of critical values for the Wilcoxon test[9]

Appendix E

Wilcoxon signed-rank test results

Table E.1: Wilcoxon signed-rank test result individual parameter with performance test 2

H_0	data-1	data-2	data-3	data-4
DC	0.0	0.0	0.0	0.0
Tk	0.0	8.0	43.0	18.5
Per	0.0	135.5	29.0	68.5
MT	0.0	1.0	5.0	1.0
NS	0.0	0.0	9.0	1.0
Life	0.0	37.0	60.5	83.0

H_0	data-5	data-6	data-7	data-8
DC	0.0	0.0	0.0	0.0
Tk	35.0	43.0	2.0	70.0
Per	64.0	30.5	62.5	0.0
MT	3.0	5.0	0.0	17.0
NS	3.0	9.0	0.0	15.0
Life	117.0	60.0	129.5	1.0

Appendix F

Parameter evaluation results

	data-1	data-2	data-3	data-4	data-5	data-6	data-7	data-8
10	(0.001,0.0036)	(0.0307,0.0034)	(0.0215,0.0017)	(0.0115,0.0047)	(0.0147,0.0037)	(0.0352,0.003)	(0.0,0.0)	(0.028,0.0052)
50	(0.0049,0.0034)	(0.0048,0.0021)	(0.0078,0.0012)	(0.014,0.0051)	(0.0039,0.001)	(0.0038,0.0011)	(0.0,0.0001)	(0.0107,0.0008)
100	(0.005,0.0017)	(0.0051,0.0014)	(0.0068,0.0012)	(0.0136,0.0058)	(0.0028,0.001)	(0.0026,0.0008)	(0.0611,0.0003)	(0.0032,0.0003)
500	(0.004,0.0023)	(0.0051,0.0018)	(0.0034,0.0016)	(0.0159,0.0101)	(0.006,0.0013)	(0.001,0.0013)	(0.0668,0.0013)	(0.0011,0.0013)
1000	(0.0038,0.0025)	(0.0152,0.0053)	(0.0079,0.0017)	(0.007,0.0051)	(0.0213,0.0029)	(0.0221,0.0023)	(0.0188,0.003)	(0.0557,0.0214)

Table F.1: Parameters performance standard deviation verification

DC/NS	data-1	data-2	data-3	data-4	data-5	data-6	data-7	data-8
10/10	(0.0043,0.0025)	(0.015,0.0036)	(0.0215,0.0021)	(0.0079,0.0036)	(0.0202,0.0039)	(0.0305,0.0021)	(0.0125,0.0001)	(0.0493,0.0082)
50/50	(0.0046,0.0035)	(0.0053,0.0013)	(0.0064,0.0012)	(0.01,0.0064)	(0.0072,0.0011)	(0.0027,0.0011)	(0.0576,0.0002)	(0.0053,0.0009)
100/100	(0.0052,0.0027)	(0.0049,0.0011)	(0.0054,0.001)	(0.0131,0.008)	(0.0052,0.0011)	(0.0022,0.0008)	(0.0559,0.0002)	(0.0024,0.0004)
500/500	(0.0041,0.0011)	(0.0036,0.0013)	(0.004,0.0007)	(0.007,0.005)	(0.0035,0.0006)	(0.0017,0.0009)	(0.0765,0.0003)	(0.0011,0.0004)
1000/1000	(0.0041,0.0026)	(0.0028,0.0008)	(0.0032,0.0007)	(0.0034,0.0028)	(0.003,0.0008)	(0.0013,0.0008)	(0.0982,0.0003)	(0.0022,0.0006)

Table F.2: Parameters performance standard deviation verification 1:1

DC/NS	data-1	data-2	data-3	data-4	data-5	data-6	data-7	data-8
10/20	(0.0039,0.0026)	(0.0176,0.0029)	(0.0261,0.0017)	(0.0149,0.0041)	(0.0302,0.0039)	(0.0126,0.0017)	(0.0573,0.0002)	(0.0327,0.0093)
50/100	(0.0051,0.0026)	(0.0033,0.0012)	(0.007,0.0011)	(0.0179,0.0065)	(0.004,0.0013)	(0.0012,0.0008)	(0.0801,0.0003)	(0.0049,0.0009)
100/200	(0.0033,0.0024)	(0.0038,0.001)	(0.0045,0.0013)	(0.0117,0.007)	(0.0059,0.0008)	(0.0017,0.0008)	(0.064,0.0003)	(0.0016,0.0006)
500/1000	(0.0065,0.0023)	(0.005,0.0009)	(0.0043,0.0011)	(0.0078,0.0044)	(0.0034,0.001)	(0.0018,0.0007)	(0.1391,0.0007)	(0.0022,0.0005)
1000/2000	(0.0047,0.0012)	(0.0052,0.0007)	(0.0047,0.0007)	(0.0088,0.0051)	(0.0034,0.0008)	(0.0012,0.0007)	(0.1049,0.0004)	(0.0024,0.0007)

Table F.3: Parameters performance evaluation standard deviation verification 1:2

Appendix G

Support vector machine and its kernels

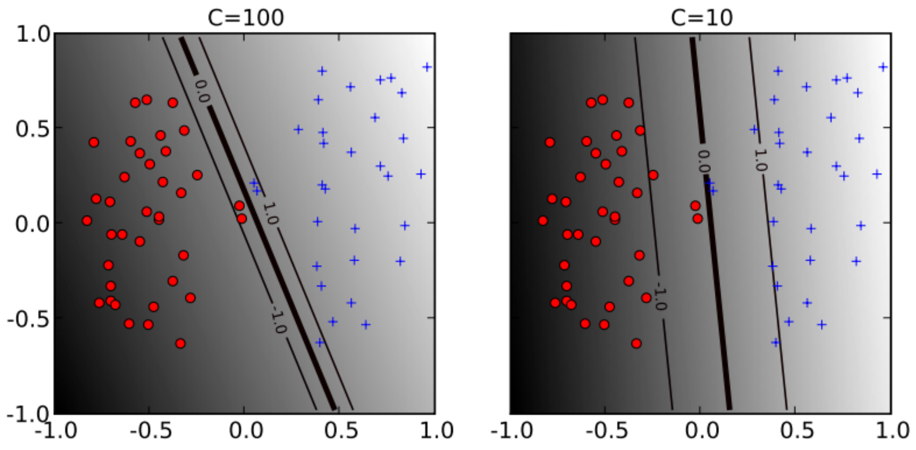


Figure G.1: The effect of the soft-margin constant, C, on the decision boundary. A smaller value of C (right) allows to ignore points close to the boundary, and increases the margin. The decision boundary between negative examples (red circles) and positive examples (blue crosses) is shown as a thick line. The lighter lines are on the margin (discriminant value equal to -1 or +1). The grayscale level represents the value of the discriminant function, dark for low values and a light shade for high values.[25]

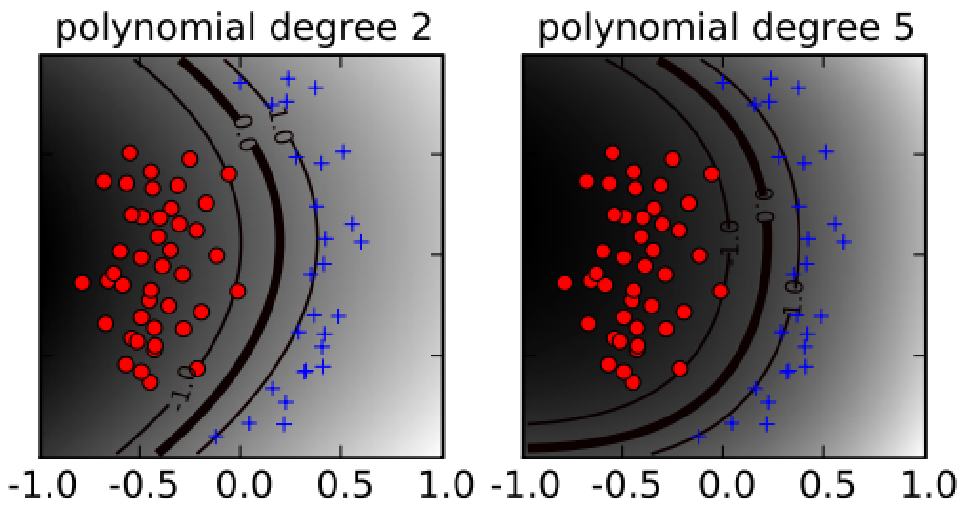


Figure G.2: The effect of the degree of a polynomial kernel. Higher degree polynomial kernels allow a more flexible decision boundary.[25]

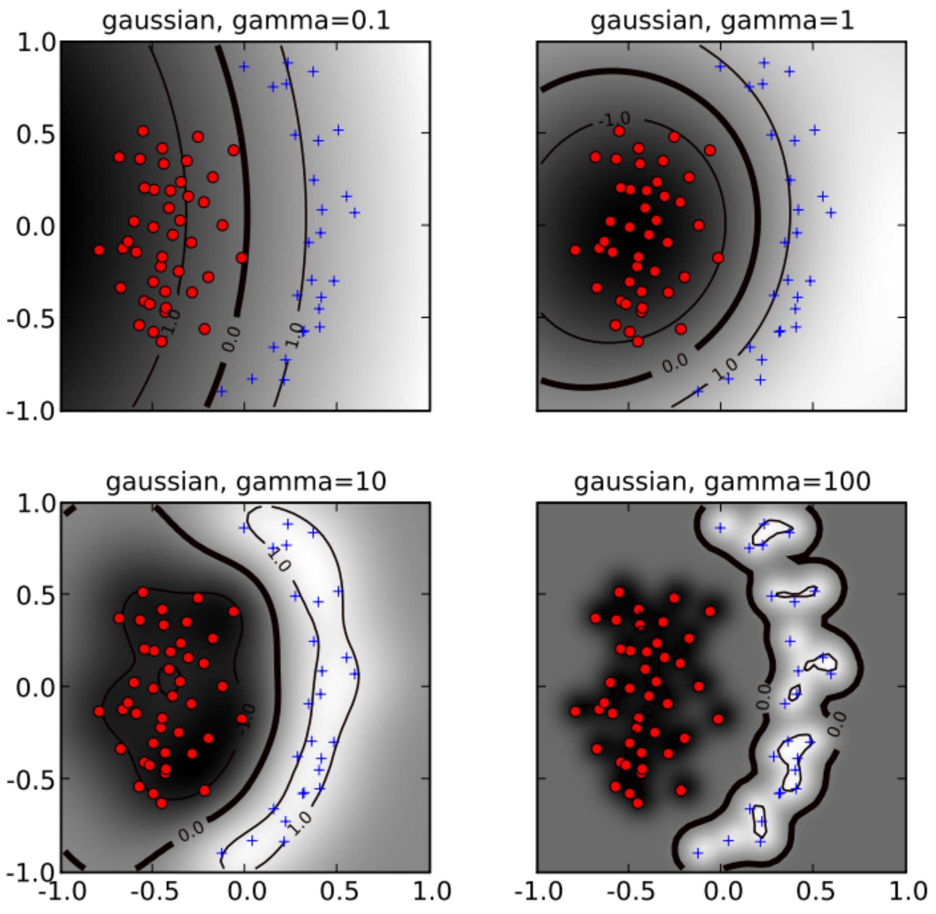


Figure G.3: The effect of the inverse-width parameter of the Gaussian kernel for a fixed value of the soft-margin constant. For small values of inverse-width (upper left) the decision boundary is nearly linear. As increases the flexibility of the decision boundary increases. Large values of inverse-width lead to over fitting (bottom).[25]

Appendix H

25 data sets used in comparison

faulty component	stuck at value	period (time steps)	duration (time steps)
IR sensor 0	1%	239	127
IR sensor 0	31%	123	3
IR sensor 0	63%	951	202
IR sensor 0	72%	596	399
IR sensor 2	28%	390	255
IR sensor 2	49%	402	75
IR sensor 2	5%	556	391
IR sensor 2	77%	995	820
IR sensor 4	23%	170	49
IR sensor 4	59%	66	37
IR sensor 4	94%	318	233
IR sensor 4	45%	878	481
IR sensor 6	35%	952	510
IR sensor 6	78%	754	375
IR sensor 6	91%	521	159
IR sensor 6	9%	288	179
IR sensor 7	46%	931	481
IR sensor 7	66%	681	515
IR sensor 7	71%	774	169
IR sensor 7	93%	229	153

faulty component	stuck at value	fault starts at (time steps)
IR sensor 0	4%	2598
IR sensor 2	51%	1718
IR sensor 4	48%	1130
IR sensor 6	77%	3438
IR sensor 7	2%	1519

Table H.1: 25 Test data sets used in comparison

Appendix I

K-nearest neighbour results

```
#This is result for KNN classify from ../KNN_data/.
#*********ftype-2-sen-6-val-81-per-186-dur-181-SVMdata
#Confusion Matrix:
#              Given labels:
#              IRsensor6  non-faulty
#  IRsensor6      3894        106
# non-faulty        0           0
#success rate: 0.973500
#balanced success rate: 0.500000
#area under ROC curve: 0.506490
#area under ROC 50 curve: 0.080000
0.9735
#*********ftype-2-sen-2-val-27-per-778-dur-742-SVMdata
#Confusion Matrix:
#              Given labels:
#              IRsensor2  non-faulty
#  IRsensor2      3819        181
# non-faulty        0           0
```

```
#success rate: 0.954750
#balanced success rate: 0.500000
#area under ROC curve: 0.804202
#area under ROC 50 curve: 0.868649
 0.95475
 #*********ftype-1-sen-2-val-51-at-1718-SVMdata
#Confusion Matrix:
#             Given labels:
#             IRsensor2  non-faulty
#  IRsensor2    2281        1617
# non-faulty     0          102
#success rate: 0.595750
#balanced success rate: 0.529668
#area under ROC curve: 0.808134
#area under ROC 50 curve: 0.930080
 0.59575
 #*********ftype-2-sen-0-val-63-per-75-dur-48-SVMdata
#Confusion Matrix:
#             Given labels:
#             IRsensor0  non-faulty
#  IRsensor0    1995        974
# non-faulty    553         458
#success rate: 0.616332
#balanced success rate: 0.551400
#area under ROC curve: 0.449801
#area under ROC 50 curve: 0.980000
 0.616331658291
 #*********ftype-2-sen-7-val-38-per-841-dur-515-SVMdata
```

```
#Confusion Matrix:
#              Given labels:
#              IRsensor7  non−faulty
#  IRsensor7     2053        825
# non−faulty     522         600
#success rate: 0.663250
#balanced success rate: 0.609167
#area under ROC curve: 0.802827
#area under ROC 50 curve: 0.964215
0.66325
#*********ftype−2−sen−4−val−45−per−878−dur−481−SVMdata
#Confusion Matrix:
#              Given labels:
#              IRsensor4  non−faulty
#  IRsensor4     410         227
# non−faulty     1995        1368
#success rate: 0.444500
#balanced success rate: 0.514079
#area under ROC curve: 0.643768
#area under ROC 50 curve: 0.804387
0.4445
#*********ftype−1−sen−6−val−34−at−3951−SVMdata
#Confusion Matrix:
#              Given labels:
#              IRsensor6  non−faulty
#  IRsensor6      0           0
# non−faulty      48         3952
#success rate: 0.988000
```

```
#balanced success rate: 0.500000
#area under ROC curve: 0.999789
#area under ROC 50 curve: 0.999789
 0.988
```

Glossary

Player Player[2] provides a network interface to a variety of robot and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. Player supports multiple concurrent client connections to devices, creating new possibilities for distributed and collaborative sensing and control.. 51

Stage Stage[2] simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry.. 50–52, 70, 90

Acronyms

ANN Artificial Neural Network. 4, 5, 9, 11–15, 47, 86, 88–90, 92–94, 97, 99–103, 105, 108, 109

D-DCA Diagnostic Dendritic Cell Algorithm. 22, 27, 28, 30, 31, 34–39, 44–47, 49, 51–53, 55, 60–64, 67, 70–73, 75, 76, 78, 80, 82–86, 88–90, 92–94, 96, 97, 101–103, 105, 107–111

DC Dendritic Cell. 22, 28, 30–35, 37–39, 42, 44–46, 49

DC Number of DC cells. 67–69, 71, 72, 75, 76, 78–86, 107, 108

FPR False Positive Rate. 55, 60–62, 69, 72, 78, 80, 93, 94, 96, 100, 102

Life Max life of cells. 67–69, 71, 72, 76, 78–81, 83, 85, 86, 108

MT Migration threshold. 67–69, 71, 72, 76, 78–83, 85, 86, 107

NS Number of random select cells. 67–69, 71, 72, 75, 76, 78–86, 108

Per Percentage threshold. 67–69, 71, 72, 76, 78–81, 83, 85, 86, 108

ROC Receiver Operating Characteristic. 50, 51, 54, 55, 60, 63, 89, 107

SVM Support Vector Machine. 4, 5, 9, 11, 16, 17, 47, 86, 88–90, 94, 97, 99–103, 105, 108, 109

the original DCA the original Dendritic Cell Algorithm. 5, 22, 25–28, 30, 31, 34, 36, 40, 67, 105

Tk Quantity T killer threshold. 67–69, 71, 72, 76, 78–81, 83, 85, 86, 108

TPR True Positive Rate. 55, 60–62, 69, 72, 78, 80, 93, 94, 96, 100, 102

Bibliography

[1] http://en.wikipedia.org/wiki/Extrapolation, jan 2010.

[2] http://playerstage.sourceforge.net/, jan 2010.

[3] http://www.nasa.gov/mission_pages/mars-pathfinder/, july 2011.

[4] http://en.wikipedia.org/wiki/Huygens_(spacecraft), july 2011.

[5] http://oceanexplorer.noaa.gov/technology/subs/abe/abe.html, july 2011.

[6] http://www.whoi.edu/page.do?pid=38095, july 2011.

[7] http://en.wikipedia.org/wiki/File:ROC_space-2.png, july 2011.

[8] http://en.wikipedia.org/wiki/Pearson_coefficient#Interpretation, jan 2012.

[9] http://www.sussex.ac.uk/Users/grahamh/RM1web/WilcoxonTable2005.pdf, jan 2012.

[10] http://en.wikipedia.org/wiki/Artificial_neural_network, jan 2012.

[11] http://en.wikipedia.org/wiki/Artificial_neuron, jan 2012.

[12] `http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm`, jan 2012.

[13] `http://en.wikipedia.org/wiki/Bleb_(cell_biology)`, jan 2012.

[14] `http://www.biooncology.com/research-education/apoptosis/pathways/intrinsic/`, jan 2012.

[15] `http://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test`, jan 2012.

[16] `http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)`, jan 2012.

[17] `http://docs.scipy.org/doc/numpy/reference/generated/numpy.corrcoef.html#numpy.corrcoef`, jan 2012.

[18] `http://en.wikipedia.org/wiki/Support_vector_machine`, jan 2012.

[19] `http://symbrion.org/`, jan 2012.

[20] `http://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test`, jan 2012.

[21] `http://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon`, jan 2012.

[22] J.A. Anderson and E. Rosenfeld. *Neurocomputing*, volume 1. The MIT Press, 1993.

[23] A. Avizienis. Toward systematic design of fault-tolerant systems. *Computer*, 30(4):51–58, 1997.

[24] A. Ben-Hur. Pyml: machine learning in python, 2010.

[25] A. Ben-Hur and J. Weston. A user's guide to support vector machines. *Methods in Molecular Biology*, 609:223–239, 2010.

[26] A. Bernieri, M. D'apuzzo, L. Sansone, and M. Savastano. A neural network approach for identification and fault diagnosis on dynamic systems. *Instrumentation and Measurement, IEEE Transactions on*, 43(6):867–873, 1994.

[27] H. Bersini. Immune network and adaptive control. *Varela and Bourgine*, 2332:217–226, 1994.

[28] R. Bi, J. Timmis, and A. Tyrrell. The diagnostic dendritic cell algorithm for robotic systems. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[29] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[30] A.P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[31] D.W. Bradley and A.M. Tyrrell. Immunotronics-novel finite-state-machine architectures with built-in self-test using self-nonself differentiation. *Evolutionary Computation, IEEE Transactions on*, 6(3):227–238, 2002.

[32] J. Carlson and R.R. Murphy. Reliability analysis of mobile robots. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 274–281. Ieee, 2003.

[33] N.V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[34] M. Chen, A.X. Zheng, J. Lloyd, M.I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 36–43. IEEE, 2004.

[35] L.H. Chiang, M.E. Kotanchek, and A.K. Kordon. Fault diagnosis based on fisher discriminant analysis and support vector machines. *Computers & chemical engineering*, 28(8):1389–1401, 2004.

[36] D. Coomans and DL Massart. Alternative k-nearest neighbour rules in supervised pattern recognition:: Part 1. k-nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 136:15–27, 1982.

[37] D. Dasgupta, Z. Ji, and F. Gonzalez. Artificial immune system (ais) research in the last five years. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 1, pages 123–130. IEEE, 2003.

[38] L.N. De Castro and J. Timmis. *Artificial immune systems: a new computational intelligence approach.* Springer Verlag, 2002.

[39] L.N. De Castro and F.J. Von Zuben. The clonal selection algorithm with engineering applications. In *Proceedings of GECCO*, volume 2000, pages 36–39, 2000.

[40] L.N. de Castro and F.J. Von Zuben. ainet: an artificial immune network for data analysis. *Data Mining: a heuristic approach*, 1:231–259, 2001.

[41] L.N. De Castro and F.J. Von Zuben. Learning and optimization using the clonal selection principle. *Evolutionary Computation, IEEE Transactions on*, 6(3):239–251, 2002.

[42] K.B. Duan and S. Keerthi. Which is the best multiclass svm method? an empirical study. *Multiple Classifier Systems*, pages 732–760, 2005.

[43] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[44] J. Fogarty, R.S. Baker, and S.E. Hudson. Case studies in the use of roc curve analysis for sensor-based estimates in human computer interaction. In *Proceedings of Graphics Interface 2005*, pages 129–136. Canadian Human-Computer Communications Society, 2005.

[45] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 202–212. IEEE, 1994.

[46] P.M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy:: A survey and some new results. *Automatica*, 26(3):459–474, 1990.

[47] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[48] J. Gertler. Analytical redundancy methods in fault detection and isolation. In *IFAC Symposium Safeprocess*, volume 1, pages 9–21, 1991.

[49] F.A. González and D. Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403, 2003.

[50] J. Greensmith. The dendritic cell algorithm. *University of Nottingham*, 2007.

[51] J. Greensmith and U. Aickelin. Dendritic cells for syn scan detection. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 49–56. ACM, 2007.

[52] J. Greensmith, U. Aickelin, and S. Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. *Artificial Immune Systems*, pages 153–167, 2005.

[53] J. Greensmith, U. Aickelin, and G. Tedesco. Information fusion for anomaly detection with the dendritic cell algorithm. *Information Fusion*, 11(1):21–34, 2010.

[54] E. Hart and J. Timmis. Application areas of ais: The past, the present and the future. *Applied Soft Computing Journal*, 8(1):191–201, 2008.

[55] S. Heath. *Embedded systems design*. Newnes, 2003.

[56] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[57] R.L. Iman and J.C. Helton. Comparison of uncertainty and sensitivity analysis techniques for computer models. Technical report, Sandia National Labs., Albuquerque, NM (USA), 1985.

[58] R.L. Iman and J.C. Helton. The repeatability of uncertainty and sensitivity analyses for complex probabilistic risk assessments. *Risk Analysis*, 11(4):591–606, 1991.

[59] R. Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance.* Springer, 2006.

[60] R. Isermann and P. Balle. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5):709–719, 1997.

[61] JFR Kerr, AH Wyllie, and AR Currie. Apoptosis (cell suicide). *Br. J. cancer*, 26:239–257, 1972.

[62] P.K. Lala. *Fault tolerant and fault testable hardware design.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1985.

[63] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 531–537. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.

[64] P. Matzinger. The danger model: a renewed sense of self. *Science*, 296(5566):301, 2002.

[65] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(4):115–133, 1943.

[66] M.D. McKay, R.J. Beckman, and WJ Conover. A comparison of three methods for selecting values of input variables in the analysis

of output from a computer code. *Technometrics*, pages 239–245, 1979.

[67] K. Murphy, P. Travers, and M. Walport. *Janeway's immunobiology*, volume 7. Garland Science, 2008.

[68] M. Neal. Meta-stable memory in an artificial immune network. *Artificial Immune Systems*, pages 168–180, 2003.

[69] R.J. Patton. Robustness in model-based fault diagnosis: the 1995 situation. *Annual reviews in control*, 21:103–123, 1997.

[70] D.E. Rumelhart, J.L. McClelland, and PDP Rese Diego) University of California (San. *Parallel distributed processing.* MIT Pr., 1988.

[71] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. *Swarm Robotics*, pages 10–20, 2005.

[72] A. Saltelli, K. Chan, E.M. Scott, et al. *Sensitivity analysis*, volume 134. Wiley New York, 2000.

[73] B. Samanta and KR Al-Balushi. Artificial neural network based fault diagnostics of rolling element bearings using time-domain features. *Mechanical Systems and Signal Processing*, 17(2):317–328, 2003.

[74] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 2010.

[75] A.J. Smola and B. Schölkopf. *Learning with kernels.* Citeseer, 1998.

[76] S. Stepney, R.E. Smith, J. Timmis, A.M. Tyrrell, M.J. Neal, and A.N.W. Hone. Conceptual frameworks for artificial immune systems. *International Journal of Unconventional Computing*, 1(3):315, 2005.

[77] T. Stibor, J. Timmis, and C. Eckert. A comparative study of real-valued negative selection to statistical anomaly detection techniques. *Artificial Immune Systems*, pages 262–275, 2005.

[78] W. Sun, J. Chen, and J. Li. Decision tree and pca-based fault diagnosis of rotating machinery. *Mechanical Systems and Signal Processing*, 21(3):1300–1317, 2007.

[79] J.A. Swets. *Signal detection theory and ROC analysis in psychology and diagnostics: Collected papers.* Lawrence Erlbaum Associates, Inc, 1996.

[80] B. Tang. Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, pages 1392–1397, 1993.

[81] D. Taylor and D. Corne. An investigation of the negative selection algorithm for fault detection in refrigeration systems. *Artificial Immune Systems*, pages 34–45, 2003.

[82] J. Timmis. Artificial immune systems: a novel data analysis technique inspired by the immune network theory. 2000.

[83] J. Timmis. Artificial immune systemstoday and tomorrow. *Natural Computing*, 6(1):1–18, 2007.

[84] J. Timmis, P. Andrews, and E. Hart. On artificial immune systems and swarm intelligence. *Swarm Intelligence*, pages 1–27, 2010.

[85] J. Timmis, A. Hone, T. Stibor, and E. Clark. Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403(1):11–32, 2008.

[86] J. Timmis and A. Tyrrell. On homeostasis in collective robotic systems. *Artificial Immune Systems*, pages 307–309, 2010.

[87] J. Twycross and U. Aickelin. Libtissue-implementing innate immunity. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 499–506. IEEE, 2006.

[88] A.T. Vemuri and M.M. Polycarpou. Neural-network-based robust fault diagnosis in robotic systems. *Neural Networks, IEEE Transactions on*, 8(6):1410–1420, 1997.

[89] V. Venkatasubramanian, R. Rengaswamy, and S.N. Kavuri. A review of process fault detection and diagnosis:: Part ii: Qualitative models and search strategies. *Computers & Chemical Engineering*, 27(3):313–326, 2003.

[90] V. Venkatasubramanian, R. Rengaswamy, S.N. Kavuri, and K. Yin. A review of process fault detection and diagnosis:: Part iii: Process history based methods. *Computers & Chemical Engineering*, 27(3):327–346, 2003.

[91] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S.N. Kavuri. A review of process fault detection and diagnosis:: Part i: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293–311, 2003.

[92] Z. Wang, Y. Liu, and P.J. Griffin. A combined ann and expert system tool for transformer fault diagnosis. *Power Delivery, IEEE Transactions on*, 13(4):1224–1229, 1998.

[93] A. Watkins, J. Timmis, and L. Boggess. Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

[94] M. Witczak. Advances in model-based fault diagnosis with evolutionary algorithms and neural networks. *International Journal of Applied Mathematics and Computer Science*, 16(1):85, 2006.

[95] Y. Yang, D. Yu, and J. Cheng. A fault diagnosis approach for roller bearing based on imf envelope spectrum and svm. *Measurement*, 40(9-10):943–950, 2007.

[96] S.J. Young. *Real time languages: design and development*, volume 21. Ellis Horwood, 1982.

[97] S.F. Yuan and F.L. Chu. Support vector machines-based fault diagnosis for turbo-pump rotor. *Mechanical Systems and Signal Processing*, 20(4):939–952, 2006.

[98] Y. Zhang, X. Ding, Y. Liu, and PJ Griffin. An artificial neural network approach to transformer fault diagnosis. *Power Delivery, IEEE Transactions on*, 11(4):1836–1841, 1996.