An Evaluation of Fault-Tolerant Robotic Controller Using Evolutionary Computation

Philip Wong

A thesis submitted to Auckland University of Technology in fulfilment for the degree of Master of Engineering

2019

Abstract

The aim of the research reported in this thesis was to design and construct several Fault-Tolerant Controllers for a simulated multi-joint planar robotic arm system. Analysis and comparisons of the convergence rates of the controllers developed will assist in determining and understanding the effectiveness of the controller to adapt itself as the number of faults in the system increases.

This research designed four Fault-Tolerant Controllers by combing two well-known Optimisation Algorithms: Genetic Algorithm (GA) and Particle-Swarm Optimisation (PSO) with two well understood robotic controllers: Artificial Neural Network (ANN) and Lookup Table (LUT). The effectiveness of the controller is measured by how fast it can recover when different numbers of faults are applied to it. A Fault-Tolerant Controller can be constructed using either active or passive Fault-Tolerant Control. Passive Fault-Tolerant Control is typically achieved via redundancy, such as having backup components integrated into the system. When faults occur, the system can remain operational by quickly switching to the backup component. Active Fault-Tolerant Control actively updates its parameter and/or architecture to adapt itself to compensate the effects of faults.

Results have shown that the Fault-Tolerant Controller PSO-LUT has the fastest convergence rate over all faults combinations applied to it, followed by PSO-ANN, GA-LUT and GA-ANN. Results from all the controllers have shown that the overall performance of the controllers is acceptable. However, in some instances the controllers can become stuck near a local optimal for a significant period before converging to a solution.

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning, except where due acknowledgement is made in the acknowledgements.

Philip Wong

February 2019

Acknowledgements

I would like to express my gratitude to my supervisors, Dr Mark Beckerleg for his support, guidance and advice throughout this research. I also greatly appreciate and express many thanks to the assistance given by Dr Jeff Kilby for his countless hours in revising my thesis and assisting me in completing the research.

Sincere thanks are also extended to all staff of the School of Engineering, in particular the technicians Justin Matulich and the administration staff for their kindness and encouragement during my studying period at the Auckland University of Technology.

Finally, I would like to personally thank my family for their understanding, support and encouragement on my decision to continue to study in postgraduate level.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
D-H	Denavit-Hartbenberg
GA	Genetic Algorithm
LUT	Lookup Table
PSO	Particle-Swarm Optimisation

Chapter 1
Introduction

1.1 Background

From the dawn of human civilization, it was always one of mankind's greatest ambitions to create an automaton artefact and/or sentient being from our own image [1; Chapter 1]. From Talus the bronze giant in ancient Greek mythology, to the Clepsydra water clock the first automated mechanical artefact described in ancient Babylonians texts [1; Chapter 1]. As human ingenuity and creativity continued to flourish, systems such as the automaton theatre of Heron of Alexandria, and Leonardo da Vinci's multitudinous brilliant devices have been designed and constructed [1; Chapter 1]. In addition, such enthusiasm was widespread across the world, with creations such as Jacquet-Droz's family of android and the Karakuri-Ninguo mechanical dolls [1; Chapter 1]. Since the Industrial Revolution, automation has transformed many industries, especially in manufacturing and agriculture [1; Chapter 1, 2]. It was that period which gave birth to the conceptualisation and realisation of the necessity to advance in machinery and automation. Nevertheless, the emergence of modern robotics did not surface until the twentieth century, when the underlying technologies such as the transistors and the integrated circuit (IC) were invented.

The term robot was derived from the Czech and the Slovak word *robota* meaning forced labour, which were conceived by the Czech writer Karel Čapek in his play *Rossum's Universal Robots* (R.U.R) in 1920 [1; Chapter 1]. Ever since the term came into being, it was quickly popularised and accepted by the general public to the extent that it became more commonly used than automaton. Since its popularisation, the study of robots has attracted a tremendous amount of research and contribution from different fields and professions, for example the famous *'Three Laws of Robotics'* envisioned by Russian science-fiction author Isaac Asimov in his work [1; Chapter 1], which explores ethics of the interaction between human and robots. With Julius Lilienfelds [3] invention of transistor in the 1920's which give rise to modern electronics, a new area of research in the field of Artificial Intelligence (AI) which explores the connection between human intelligence and the machine had emerged [1; Chapter 1]. The emergence of the field of AI benefited from the dawn of the information age, which saw an advancement in different related fields of studies such as machine-learning

algorithms, mechanics, control systems, as well as the invention of modern computers and continuous advancement in electronics[1; Chapter 1]. This gave birth to early design and development of robots and the research field of robotics was introduced. With advancements in all the related fields of research which drive researchers into new areas of research and discovery, this in turn prompted novel concepts and solutions. For example in 2016 Google DeepMind utilised a deep neural network to encode AlphaGo to defeated the world's number 1 player in Go, an ancient chess like game originated from China [4]. That feat outshines all earlier processor such as IBM's Deep Blue and Watson, because in a game of Go on a 19×19 Go chess board it has 2.08×10^{170} legal positions, making it impossible to use older machine-learning algorithms, for example the brute-force calculation used in Deep Blue to make enough predictions ahead of each opponent's move to secure a win. Thus, since its inception the field of AI has achieved some unparalleled accomplishments, but failed to formalise the problem space it is concerned with, namely solving the strong AI problem as noted by Yampolskiy [5]. During the second half of the twentieth century, with the development of essential components such as computers, electronics and Internet of Things, enabled more advanced robots and robotic controllers to be designed and programmed. With the advancement in robotics and its related area, robots have become increasingly more intelligent and able to perform more complex tasks, which leads to robotics being integrated into a wider range of applications. For example, industrial robots in a factory and the new type of intelligent robotic described by Taylor et al. [6] that assist surgeons to perform certain tasks, were proven to be more efficient and have less chances of making errors compared to its human counterparts when performing the same repetitive and tedious tasks.

From the 1980s a new paradigm in robotic research had surfaced, which studied the intelligent connection between how robots dynamically perceive and interact with the real world [1; Chapter 1]. Therefore, robots were equipped with sensors to observe and extract information by itself relative to its surrounding environment. In addition, robots are equipped with locomotion equipment to enhance its manoeuvrability in its environment and/or manipulation apparatus to interact with objects present in the environment [1; Chapter 1]. By the 1990s came the realisation of the need to design robots with a higher degree of autonomy, to replace humans in performing tasks in hazardous environment, to enhance its human operator's performance and reduce fatigue, or to develop products aimed at improving quality of life [1; Chapter 1]. With

the ambition to increase human-to-machine and machine-to-environment interaction, at the end of the second millennium robotics underwent a major transformation. Robotics shifted away from a dominantly industrial focus, and rapidly expanded into human-centred and life-like robotics which provide unlimited benefits and limitless possibilities to humans, while they are expected to safely and dependably integrate into and co-habitat with human society [1; Chapter 1, 2]. For example, in search and rescue, disaster response, health care, transportation, entertainment, education, manufacturing and assistance. In addition, modern robotics proved to be an invaluable tool to venture into and explore areas which are deemed inaccessible and/or hazardous for human expeditions, from nuclear exclusion zones to distant planets and deep oceans.

In today's world, there is an increasing need for and availability of personal service robots being integrated into human life and society; for example Honda's ASIMO[1] [7]. In addition, with the advancements in software and machine learning algorithms that are able to assist in scientific exploration, for example fast machine learning algorithm are able to self-learn and recreate the Nobel prize winning experiment by Wigley et al. [8],. The classical definition of autonomous systems which had defined generations of systems that simply apply pre-programed reactions in response to the system's inputs is becoming obsolete. By contrast, true autonomous systems that seek to carry out goal-oriented tasks whose implementation details are not predefined, either by necessity or as a design strategy developed by Lussier et al. [9] are emerging to the centre stage. Yet a major setback to their wider adoption into more complex environments and dynamic situations outside the factory and laboratory environment is their fragility against faults [2]. Most contemporary robots and robotic controllers lack or have limited ability to compensate for or self-generate a novel solution for situations that are not anticipated during the design phase. In addition, it is impractical to anticipate every possible situation that the robot may encounter during its deployment and an engineer or technician is not always available to provide the support the system needed to continue with its task [2], especially when there are an increasing number of robots deployed in harsher and more complex environments [10]. A robot may be rendered useless or become a burden, if it is unable to adapt to unanticipated situations or when its objectives are updated but service and repair are not immediately available, for example

[1] **ASIMO** (**A**dvanced **S**tep in **I**nnovative **Mo**bility)

space-exploration or surveying the nuclear exclusion zones of the exploded Fukushima nuclear plant. Therefore, this research is set to investigate and compare optimisation algorithms controllers to generate a novel solution for a robotic controller that enhances its adaptability in unknown environment, and its fault tolerability.

1.2 Research Objectives

The research question is to investigate and compare the effectiveness of two Optimisation Algorithms, which are: Genetic Algorithm and Particle Swarm Optimisation, in combination with two robotic controllers to create an active Fault Tolerant Control (FTC) to control a simulated robotic arm system. The eight joint robotic arm used in this research is tasked to move from a set starting point to a location set by the user within its workspace, with or without occurrence of fault in the system. For this research, a fault is defined as when one or more of its joints is no longer able to rotate and is permanently set to 0°. The effectiveness of the optimisation algorithm and the controller itself are assessed and compared with the following aspects:

(i) The performance of the controllers.
(ii) The effects on the performance of the controller when the parameters of the optimisation algorithm are changed.
(iii) The effects on the controller when a fault(s) is introduced at the start or during the time a solution is being generated.

The Optimisation Algorithms investigated in this research are Genetic Algorithm (GA) and Particle-Swam Optimisation (PSO). The controllers to be optimised are Artificial Neural Network (ANN) and Lookup Table (LUT). GA, PSO and ANN are algorithms inspired from observed events in nature, of which they belong to a larger group known as Evolutionary Algorithm.

Evolutionary Algorithms are metaheuristic[2] algorithms that have been widely researched and popularly used for autonomous robots due to their adaptability in unknown environments and fault tolerance. In robotic controller design and

[2] Definition of Metaheuristic: Solution methods that orchestrate an interaction between local improvement procedures and higher-level strategies to create a process capable of escaping from local optima and performing a robust search of solution space.

implementation, a common and extensively studied practice is to implement an Evolutionary Algorithm with an ANN, where the weights and/or the structure of the network are evolved by the Optimisation Algorithm. Another practice that has gained popularity in recent years is to update the weights and/or structure of an ANN, using posterior knowledge calculated by Bayesian Inference.

An ANN is a relatively well-established concept, its origin can be traced back to shortly after the Second World War [11]. Since then it has been extensively studied and it is highly popular in fields such robotics, machine learning, and problems that involves classification, regression and prediction of the data. An ANN can give relatively accurate approximated output from its input data. The ANN design used in this research is a standard feed-forward two-layer Neural Network. Other less-studied evolvable robotic controllers are LUT where the parameters of the table are evolved using a GA or PSO.

LUT are commonly used in low-powered embedded systems, because it utilises an index-based system which reduces computational runtime and generally puts less computational load on the Microcontroller unit. There are various ways to configure a LUT, depending on the complexity and the nature of the computational task that it is trying to solve, but when used as a robotic controller it is generally restricted by the quantisation of the inputs, outputs and the nature of the problem it is trying to compute. However, once its parameters are properly set up a LUT controller generally requires far fewer computational resources from the Microcontroller Unit than other controller system examined in this thesis.

The following aspects of the Evolutionary Computation are evaluated and compared:

(i) Overall efficiency of each type of Evolutionary Computation, both numerically and visually.
(ii) Efficiency of the Evolutionary Computation had on the controller are determined by the number of iterations required to converge to a solution with the desired fitness level.
(iii) Investigate the effects of varying parameters within the Evolutionary Computation has on the convergence rate to a desired fitness level.
(iv) Effects on the Evolutionary Computation convergence rate when certain physical properties of the robotic arm system are changed.

1.3 Thesis Structure

The following chapters are structured in the following manner. Chapter 2 provides a detailed literature review in the theory of fault-tolerant control and fault-tolerant controller. Chapter 3 provides a comprehensive description of the system developed for this research to test the developed fault-tolerant controllers. Chapter 4 contains the design, architecture and description of the fault-tolerant controller developed in this research. Chapter 5 presents the results generated by the developed fault-tolerant controllers, and a comprehensive analysis of the results. Finally, chapter 6 gives the conclusion and recommendation of future work related to the research presented in this thesis.

In summary, this chapter introduces the research question, which is to compare the effectiveness of two Optimisation Algorithms for active Fault Tolerant Control (FTC) as well as the overview and layout of this thesis. The next chapter provides an in-depth discussion literature review on existing work and system developed for this research.

Chapter 2
Literature Review

Humanity has become more dependent on the availability and stability of complex industrial processes, that are required to meet the ever-increasing demands of high production and product quality. As well as improving efficiency in economic and ecological operation, there is a need to design systems with increased complexity and automation integrated into it [12-15]. Modern industrial process systems consist of many different components; such as human operators, robots, material distribution, transportation, power distribution, tools and many more. So, to ensure efficient and high-quality operation of the system, each component is required to correctly perform its assigned tasks. As the level of complexity increases along with the number of components, subsystems and their interactions, the probability for faults to occur within the system increases. When a fault occurs, it may lead to wide-spread consequences and affect the performance of the process or system as a whole [12]. Additionally, for safety-critical systems such as nuclear power plants, safety, availability, dependability and reliability concerns are of utmost importance [16-18]. If left unattended, any unresolved faults and failures in the system may result in catastrophic consequences. In recent decades definitions and classifications of faults have been extensively studied, and so a fault can be classified into two forms [17, 18]:

1. The first is a revertible malfunction in the system's structure or parameters that cause the system to behave in a manner which leads to degraded system performance or the loss of functionality.
2. The second is where the system has a permanent loss in functionality.

Many researchers [12, 14, 16, 17, 19, 20] have classified the cause of faults into five main categories, which are:

1. *Internal faults:* faults from within the system itself, which may result in one of many consequences, for example power loss, a break in communication or a physical breakage (for example a malfunction in the actuator).
2. *Sensory faults:* faults caused by faulty sensors and incorrect sensory signals, which may cause a false alarm to occur when the system is operating normally, or a false negative where there is an actual fault occurring in the system which was undetected by the sensors.

3. *Operational fault:* faults caused by the controller performing an incorrect control action. This may be caused by a sudden change in the system's environment that leads to the system operating outside its intended regions.
4. *Design fault:* faults that were undetected during the design phase of system but reduce the performance of the system when the system is deployed.
5. *Combinational faults:* a combination of two or more of the faults described above. These may be one of the hardest types of faults to diagnose.

Further, faults can be broken down into the following classifications [18]:

(i) Abrupt faults, are malfunctions that occur suddenly.
(ii) Incipient faults, occur at a more gradual rate.
(iii) Permanent faults, which cause permanent failure to the system.
(iv) Transient faults, causing temporary malfunctions that disappear after a period.
(v) Intermittent faults, are repeated occurrences of transient faults.
(vi) Hidden faults, are faults that occur when a certain component or subsystem is activated, otherwise they are undetectable.

Faults are bound to occur for any physical system because it is practically impossible to build a perfect system, and there are countless conditions that can trigger a fault to occur. In addition to this, faults may also occur due to impurities and fatigue of materials used to construct the system, physical wear and tear of components and damage [20]. It is impossible for an engineer to anticipate every possible fault and take preventative steps when designing the system. In the event of a fault occurring, a system with a conventional feedback control design may result in deterioration in performance, which may lead to instability in the system, loss in functionality and ultimately may lead to a total system failure, where maintenance or repair are not immediately available[12, 15].

A fault-tolerant controller is a controller that under nominal operations, acts like any other controller available, but its true features only appear when a fault or faults occurs in the system being used. Controllers are usually designed for the system to operate in a faultless situation, such that the closed loop meets the assigned performance specifications [12].

2.1 Fault-Tolerant Controller

A well-designed fault-tolerant controller should be able to modify itself to adapt to uncharacteristic behaviours in the system due to a fault(s). The architecture of a fault-tolerant controller, as shown in Figure 2.1, typically consists of two parts; (i) Fault Diagnosis (FD) and (ii) Fault-Tolerant Control (FTC), and it can be either passive or active [18].

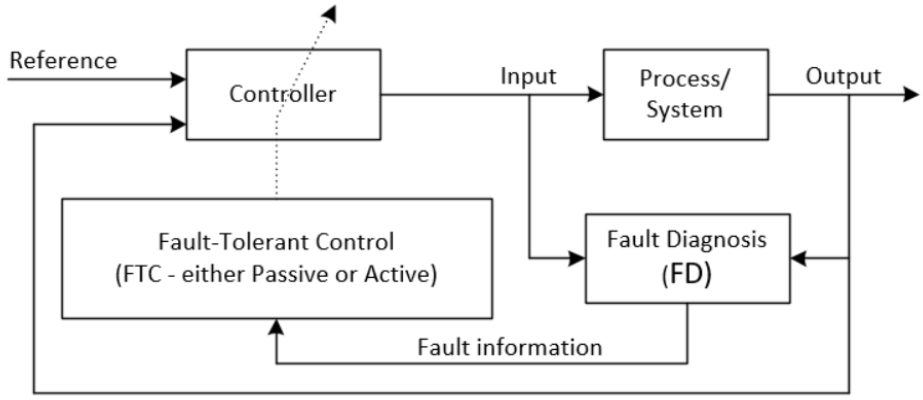


Figure 2.1. Architecture of Fault-Tolerant Controller

The function of FD is to provide critical information about the fault, i.e. detects, analyses and isolation the faults and passes this information to the FTC to the controller [16, 17, 21]. The FTC enables the system to continue performing its intended functions in the presence of faults at a relatively normal operation or to fail gracefully rather than an abrupt system failure [16, 17, 21]. FTC deals with the interaction between the system and its controller, by either: (a) passively compensating or (b) actively redesigning the controller to adapt to the fault when it occurs.

FD and FTC are not governed by the conventional feedback mechanism of the controller [12], but rather by a supervision system that governs the control structure, algorithm and parameters of the feedback controller.

2.1.1 Fault Diagnosis

Sensor and actuator failure, equipment fouling, feedback variation, product changes and seasonal influences make up to 60% of industrial controller problems [15]. When faults occur, they may significantly cause the system's operation to deviate outside its nominal operating range and may lead to failure. Therefore, diagnosis provides the opportunity for early detection, isolation and analysis of faults, enabling the system to adapt and/or

correct the faults before substantial damage is dealt to the system, or system failure occurs [12, 18, 20, 21]. In addition, a good fault diagnosis system should be robust against noise, incorrect signals and their propagation through the system, and should have a high sensitivity and ability to detect faults [18]. Techniques used in fault diagnosis can be generalized into the following:

(a) *Hardware redundancy diagnosis:* can be achieved by reconstruction of the system components using multiple redundant hardware components [13, 16]. Faults become detectable by continuously comparing the output of the components against one of its identical redundant components. Hardware redundancy diagnosis can simultaneously achieve fault detection and isolation, which is found to be highly reliable, but it is expensive to implement, and therefore its application is usually restricted to several safety-critical components.

(b) *Analytical model-based diagnosis:* utilizes mathematical models to achieve fault detection and isolation in the system [13, 22]. Mathematical modelling is used to represent the dynamics and features of the system. Fault diagnostic are performed by first generating the residual signal, which calculates the difference between the measured output of the system and the estimated output calculated from the model. The residual signal is then evaluated and analysed to determine the fault or faults. With the distinctive analytical process and the way that the system is represented in terms of mathematical models, many researchers believe that this type of fault diagnosis is the most effective in fault detection for dynamic processes.

(c) *Signal processing-based diagnosis:* detects faults based on the assumption that certain signals contain information about the faults. The form of symptoms and associated fault diagnosis can be attained by analysing such signals [13]. These symptoms are typically found in the form of time-domain functions such as magnitudes or frequency-domain functions such as frequency spectral lines. This approach is predominately used for a steady state system, as it has a limited efficiency in fault detection for dynamic systems.

(d) *Statistical data-based diagnosis:* uses the availability of data collected using historical data from previous operation cycles and online measurement data of the system to diagnose the fault. In order for a statistical-data based

diagnostic to work properly it needs to follow the procedures suggested by Ding [13]. First the system needs to be trained using historical data sets that are presented as independent knowledge of the system under monitoring and transformed to a diagnostic system or algorithm. Then the trained diagnostic system is set online, in which live inputs are processed by the trained system for reliable fault detection and identification.

(e) *Knowledge-based fault diagnosis:* utilises a qualitative model which represents the prior knowledge of the system's functionality and output obtained when monitored [13]. Faults are then detected by running a well-developed search algorithm through the model. Examples of the knowledge-based fault diagnosis are the Bayesian Inference engine and the Expert System, both of which draw their conclusions based on combining their inbuilt knowledge base, data base, inference engine and explanation component.

Overall the role of fault diagnosis is to monitor the behaviour of the system and provide all essential information when component faults are observed [17]. Theoretically, fault diagnosis can be subdivided into three subtasks which are: (a) Fault detection, (b) Fault identification, and (c) Fault Isolation.

2.1.2 Fault-Tolerant Control

FTC is a method that enables the system to adapt and adjust its structure, parameters and behaviour to accommodate the faults [12, 21]. By doing so the system is said to become fault-tolerant, which enables the system to maintain a certain degree of nominal operation or fail gracefully when faults occur, rather than abrupt system failure when maintenance and repair is not immediately available [20]. Typically, controllers are designed to meet assigned tasks under faultless situations, whereas a fault-tolerant controller handles the assigned task even though the system is subject to faults allowing the overall system to satisfy its goal [15]. FTC enables the controller to react to the existence of the fault by reconfiguring its process and behaviour to the faulty behaviour of the system, such that the system appears to the external observer as operating normally and is able to continue with its tasks, despite the presence of the fault [12]. Furthermore Blanke et al., Noura et al. and Zolghadri et al. [12, 15, 18] point out that there are two types fault-tolerant control:

1. *Passive fault-tolerant control* uses robustness and redundancy of the system design, to negate the effects of the fault on the system. Passive FTC is typically achieved by increasing the system's redundancy [12, 16, 21, 23], for example by having two sets of components so that if one set becomes faulty, the FTC can quickly switch to the backup set while shutting down the faulty set to be repaired and prevent further damage to the system. However passive FTC is not as popular and widely implemented as active FTC due to the significant increase in cost for having abundant sets of spare components. They are most widely adopted in critical-safe systems such as nuclear power plant, chemical plants and airplanes [16, 18].
2. *Active fault-tolerant control* uses a controller that actively attempts to adapt and adjust its operation to compensate for the fault identified by the supervision system, by changing the structure and parameter of the control loop. The architecture of an active fault- tolerant control can be generalized into two blocks: diagnostic and adaption. The diagnostic block measures the input and output signals and compares the difference with the system model, resulting in a characterization of the fault which is passed to the adaption block. The adaption block utilises this information to adjust the controller and system's behaviour to compensate for the fault. The redesigned block is considered successful if it can generate a solution for the controller to compensate for the faulty situation and the fault is said to be recoverable, otherwise it is considered to be non-recoverable. Recoverable faults allows the controller to revert the system operation back to a nominal or degraded performance. For non-recoverable faults, the controller needs to make a supervision-level decision, adjusting the system objective from the current objectives which the system can no longer perform, to a safer objective (for example a safe shutdown), to prevent further deterioration that may cause imminent danger to the system and its surroundings.

In recent years there has been an increase in the number of fault-tolerant automation systems built with inbuilt *Fault Diagnosis* and *Fault-Tolerant Control.* When the system has both components it is commonly refer to as fault-tolerant system [12, 13, 15-18, 21, 23]. This research did not create a Fault Diagnostic system, however it will be required in a real life situation to update the simulation. Active FTC actively redesigns the controller to adapt to the fault [17, 21]. In recent years active FTC has gained significant popularity and a significant amount of effort has been invested into its development. Thus, this research will only focus on developing a fault-tolerant controller using active FTC.

FTC has become an important field of research in contemporary automation control [17]. Fault diagnosis enables the controllers to detect and localize the origin of faults, and it provides meaningful diagnostic information about the fault. FTC enables a system to compensate and maintain a relatively nominal operation or adapt its operations to a new operating strategy at a reduced level, instead of an immediate system failure when one or more components experience faults in situations where service and repair is not immediately available [17, 24, 25]. Therefore, a well-designed Fault-Tolerant controller should be able to: (a) compensate for foreseeable faults, (b) adapt to most faults that were not anticipated during design phase, and (c) enable the system to remain operational with minimal or no support from a human operator.

As the demands on system performance continue to increase, the complexity and the size of the systems also increases drastically. As the complexity increases it becomes practically impossible for engineers to design a perfect system and anticipate every possible situation the system may encounter, and hence the reliability of the system may decrease unless preventative and compensatory (i.e. fault-tolerant) measures are taken [23].

Active Fault-Tolerant Controls enable the controller to adapt to a fault within the system autonomously and allow the system to remain operational. This is essentially important for systems deployed in an environment where assistance and repair are not immediately available. For example, systems deployed in deep-space exploration or in nuclear disaster exclusion zones. With the help of optimisation technique such as Evolutionary Computation or machine learning, FTC should be able to autonomously redesign the controller to compensate for the fault when it occurs.

2.1.3 Existing Work on Fault-Tolerant Controller

Software controllers are algorithms used to drive the system to perform its tasks without it the system would just be a network of interconnected hardware. Throughout the years many controllers have been developed for various reasons. This chapter discusses some of the most popular software controllers developed to date which can be used in Fault-Tolerant Control (FTC), and the Fault-Tolerant Controllers developed for this research. This section provides brief descriptions of some published research and their proposed FTC for various systems.

Mora et al. [26, 27] research used Evolutionary Algorithm to convert FPGA into Evolvable Hardware, which was used in image processing as an FTC to remove noise (which was considered to be a fault) in an image. The research carried out by Dobai and Sekanina [28], implements an optimisation algorithms into a pre-existing Evolvable Hardware system, showed that the evolution time for a solution to converge had been significantly reduced from 8 minutes to less than 2 seconds. This work provides important insights that, by combining various optimisation techniques and/or machine learning algorithms, FPGA can be an effective fault-tolerant controller for hardware or software application.

Sedrine et al. [29] examined using GA optimisation as FTC for a five-phase flux-switching machine. Their experimental results showed that when a short-circuit occurred, the GA alone was sufficient to reconfigure the reference current to minimize torque ripple and copper loss, thus improving the operation of the machine in the presence of faults.

Merheb et al. [30] proposed combining passive FTC (using sliding-mode theory) and active FTC using a model reference adaptive approach to formulate a new FTC algorithm for unmanned aerial vehicles. Results from their simulations show that when faults occur during operation, small faults were tolerated by the passive FTC while large faults were adapted and compensated by the active FTC.

Li and Chen [31] investigated using a second-order sliding-mode control as a FTC to deal with external disturbance and internal faults, which are commonly found in digital computation and continuous-time systems.

Zhang and Gong [32] implemented an exponential-stability tolerant controller into Networked Control System as its fault-tolerant control, and showed consistent results to prove its effectiveness.

Dong et al. [24] introduced an adaptive FTC strategy based on a dual-quaternion description model introduced by Clifford [33] and finite-time control method theorised by Man et al. [34]. The proposed strategy is used for finite-time stability closed-loop spacecraft formation system. Dong et al. [24] suggested that the proposed adaptive FTC strategy can maintain high-precision formation control of the spacecraft, and adjust itself in minimal possible time while its operation remains online, without any need to completely redesign the controller every time a new situation is encountered.

Shen et al. [21] proposed a Takagi-Sugeno (T-S) fuzzy system-based FTC for gain bias faults in Near Space Hypersonic vehicle. T-S fuzzy system enabled the system to greatly simplify the analysis and synthesis of complex nonlinear systems, by combining several local linear models together to become a universal approximation for any smooth nonlinear function. In addition to building an FTC using T-S fuzzy system-based FTC, they proposed a neural network-based adaptive FTC for faults that cannot be modelled, and therefore the effect of the fault remains unknown. According to simulation results provided by the neural network-based adaptive FTC was able to handle un-modelled actuators faults; giving an advantage over passive FTC and traditional active Fault-Tolerant property; and minimizes the adverse effect of time delay due to fault diagnosis by not requiring a FD model, which is needed in typical active FTC to work.

Witczak [17] proposed using a multi-layered neural network for FD to significantly reduce uncertainty in fault diagnosis for nonlinear, discrete-time Multiple-Input-Multiple-Output systems. In addition, he proposed two FTC designs which are: multiple-model T-S fuzzy system approach similar to Shen et.al. [21] and H_∞ approach similar to [35, 36]. Aghababa et al. [35] considered using a multiple-input-multiple-output linearization technique to develop FTC for nonlinear multiple-machine power systems that can handle scenarios such as multiple actuator failures. Where Jiantao el al. [36] applies the H_∞ methodology to reconfigure the collective formation and coordinators of multi-vehicle systems such as Unmanned Aerial Vehicles (UAVs) and satellites, where information transferred between adjacent vehicles is disturbed and

individuals are subjected to faults caused by stochastic disturbance, measurement noise and actuator fault.

Mallavalli and Fekih [37] proposed using a Super-Twisting control algorithm and Higher Order Sliding Mode Observer design as real-time continuous FTC for UAV, to battle with faults such as disturbance and actuator failures.

Wafa et al. [38] designed a Proportional–Integral–Derivative and Fuzzy Logic Controller hybrid FTC for Electronic Throttle Valve described by a switched discrete-time system with external input disturbance and internal actuator faults, that is observed by unknown input observers. In addition, they used a genetic algorithm to optimise the output of the controller.

Yang et al. [39] proposed implementing an adaptive Proportional–Integral controller as passive FTC instead of the traditional passive FTC (i.e. by redundancy), for nonlinear satellite attitude systems with unknown external disturbance and actuator fault. Simulation results showed that the Proportional–Integral controllers are capable of rapidly converging to a working solution, and are considered as an effective passive FTC.

Machmudah et al. [40] compared results generated using GA and PSO, it focuses on comparing the efficiency of the algorithms to generate optimal trajectories for a 6^{th}-degree polynomial joint angle path and total travelling time under kinodynamic constraints for a 3 degree of freedom planar robotic arm. It provides meaningful insights to design FTC for planar systems, and the algorithm described in the paper can be transfer from optimal trajectories planning to FTC.

2.2 Machine Learning

Professor Tom Mitchell defines the discipline of machine learning as a natural outgrowth of the intersection of Computer Science and Statistics [11, 41]. Machine learning is a branch of artificial intelligence that aims at enabling machines to perform their jobs skilfully by using intelligential software [42]. Machine learning focuses on what computational architectures and algorithms can be used to most effectively capture, store, index, retrieve and merge data, how multiple learning subtasks can be orchestrated in a larger system, and computational tractability [42]. The goal of machine learning is to construct a model that takes the input and produces the desired

results, the resultant model can sometimes be easily understood and explained by its human counterpart, whereas, at other times it's like a black box and could not be intuitively explained. The model is an approximation of the human intelligent process the machines needs to mimic [42], hence an error function from the approximated result can be obtained to determine the performance of the model. Current machine learning algorithms belong to one of four techniques, which are:

1. *Supervised Learning:* where machine learning is tasked to learn and infer a function that maps an input to an output based on labelled training data consisted of a set of training examples. The label from the output data is the explanation of its respective input example and it is provided by an external (normally human) supervisor. Regression and Classification are two of the most classic supervised learning algorithms.
2. *Unsupervised Learning:* this type of machine learning works without supervisors or training data, and unlabelled data. Therefore, unsupervised learning is commonly used to find hidden structures in the data. An example of unsupervised learning algorithm is clustering, where items that are like each other are clustered together.
3. *Semi-supervised Learning:* this type of machine learning utilizes a mixture of classified and unclassified data, which allows it to learn and generate an appropriate model for the classification of data. The model then can be used to predict classes of future test data better than the model generated by using supervised learning alone.
4. *Reinforcement Learning:* this type of machine learning uses observations gathered from the interaction with the environment to maximize rewards and minimise risk.

Of the four machine learning techniques described, reinforcement learning is a promising technique to be used in developing FTC, as it generates its model via interaction with its environment and does not require labelled data to function.

2.2.1 Evolutionary Computation

Evolutionary Computation has been developed from when researchers took Nature's solution as an inspiration to develop automated problem solvers and AI algorithms [11, 43-45]. Two powerful problem solvers found in nature are the human brain and evolution. Evolutionary computation is the study of non-deterministic search

algorithms that are based on and inspired by Darwin's: *'On the Origin of Soecies'* [11, 43, 46]. Early pioneers such as John Holland, Ingo Rechenberg, Hans-Paul Schwefel and Lawrence Fogel have laid important groundwork in evolutionary computation [11, 43-45, 47], allowing it to become an effective search and optimization technique:

1. Holland strongly emphasized in his work the importance of the crossover operator.
2. Rechenbery and Schwefel discovered in their work that for optimization problems, small random mutations of the model's variables are proven to be an effective optimization technique, and so they gave rise to evolutionary strategy.
3. Fogel studied the possibility of evolving finite-state machines to predict symbol strings of symbols termed as chromosomes, generated by Markov processes and non-stationary time series.

The computerisation age during the second half of the twentieth century has witnessed exponential growth in demand for automated problem-solving algorithms and the complexity of the problem it is trying to solve. Research and development cannot keep up with these demands, while development time and resources available are constantly decreasing [43]. This calls for a robust algorithm that is applicable to a wide range of problems, which does not require much modification for a specific problem and has a satisfactory performance to deliver a working solution within an acceptable timeframe [43]. The performance of these algorithms such as the expert system is no longer able to meet current demands, as classic algorithms typically require a significant time and resources to develop and/or require a significant lengthy period to solve a complex problem. Whereas, Evolutionary Computation demonstrates promising signs of being able to solve ever-increasing sets of problems (in size and complexity) in a shorter amount of time. One of the reasons why Evolutionary Computation can solve complex problems at a much faster rate than classic automation systems, lies within its structure and architecture. Evolutionary Computation is capable of quickly producing a *'good enough'* solution (also known as approximated solution) that is not necessarily optimized. Whereas the classic approach typically requires precise modelling of the problem and constraints ridden approaches, to ensure the optimization of its solution. When faults cause a deviation significant enough, traditional methods are often unable to comprehend and correct the system. By contrast, Evolutionary Computation can simply generate a new set of approximated solutions adjusting to the deviation. Several

examples of complex problems that Evolutionary Computation is capable of solving are optimization problems such as the well-known *'Travelling Salesman Problem'* [11, 43-46, 48], where the model and desired output is known and it is the Evolutionary Computation's task to configure the required path to produce the desired output. Classification or system identification problems can also be evolved where, corresponding inputs and outputs are known and it's up to Evolutionary Computation to compute the required model to link the correct output for each known input. An example of such problems is objections identification within a picture. The simulation problem, in which the inputs and the system model are known and it is used to calculate the output corresponding to the provided input, for example calculating the forward and/or inverse kinematics of a robotic system for it to perform a certain task.

2.2.2 Genetic Algorithm

GA is a metaheuristic optimization technique which has its roots in biological evolution and reproduction, and it is one of the most widely-known types of evolutionary computation [47, 49, 50]. Biological evolution uses the concepts of survival of the fittest to transfer its DNA to the next generation via reproduction, when a new generation of offspring is created each gene in the child chromosomes, has a chance for random mutation to occur. GA was first introduced by John Holland [46, 51], it was well received and since been extensively researched and widely adapted into many fields such as engineering, computer science and mathematics, both in literature and commercially GA is applied by first generating a population of random possible solutions known as chromosomes, the initial population chromosomes can be computed in forms of:

- Weightings and network connections approach using ANN.
- Parameters and size of a LUT.
- Membership functions, fuzzy rules and logic of a FLC.
- Configuration bit-stream of evolvable hardware.

Each chromosome is evaluated to determine its fitness to solve the problem, i.e. how well the solution can solve or perform a specific task if this chromosome is used. Chromosomes are selected and paired together with the chromosomes with the better fitness kept, this method is known as *tournament selection*. At each generation pairs of parent chromosomes are used to reproduce pairs of children chromosomes, and each

gene within the child chromosomes there is a chance for random mutation to occur. There are numerous literature reports [43, 49, 50, 52, 53] which discuss different reproduction methods used on a GA to improve its performance. The most common reproduction methods currently being used are *mutation* and *crossover*. Crossover has several methods such as: single-point crossover, two-/multi- point crossover and array uniform crossover [54]. Finally, the process of evaluating the fitness, selection and reproduction are repeated until the chromosome's fitness reaches predefined optimize level, or a predefined generation level is reached. Then, the chromosome with the best fitness level is the optimized solution and adapted into the system [49, 50, 53-57].

2.2.3 Particle-Swarm Optimisation

Particle Swarm Optimisation is a type of Swarm Intelligence, which itself is a subclass of Evolutionary Computation [58] that has gained popularity and interest from researchers across many disciplines. Swarm Intelligence is the collective intelligence behaviour of self-organized and decentralized systems borrowed from behaviours of social insects, bird flocking, ant colonies, animal herding, bacterial growth and fish schooling [58, 59]. Flocks in Swarm Intelligence follow very simple rules [58], as there is no centralised control structure dictating how an individual flock should behave. Individual flock behaviour is local to itself, while social interaction between flocks leads to the convergence of a global behaviour that is unknown to the individual flock [49]. Self-organization relies on the feedback, fluctuations and multiple interactions between an individual and its neighbours [59]. Feedback (either positive or negative) provides amplifications and stabilizations to the algorithm, fluctuations introduce heuristic and randomness to the algorithm, to prevent the solution converging to a local optimum instead of global optimum. The decentralisation of system allows the Swarm Intelligence to divide the problem into smaller individual tasks, while each individual is set to solve its own task, they also collectively work together to converge into a global solution [59]. Since its conception, many different types of Swarm Intelligence algorithms are being developed: the majority of them are collectively known as *Artificial Life*, PSO was introduced by Kennedy and Eberhart [58-60], Artificial Ant Colony proposed by Marco Dorigo [59, 61] and Artificial Bee Colony proposed by Dervis Karaboga [59, 62] are some of the most popular SI algorithms.

PSO is an optimisation technique that mimics swarm behaviour to guide the particles to search for a global optimal solution, and it is made up of three major mechanisms:

1. *Separation:* the behaviour of avoiding crowded local flocks. That is done by randomly assigning the initial position and velocity of each local flock. This is done to reduce the chance of converging to a local optimum as the result.
2. *Alignment:* the behaviour of moving towards the average direction of local flock, as local flock moves in the direction towards the global optimal.
3. *Cohesion:* the behaviour of moving towards the average position of local flock, with the average position and velocity of movement updated at every iteration.

PSO are said to be an efficient optimisation algorithm by searching an entire high-dimensional problem space, due to its stochastic optimisation technique based on the movement and intelligence of the swarms collectively. By applying the concept of social interaction to the problem space, global optimum is updated by choosing the local flock with the best optimum while all other flocks move towards that direction. Further, unlike other classical optimisation techniques, it does not require the gradient of the problem being optimised. The main points of PSO are that it is:

- Simple to implement.
- Only has a few parameters to be set.
- Effective in global search.
- Insensitive to scaling of design variables
- Easily parallelised to concurrent processing.

PSO has the tendency to result in a fast and premature convergence to some mid-optimum point and having slow convergences in a refined search area. These issues need to be further addressed and adjusted for the problem it is trying to solve, for it to become more efficient.

2.2.4 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) was invented by Julian Miller [46] to evolve digital circuits, which is a subclass of another member of evolutionary computing algorithm known as Genetic Programming [63-65]. The two most prominent differences between CGP and EC are:

1. CGP uses directed acyclic graphs known as '*trees'* (a two-dimensional grid of computational nodes) in replacement of chromosomes.
2. EC typically is used to find inputs to achieve maximum payoff, whereas CGP is used to seek models with maximum fit.

However, for modelling problems that seek maximum fit it can be considered as a special case of optimization. Therefore these models are treated as individuals and its fitness is the model quality to be optimized via evolution. Since typical CGP utilize trees as chromosomes to represent the models it tries to optimize, the two important differences in model representation between other evolutionary computation such as CGP and GA are:

1. Chromosomes in CGP are nonlinear structures, whereas GA are typically linear vectors.
2. CGP chromosomes can be different size measured by the number of nodes of the tree, whereas chromosome sizes in GA are typically fixed.

Strictly speaking CGP shares identical operation procedures compare to GA, with some minute alteration in crossover and mutation to accompany the tree of chromosomes [65-68]. For CGP, mutation and crossover both occur at random junctions of the tree.

In summary, this chapter provides a detailed literature review on existing work related to FTC, FD and Fault-Tolerant controller. In addition, a review of several popular machine learning algorithms is also included in this chapter.

Chapter 3

Simulation and Kinematics of the Robotic Arm

3.1 Simulation of the Robotic Arm

This research is based on a simulation of a serial chain planar robotic arm developed using C# language in Microsoft Visual Studio. Figure 3.1 is the schematic of the simulated robotic arm system developed for this research.

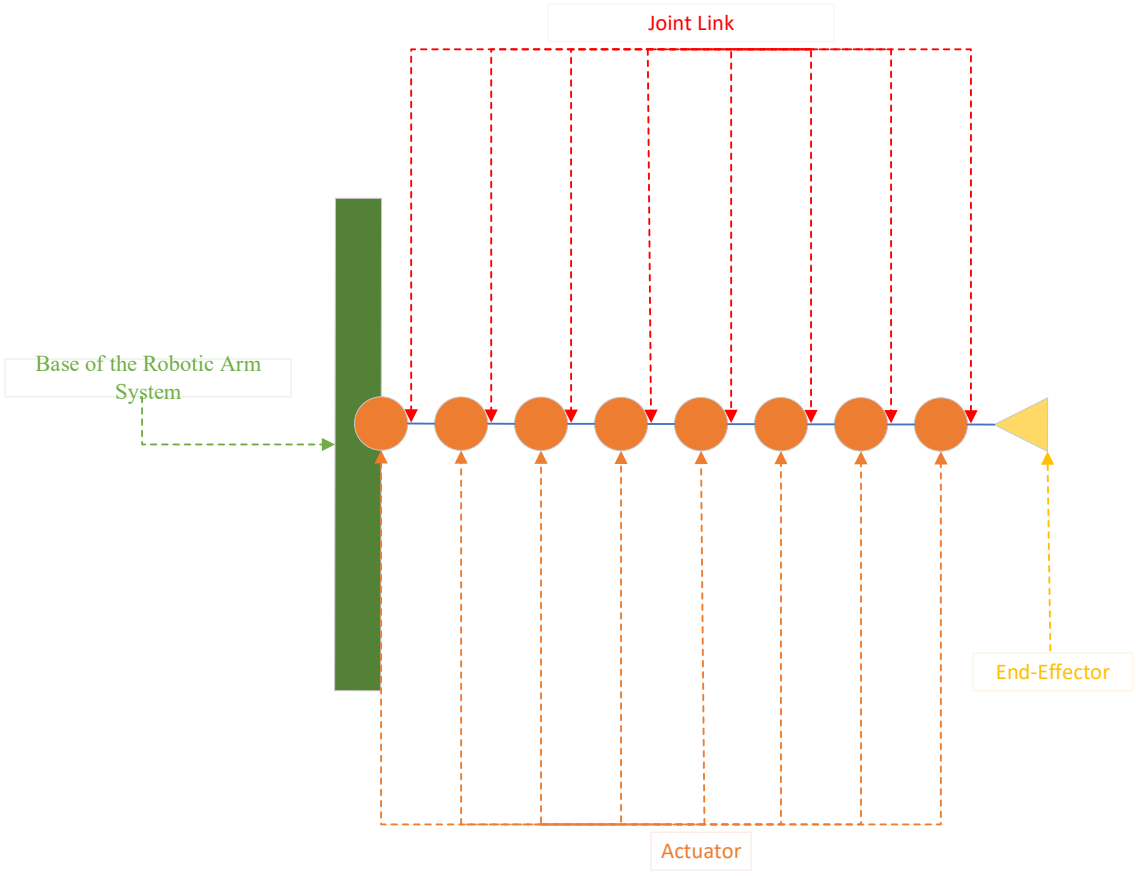


Figure 3.1. Schematic of the Simulated Robotic Arm

The simulation developed for this research is presented in the form of a Graphical User Interface (GUI) as shown in Figure 3.2 in which the user can:

- Manually drive each individual joint of the system.
- Select the combination of fault(s) presented in the system.
- Select the type of Fault-Tolerant controller to use to generate a new set of solutions for the given fault(s) combination.

- Visually and numerically observe the convergence of the new solution of the controller.

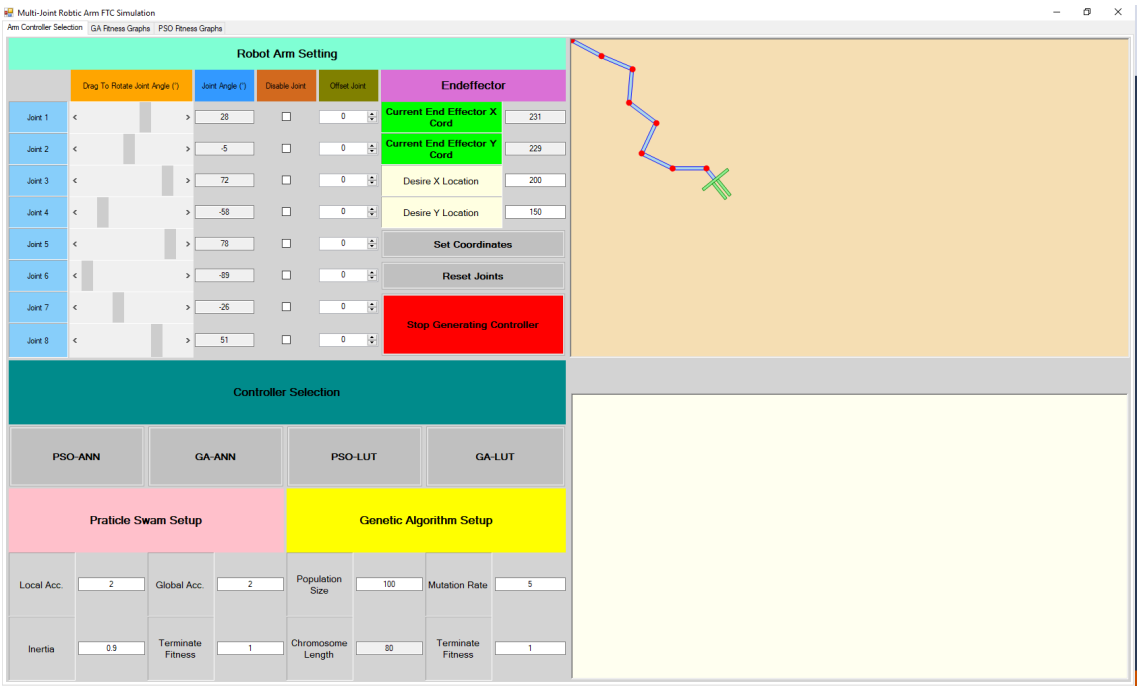


Figure 3.2. GUI of Multi-joint Planar Robotic Arm System Simulator

3.2 Kinematics of the Robotic Arm

A serial chain planar manipulator is a system in which all its moving links move perpendicular to the X-Y plane. The system developed for this research uses an 8- joint configuration, each link has equal length and a gripper at the end of the manipulator. Each revolute joint is driven by an actuator that can rotate between $-45^{\circ} \sim 45^{\circ}$ on the Z-axis and has a single DOF. For an open-loop serial-chain planar robotic arm system, the mobility of a chain is the number of DOF and can be calculated using equation (3.1) [69, 70].

$$M = \sum_{i=1}^{n} f_i \tag{3.1}$$

where,

M = mobility

n = number of joints

f_i = connectivity of joint

For the multi-joint planar robotic arm: $n = 8,\ f_i = 1$, thus:

$$M = \sum_{i=1}^{8} 1 = 8$$

For the closed-loop serial chain planar robotic arm system, the number of DOF can be calculated using equation (3.2) [69; Chapter 1, 70; Chapter 1]:

$$n_{dof} = \lambda(n-k) + \sum_{i=1}^{k} f_i \tag{3.2}$$

where,

n_{dof} = number of Degrees of Freedom.

λ = number of independent kinematics constraint

n = number of joints.

f_i = connectivity of joint.

k = number of links.

For the multi-joint planar robotic arm: $n = 8,\ \lambda = 6,\ f_i = 1, k = 8$, thus:

$$n_{dof} = 6(8-8) + \sum_{i=1}^{8} 1 = 0 + 8 = 8$$

Kinematics studies the motion of bodies, without consideration of the forces or moments that causes the motion. Formulating the suitable kinematics models for the mechanism of the system is very crucial for analysing the behaviour of the system. Kinematics of the system can typically be modelled using Cartesian or Quaternion space [71; Chapter 5]. The study of kinematics can be broken down into forward kinematics and inverse kinematics. Forward kinematics is the study of position and orientations of the end-effector in Cartesian space, from the known joint variables in the Joint space. Inverse kinematics is the study of calculating the required joint variables in the Joint space, for the system to reach the desired coordinates and orientation in the Cartesian space. Inverse kinematics problems typically are solved by using analytical or numerical methods. Analytical methods typically utilise either geometric or algebraic approach to analytically find the required joint variables of a given configuration. Inverse kinematics is much harder to solve, computationally more

expensive and takes longer to solve when compared to forward kinematics. Therefore only a very small class of simple systems have complete analytical solutions to an inverse kinematics problem [71; Chapter 5]. The inverse kinematics problem is considered hard to solve because [71; Chapter 5]:

- Kinematics equations are coupled, which increases the difficulties significantly.
- Multiple solutions and singularities exist.
- Mathematical solutions for inverse kinematics problems may not be physically suitable, and a suitable method for its solution depends on the structure of the system.

3.2.1 Forward Kinematics

To analyse the forward kinematics of the system, it is required to attach a coordinate frame to each link and then use it to represent relationships between the links [71; Chapter 5, 72, 73]. Thus, each link has its own set of coordinate frame $o_i\, x_i\, y_i\, z_i$, where,

o_i, the orientation of link *i*
x_i, the *x* coordinate of link *i*
y_i, the *y* coordinate of link *i*
z_i, the *z* coordinate of link *i*

When $joint\ i$ is actuated, $link\ i$ and its coordinate frame $o_i x_i y_i z_i$ experience a resulting motion. Otherwise, the coordinates of each point on $link\ i$ remain constant when it is expressed in the i^{th} coordinates frame.

In addition, the following assumption of the joints of a serial chain system can only be either revolute or prismatic such that,

- For revolute joints, rotation (θ) is its variable of movement

For prismatic joints, translation (d) is the variable of movement.

A homogeneous transformation, A_i, which can be used to expresses the position and orientation of $\theta_i x_i y_i z_i$ with respect to $\theta_j x_j y_j z_j$ is known as *transformation matrix*, T_j^i, [71; Chapter 5] where,

$$T_j^i = A_{i+1}A_{i+2}A_{i+3} \dots A_{j-1}A_j \, , if \; i < j$$

$$T_j^i = I \, , if \; i = j$$

$$T_j^i = (T_j^i)^{-1} \, , if \; j > i \tag{3.3}$$

Note Ai is the transformation of joint I with respect to joint i-1

T_j^i expresses the position of joint j wrt to joint i

The transformation matrix described in equation (3.3) enables the attachment of various frames to the corresponding links and follows the position of any point on the end-effector. When it is expressed in coordinate frame, it's constant is independent to the configuration of the system. The position and orientation of the end-effector with respect to the base frame can be calculated using the homogeneous transformation matrix given in equation (3.4).

$$H = \begin{bmatrix} R_n^0 & O_n^0 \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

where,

O_n^0, is a 3×1 vector holding the x, y, z coordinates of the origin of the end-effector frame with respect to the base frame.

R_n^0, is a 3×3 rotation matrix from the homogeneous transformation matrix given in equation (3.4), the position and orientation of the end-effector in the base frame can be derived as given in equation (3.5).

$$H = T_n^0 = A_1(q_1)A_2(q_2)A_3(q_3) \dots A_i(q_i) \tag{3.5}$$

where,

$$q_i = \begin{cases} \theta_i, & \text{joint i revolute displacement} \\ d_i, & \text{joint i prismatic displacement} \end{cases}$$

Thus, the homogeneous transformation A_i can be rewritten as equation (3.6).

$$A_i = \begin{bmatrix} R_i^{i-1} & O_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

And the transformation matrix can be updated to equation (3.7).

$$T_j^i = A_{i+1}A_{i+2}A_{i+3} \dots A_j = \begin{bmatrix} R_j^i & O_j^i \\ 0 & 1 \end{bmatrix} \tag{3.7}$$

The rotational matrix, R_j^i, is used to express the orientation of $o_j x_j y_j z_j$ relative to $o_i x_i y_i z_i$, and it is calculated using equation (3.8).

$$R_j^i = R_{i+1}^i \dots R_j^{j-1} \tag{3.8}$$

The coordinate vector, O_j^i, of each join can be calculated recursively using equation (3.9).

$$O_j^i = O_{j-1}^i + R_{j-1}^i O_j^{j-1} \tag{3.9}$$

By calculating the function $A_i(q_i)$ of each link and finding its product whenever it is required, the forward kinematics of the system and the position and orientation of the end-effector can be determined. In short, forward kinematics can be seen as recalculating the set of coordinate points between two interconnected joints by using a series of matrix operations when motion had occurred. However, the before mentioned technique for finding forward kinematics can be cumbersome and tedious to calculate. Therefore, conventions such as the Denavit-Hartenberg (D-H) representations of a joint, are introduced, as it provides a significant improvement in streamlining and simplification in calculation.

3.2.1.1 Denavit-Hartenberg Convention

The D-H representation is a convention for assigning and manipulating the coordinate frames. The D-H parameters uses four variables to define the reference coordinate frame for each link in a serial chain manipulator [71; Chapter 5, 72, 73]. It defines a homogeneous transformation matrix, which can used to analyse kinematics (both forward and backward) of the manipulator, numerically and analytically. Forward kinematics is used to describe the relationship between the individual joints of the manipulator and the position and orientation of the end-effector, such that it determines the position and orientation of the end-effector given the values for the joint variable of the robot. Inverse kinematics is used to determine values for the joint variables to achieve a desired position and orientation for the end-effector.

The homogeneous transformation matrix, A_i, itself is a product of four simple transformation, as given in equation (3.10) [71; Chapter 5, 72, 73].

$$A_i = rot_{z\,\theta_i} trans_{z\,d_i} trans_{x\,r_i} rot_{x\,\alpha_i} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

where,

$c_\theta = \cos(\theta)$
$s_\theta = \sin(\theta)$
$\theta_i =$ rotation of revolute joint *i*
$\alpha_i =$ rotation of link *i*
$d_i =$ offset of prismatic joint *i*
$a_i =$ length of link *i*

The rotation matrix that rotates about the Z axis for joint *i* is given by equation (3.11):

$$rot_{z\,\theta_i} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

The translation matrix that translates along the Z axis for joint *i* is given by equation (3.12):

$$trans_{z\,d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

The translation matrix, that translates along the X axis for link *i* is given by equation (3.13):

$$trans_{x\,r_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

The rotation matrix, that rotates about the X axis for link *i* is given by equation (3.14):

$$rot_{x\,\alpha_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

The homogeneous transformation matrix is described in equation (3.6), which requires 6 variables to be fully defined, whereas the homogeneous transformation matrix under D-H convention as given in equation (3.3) requires only 4 parameters ($\theta_i \alpha_i d_i a_i$). Using D-H representation gives the user a considerable freedom to choose the origin and the coordinate axes of the reference frame, as long as it is rigidly attached to link i and remain consistent when applied across all the link's reference frame. Then the frame can be placed anywhere within the physical link or in free space [71; Chapter 5].

In addition to the 4 parameters described in equation (3.10), for the D-H convention to be fully defined, two additional conditions a need to be satisfied when assigning of the coordinate frames, which are:

- The *X*-axis of the current frame must be perpendicular to the *Z*-axis of the previous frame.
- The *X*-axis of the current frame must intersect the *Z*-axis of the previous frame.

For any given robot manipulator, there are multiple ways to choose the frames that satisfy the two conditions above, in which some are more intuitive than others. Thus, it is entirely possible for different people to derive differing, but equally correct, coordinate frame assignments for the same system. However, regardless of the assignment of the intermediate link frames, the same end-results matrix (T_n^0) will always converge.

Calculating the forward kinematics for any manipulator using the D-H convention, can be done using the following procedure:

1. Set and label the joint axes $Z_0 \dots, z_{n-1}$
2. Establish the base frame and set the origin anywhere on the z_0-axis. Using the origin, the x_0 and y_0 axes can be established using the right-hand rule.
3. Set the origin O_i that is normal to z_i and z_{i-1} intersects Z_i. If z_i intersects z_{i-1}, set O_i at this intersection. If z_i and z_{i-1} are parallel, O_i can be set anywhere along z_i.
4. Set x_i along the common normal between z_{i-1} and z_i through O_i or in the direction normal to the $z_{i-1} - z_i$ plane if z_{i-1} and z_i intersect.
5. Establish y_i using the right-hand rule to complete the frame
6. Repeat Steps 3 to 5 to establish the coordinates for the remaining joints i, for $i = 1, \dots, n-1$.
7. Establish the end-effector frame, which consists of the 4 D-H parameters $o_n x_n y_n z_n$. where,
    i. Assuming the end-effector joint is revolute, then set $z_n = a$ along the direction z_{n-1}.
    ii. Set the origin o_n along z_n
    iii. x_n and y_n of the end-effector can be calculated using:

a) Set $y_n = s$ the direction of the end-effector closure and set $x_n = n = s \times a$, if end-effector is a gripper

b) The right-hand frame rule, for non-gripper end effector

8. Calculate the link parameters $a_i, d_i, \alpha_i, \theta_i$ where,
   - $a_i =$ distance along x_i from O_i to the intersection of the x_i and z_{i-1} axes
   - $d_i =$ distance along z_{i-1} from O_{i-1} to the intersection of the x_i and z_{i-1} axes. d_i is variable if *joint i*, is prismatic.
   - $\alpha_i =$ the angle between z_{i-1} and z_i measured about x_i
   - $\theta_i =$ the angle between x_{i-1} and x_i measured about z_{i-1}. θ_i is variable if *joint i* is revolute.
9. Substitute the parameters in step 8 into the homogeneous transformation matrices described in equation (3.10).
10. Use the transformation matrix T_n^0 described in equation (3.7), to calculate the position and orientation of the end-effector relative to the base coordinates.

An example of establishing the frames for a 3-joints system using the D-H convention procedure above is provided by Wittenburg [71; Chapter 5], as shown in Figure 3.3. The joint axes z_0 and z_1 in Figure 3.3 are normal to the page and the base frame $o_0x_0y_0z_0$ is located at the first joint. After the base frame is established, the frame $o_1x_1y_1z_1$ can be established by using the D-H convention, and the origin O_1 is located at the intersection of z_1 and the page. The final frame $o_2x_2y_2z_2$ is established after locating the origin O_2 at the end of link 2.

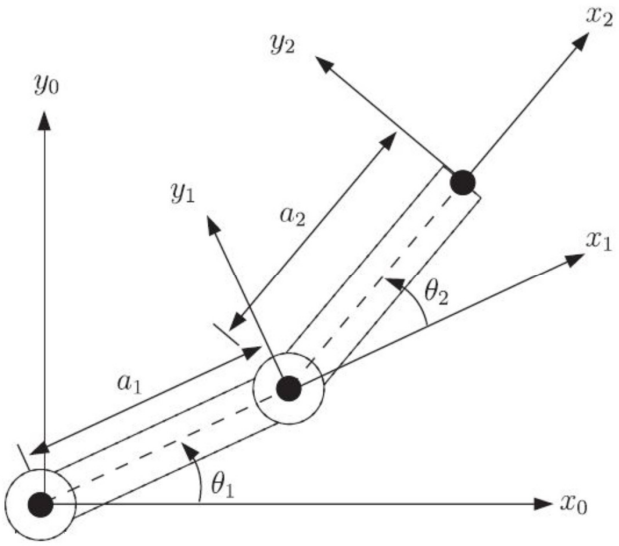


Figure 3.3. Example of Establishing the Reference Frame for a 3-joints System Using D-H Convention

3.2.1.2 Forward Kinematics of the System Using D-H Convention

Using the D-H convention, the kinematics of the 8-link planar manipulator system developed for this research can be established in Table 3-1.

Table 3-1. Link Parameters for 8-link Planar System

Link	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3
4	a_4	0	0	θ_4
5	a_5	0	0	θ_5
6	a_6	0	0	θ_6
7	a_7	0	0	θ_7
8	a_8	0	0	θ_8

The homogeneous transformation matrix of the links, which is given in equation (3.15) to equation (3.22), can be determined by using equation (3.10),

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$A_3 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & a_4c_4 \\ s_4 & c_4 & 0 & a_4s_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$A_5 = \begin{bmatrix} c_5 & -s_5 & 0 & a_5c_5 \\ s_5 & c_5 & 0 & a_5s_5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$A_6 = \begin{bmatrix} c_6 & -s_6 & 0 & a_6c_6 \\ s_6 & c_6 & 0 & a_6s_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

$$A_7 = \begin{bmatrix} c_7 & -s_7 & 0 & a_7c_7 \\ s_7 & c_7 & 0 & a_7s_7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

$$A_8 = \begin{bmatrix} c_8 & -s_8 & 0 & a_8 c_8 \\ s_8 & c_8 & 0 & a_8 s_8 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

where,

$c_i = \cos\emptyset_i$, for $i = 1,2,\ldots,8$

$s_i = \sin\emptyset_i$, for $i = 1,2,\ldots,8$

a_i = length of link i, for $i = 1,2,\ldots,8$

By substituting the homogeneous transformation matrices (described in equation (3.15) to equation (3.22)) into equation (3.6), the transformation matrix, T_0^7, of the system is derived as given in equation (3.23) and the forward kinematics of the system can be calculated.

$$T_8^0 = A_1A_2A_3A_4A_5A_6A_7A_8 = \begin{bmatrix} c_{\sum_{i=1}^{8} i} & -s_{\sum_{i=1}^{8} i} & 0 & \sum_{i=1}^{8} a_i c_i \\ s_{\sum_{i=1}^{8} i} & c_{\sum_{i=1}^{8} i} & 0 & \sum_{i=1}^{8} a_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

where,

$c_{\sum_{i=1}^{8} i} = \cos(\theta_1 + \theta_2 + \theta_3 + \cdots + \theta_8)$

$s_{\sum_{i=1}^{8} i} = \sin(\theta_1 + \theta_2 + \theta_3 + \cdots + \theta_8)$

$$\sum_{i=1}^{8} a_i c_i = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3) + \cdots + a_8 \cos(\theta_1 + \theta_2 + \theta_3 + \cdots + \theta_8)$$

$$\sum_{i=1}^{8} a_i s_i = a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) + a_3 \sin(\theta_1 + \theta_2 + \theta_3) + \cdots + a_8 \sin(\theta_1 + \theta_2 + \theta_3 + \cdots + \theta_8)$$

In summary, this chapter described in detail the kinematics of the robotic arm system. This chapter also described in detail the GUI of Fault-Tolerant Controller and the simulation of the robotic arm system were developed for this research.

Chapter 4
Design of the Fault-Tolerant Controllers and Associated Systems

This chapter presents a detailed description of the Fault-Tolerant Controllers developed and used for this research.

4.1 Genetic Algorithm

Genetic Algorithm (GA) is a metaheuristic optimisation technique for solving both constrained and unconstrained problems that was introduced by John Holland, which was based on Darwin's theory of evolution and natural selection [51]. At each generation, GA utilises three basic operations, which are: (i) Selection, (ii) Reproduction, and (iii) Fitness Evaluation, for evolving a population of individual solutions, known as chromosomes, until the termination criteria are achieved either, by an individual chromosome's fitness function meeting the optimal solution criteria, or other termination criteria such as maximum generations met [74]. The architecture of a GA algorithm is shown in Figure 4.1.

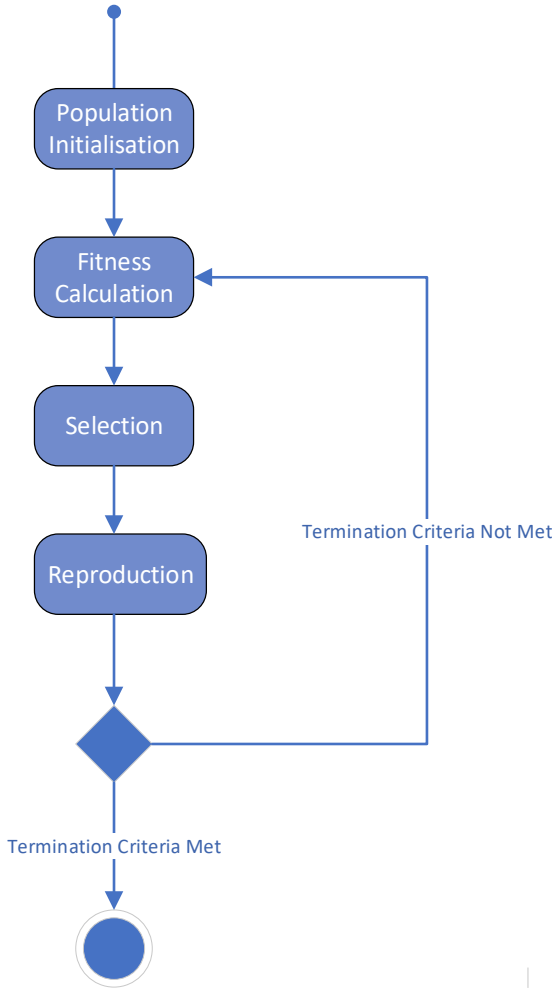


Figure 4.1. Architecture of a Genetic Algorithm

GA is based on a population of candidate solutions that represents a solution to the optimisation problem that it is trying to solve [49]. The population of solution candidates, commonly referred to as genotype or chromosome, are either initialised randomly or manually entered [49, 75]. After the population is initialised, at every generation the GA applies its three operators (Selection, Reproduction, and Fitness Evaluation) to the population of candidate solutions to reproduce a new set of offspring population. The parent chromosomes are updated and replaced by the offspring chromosome, if an offspring chromosome outperformed a parent chromosome. This process of continuously using the operators to generate new sets of offspring chromosomes is repeated until a termination criterion is met. Examples of termination criteria can be that an optimal solution is found or when the algorithm exceeds a certain number of generations.

4.1.1 Fitness Function

The fitness function used for this research is the distance from the gripper to the desired set point, calculated using the Euclidean error distance given in equation (4.1).

$$fitness = \sqrt{(x_{desire} - x)^2 + (y_{desire} - y)^2} \quad (4.1)$$

where,

x_{desire} is the x coordinate the system needs to reach

x is the x coordinate where the chromosome produces the best fitness

y_{desire} is the y coordinate the system needs to reach

y is the y coordinate where the chromosome produce the best fitness

The fitness used in GA is calculated from evaluating each individual chromosome on a fitness function [49]. The fitness function measures the quality of each candidate solution in the population. The fitness function differs according to its application, and it could have either single or multiple objectives that it needs to achieve. The purpose of implementing a fitness function is to determine whether an offspring should be kept, or did a solution converge to a global optimal and thus the algorithm had met the termination criteria.

4.1.2 Selection

This research only considered using Elitist based Tournament, as the selection method for all the GA-based Fault-Tolerant Controller developed. The selection method determines how to select individuals from the parent chromosome population to reproduce a new generation of offspring chromosome [49, 74, 75]. There are several selection methods available to choose from, the most common selection methods are: (a) Tournament, (b) Roulette Wheel, (c) Deterministic Sampling and (d) Stochastic Remainder Sampling [49, 74, 75]. Each selection method has its own unique characteristics, despite it drawing its results from the fitness level of each individual chromosome. Reed and Minsker [74] points out that Tournament is more efficient and less prone to premature convergence relative to other selection methods. While Oliver [49] suggested that Roulette wheel enables the parent solutions to be randomly selected

in a uniform distribution, and it has the advantage that each individual from the parent chromosome has a positive probability of being selected to reproduce. Elitist Tournament selects random groups of parent chromosomes from the population, with the best individual of that group being selected [49, 74, 75]. There are multiple ways for how individuals within the children population are selected to compare against each other, to become the parent chromosome in the next generation. One common method is to select randomly a number of individuals from the children population, and only the one with the best fitness is selected to become a parent in the next generation. If the child has a better fitness value than the parent, the parent is replaced by the child, the updated parent population will be used to reproduce a new set of children population in the next generation, the child that did not successfully replace a parent is discarded.

4.1.3 Crossover and Mutation

This research uses two-point crossover and 5% mutation rate on the four Fault-Tolerant controllers developed. Crossover is the operator used to combine two parents' chromosomes in order to generate two new children chromosomes. After the selection operator has selected and sorted the parent population to be used to reproduce a new children population, the crossover operator couples chromosomes of the parent population to mate with a specified crossover probability [49, 51, 74-76]. There are multiple methods as to how crossover occurs, some of the most common methods used are one-cut, two- or multi- cut points, and uniform crossover as researched by Vasconcelos et al. [75] and Reed et al. [74], Srinivas and Patnaik [76] recommended an adaptive probabilities approach

One-cut point crossover is the method in which a random crossover point is selected, and the parent chromosomes are swapped at the cut point to reproduce two children chromosomes. A visual representation of single cut-point crossover, shown in Figure 4.2.

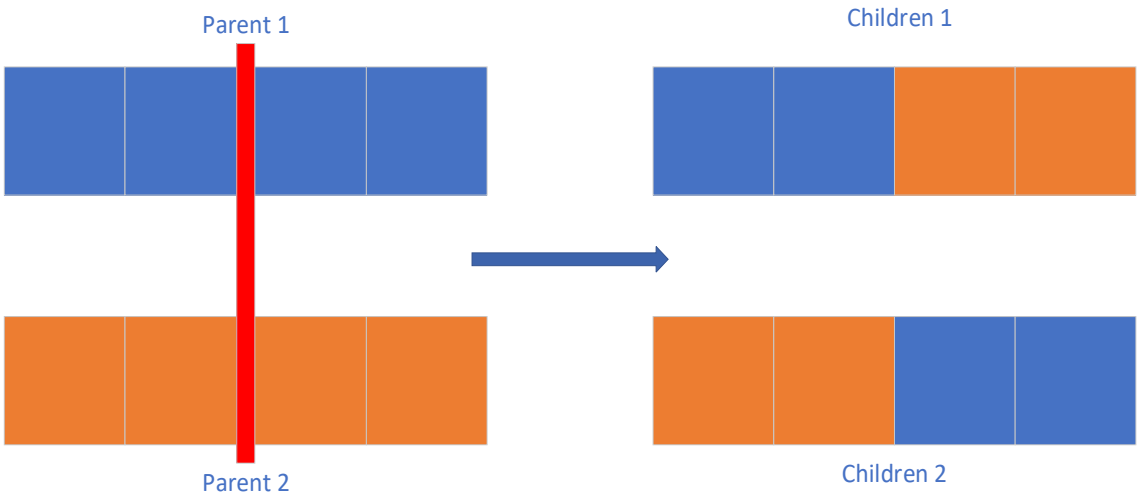


Figure 4.2. Single Cut Point Crossover

Multiple or two cut point crossover is a generalisation of the one cut-point crossover where multiple cut points are randomly chosen, and alternating segments are swapped to reproduce the two new children offspring. A visual representation of two cut-point crossover, shown in Figure 4.3.

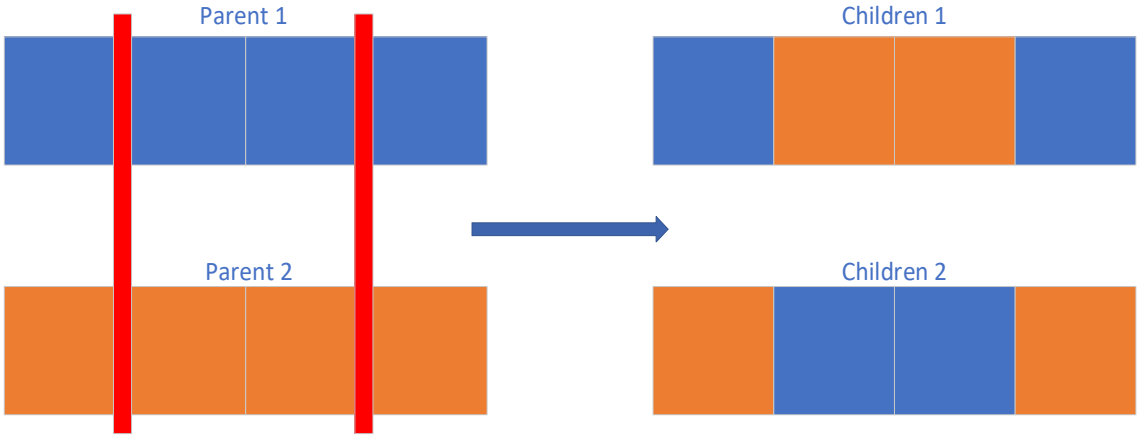


Figure 4.3. Multiple- / Two- Cut Point Crossover

For uniform crossover, instead of randomly segmenting the parent chromosomes, they are segmented and crossover in a uniform manner. A visual representation of a uniform crossover is shown in Figure 4.4.

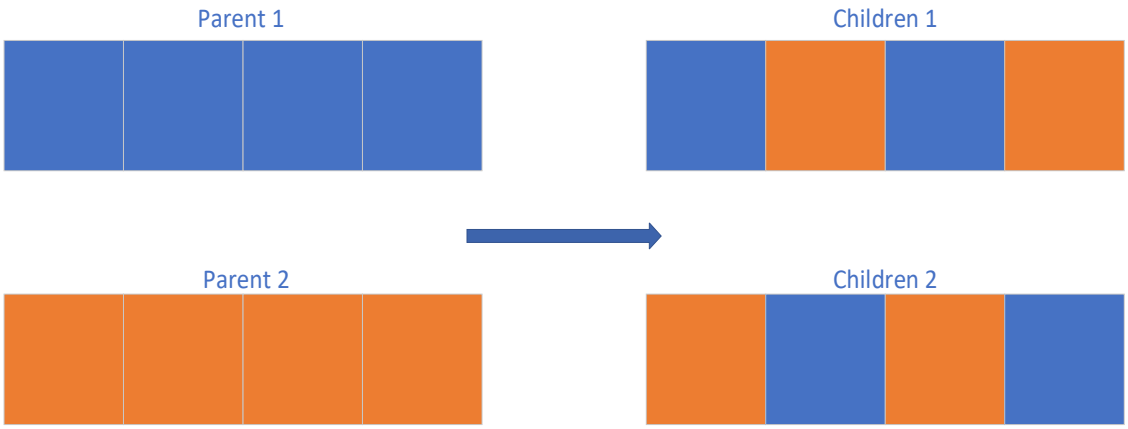


Figure 4.4. Uniform Distributed Crossover

When crossover occurs, it randomly picks two individuals from the parent populations and combines their chromosomes to reproduce two or more children. This process is repeated until the children population has the same size as the parent population, then the children population will become the parent population in the next generation [49, 51, 74-76].

The mutation operator typically occurs directly after the crossover operation. The main reason of having the mutation operator, is because by depending on the crossover operator alone, the GA may only generate children that are very similar to its parents. This may cause the new generation to have a low diversity and may lead to the algorithm being stuck at a local optimum. Therefore, the mutation operator solves the problem by introducing randomness to the children population. This is achieved by randomly changing one or more gene in the children chromosomes. How many genes will be mutated is governed by the GA's mutation rate [49, 51, 74-76].

4.2 Particle-Swarm Optimisation

The term Swarm Intelligent (SI) was first introduced to describe the collective behaviour of decentralised and self-organised natural or artificial systems [77], with the current adaptation more commonly referred to as a class of metaheuristics. A typical SI system consists of a population of agents interacting locally with another and its environment [60, 77]. Each individual agent shows no intelligence but follow simple rules without a centralised control structure governing its behaviour. Agents interact with each other locally with a certain degree of randomness, and although unknown to individual agents an intelligent global behaviour emerges from such local interactions [77].

Particle Swarm Optimisation (PSO) is a type of SI algorithm, which is a population-based metaheuristic optimisation technique developed by Kennedy and Everhart [78]. They got the inspiration from social behaviour of social animals such as bird flocks and fish schools, which shared many similarities with Evolutionary Computation algorithms such as GA. PSO works by first spawning an initial population of random solutions known as particles and searches for optima by updating the properties of individual particles in each generation. Unlike GA, particles in PSO do not converge to a global optimum using evolution operators such as crossover and mutation. Rather, at each generation, the trajectory of each individual particle in the solution space is updated by using a set of two simple equations. The equations determine the particle's velocity and position, which are heavily influenced by the particle's own experience and the current best particles [77]. The velocity and position calculation equations are provided in equation (4.2) and equation (4.3) respectively [60, 77].

$$v_i(t+1) = wv_i(t) + c_{local}r_{local}[\hat{x}_i(t) - x_i(t)] + c_{global}r_{global}[g(t) - x_i(t)] \tag{4.2}$$

where,

c_{local} is the local acceleration coefficient

c_{global} is the global acceleration coefficient

$v_i(t+1)$ is the update velocity

$v_i(t)$ is the velocity of particle *i* at time *t*

$\hat{x}_i(t)$ is the particle *i*'s individual best solution at time *t*

$x_i(t)$ is the particle *i*'s solution at time *t*

$g(t)$ is the swarm's best solution at time *t*

r_{local} is a random value for the cognitive component and r_{global} is a random value for the social component, both of which have the range between 0 and 1. These value are used to introduce stochastic influence on the velocity update.

w is the inertia coefficient, which typically set between 0.8 and 1.2. Lowering the inertia, will increase the convergence rate, while increasing the inertia enables the algorithm to explore the search space. For this research, the inertia is set to 0.9.

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4.3)$$

where,

$x_i(t+1)$ is the updated solution of the particle *i* at time *t+1*

$x_i(t)$ is the current solution of the particle *i* at time *t*

$v_i(t+1)$ is the updated speed of particle *i* at time *t*

The part $c_{local}r_{local}[\hat{x}_i(t) - x_i(t)]$ is collectively known as the cognitive component, which guides the solution spaces towards finding the best solution for each particle individually. The part $c_{global}r_{global}[g(t) - x_i(t)]$ is commonly known as the social component, which guides the solutions spaces of every particle towards a global best solution. Hence these parameters settings have:

- A higher social component than cognitive component, i.e. by setting the acceleration coefficient of 1 local acceleration and 2 global acceleration, at each iteration, the swarm's collective known optimum has a stronger influence towards the convergence of a global optimum of the solution space
- A higher cognitive component than social component, i.e. by setting the acceleration coefficient of 2 local acceleration and 1 global acceleration, at each iteration, individual particle's optimum has a stronger influence towards the convergence of a global optimum of the solution space
- An equal social and cognitive component, i.e. by setting the acceleration coefficient of 2 local acceleration and 2 global acceleration, at each iteration, both individual particle's optimum and the swarm's collective known optimum has the same influence towards the convergence of a global optimum of the solution space

According to Sun et al. [77], PSO is easier to implement, more effective in performing difficult optimisation tasks, and is able to generate better results in a faster and cheaper way compared to several other algorithms with fewer parameters to adjust. Thus in recent years PSO has been frequently used as the favouriate optimisation algorithm, despite that it is well-known that PSO may have the tendency of converging to a local optimum, instead of global optima if the problem space is not well-understood or the algorithm has not been designed correctly [58-60, 77].

4.3 Artificial Neural Network

This research focusses on using a 3-layer (1 input, 1 hidden, 1 output) feedforward neural network with no bias [79, 80] as one of the base controllers to be converted into a Fault-Tolerant Controller, as shown in Figure 4.5. It consists of 2 neurons in the input layer, 8 neurons in the hidden layer, and 8 neurons in the output layer. Each output neuron in the output layer, are then translated to a corresponding output angle each joint needs to rotate. For the ANN used in this research, the inputs are the desired X and Y coordinates which the end-effector wants to reach, and the output is the output angles for each joint to rotate to. The outputs from the ANN are then used to calculate its fitness. The activation function used for the hidden layer and output layer is a linear activation function.

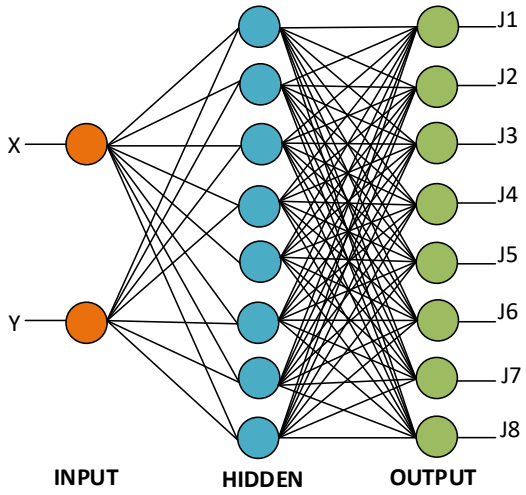


Figure 4.5. The 3-Layer Feed forward ANN for this research

ANN are multi-layered networks with multiple-input-single-output structure that has its roots inspired by neuroscience and the human brain. Since its conception it has become highly popular and has been extensively researched and developed in automated problem solving, output prediction, pattern recognition and classification [1; Chapter 1, 79, 81, 82]. The neural networks found in living organisms consist of a network of interconnected neurons, each neuron receives signals from its dendrites and sends output signals via its axon synapse. When electrical signals entering the dendrites reach a certain threshold, the neuron would fire and transmit electrical signals to the next neuron via the axon. Biological neural networks can be easily translated into software, with most of the ANN developed only differing in the number of layers and type of activation function used. Each artificial neuron receives its input value as a numerical

value, which is multiplied by a weighting factor. When the sum of the input signals exceeds the firing threshold, the output value of the neuron will change and will transfer to the next neuron via a transfer function. The output value can be any convenient mathematical function, and the neuron can have a different decay rate for its accumulated input signal. The structure of the network and the neuron transfer functions are predetermined during the design phase, while the interconnection weightings are adjusted during the training phase [79].

Each layer of an ANN consists of one or more processing elements known as neurons. Each neuron has weighted inputs known as synapses, an activation function that defines the output given an input, a bias function and an output. A weighted input is the product of an input signal and its associated weight. The sum of all the weight inputs produces the activation signal of the neuron in a process known as propagation. A mathematical formula of how the activation signal is calculated is given in equation (4.4).

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j} \tag{4.4}$$

Equation (4.4) is the propagation value of a neuron where, $p_j(t)$ is the input to the neuron j , which is the summation of the products of the outputs $o_i(t)$ and its weights, plus its associated bias value of predecessor neurons connected to neuron j.

The value from $p_j(t)$ in equation (4.4) is then passed into its associated activation function to calculate an output value from the neuron j, which is connected to the neurons in the next layer as inputs [11, 41, 42, 79, 80]. Some of the most commonly used activation functions are linear, step, sigmoid, tanh and rectified linear unit functions. Most of these activation functions have their own subcategories and variation, but comprehensive investigation and discussion of activation functions used for ANN is not within the scope of this research, thus it will not be further examined. A graphical representation of the propagation of a typical neuron in ANN is shown in Figure 4.6.

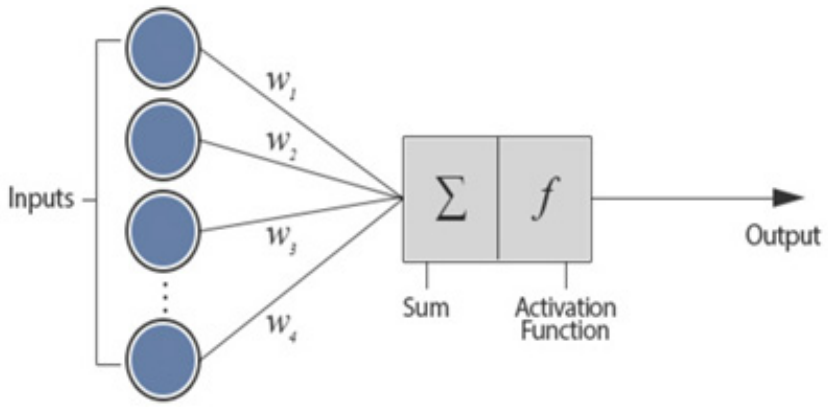


Figure 4.6. Graphical representation of a typical ANN neuron

Due to the highly-connected nature of the neural network and the innate interactions characteristics of neurons between different layers, it can be seen as a non-linear, multi-layered, parallel regression technique that can be used for machine learning, pattern recognition, signal processing, prediction, clustering of data and many more, after it has been properly trained [79]. Training is the process of optimising the weights of the ANN, such that the performance error is minimised, and the network reaches a specified level of accuracy. There are many methods used to train the weights. One of the earliest-developed and most-common training method used to determine the error contribution of each neuron is called known as backpropagation, which calculates the gradient of the loss function [11, 41, 42, 79, 80]. Backpropagation utilises output from the neurons in the output layer and feeds it back to the network as part of the new inputs. Thus, at each iteration the weights are updated, and overall accuracy gets improved over time. There are many methods developed for backpropagation to use in training the weights, for example training via steepest descent, quasi-Newton, conjugate gradient described in books by Mitchell [11], Negnevitsky [44] and Shanmuganathan and Samarasinghe [79]; and a stochastic gradient descent method described by Mohammed and Kostanic [84].

Training methods used for backpropagation can be easy to understand but complex to implement. For example, the stochastic gradient descent method can be summarised into equation (4.5) [11, 41, 42, 79, 80, 84]:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\delta C}{\delta w_{ij}} + \xi(t) \tag{4.5}$$

where, η is the learning rate, C is the loss function and $\xi(t)$ is a stochastic term.

The lost function *C* depends on multiple factors, such as the learning rate and activation function used for the ANN. Therefore, backpropagation requires a significant amount of data, lengthy training period and expert knowledge on both the problem space and the solution space in order to achieve a high accuracy rate. Hence, this research proposed using GA to evolve the weights of the ANN instead of using backpropagation to train the weights. Further description of the proposed method will be provided later in this section.

Since its original development, many different types of activation functions were developed and implemented for various ANN application, with the activation functions typically classified into the following categories [11, 41, 42, 79, 80, 84]:

- Unit Step
- Linear
- Sigmoid
- Hyperbolic Sigmoid
- Sinusoid
- Exponential

ANN as traditionally used to predict future events. However, due to its popularity, it has widely been used across various topics to test its capabilities. ANN was used in [85] to generate solutions for path navigation and maze solving in a robotic system. Faelden et al. [86] showed that by using ANN's they can balance a Quadrotor-Unmanned Aerial Vehicles (QUAV) while it is in operation. Their results, along with findings in Subramanyam et al. [87], suggested that ANN is a highly flexible and adaptable controller, which is likely to work well with many available machine learning techniques to enable allow the controller to achieve autonomous FTC capability. Lang et al. [88] demonstrated that an ANN can be used to perform object recognition of words, in addition Knerr et al. [89] use a single layer ANN for handwriting recognition. Yadav et al. [90] apply ANN to solve various differential equations. Im and Nguyen [81] used ANN to resolve the berthing schedule of ships issue, which is a real-life version of the traveling salesmen problem.

In summary, after training an ANN can learn identify relationships that may exist between the inputs and outputs [79, 80]. Since its origin it has increasingly been used

across a variety of application areas where imprecise data or there are non-linear relationships which are difficult to quantify by using traditional analytical methods. Many learning algorithms are developed and used to train an ANN, which can generally by classified into the following categories: (i) supervised learning, (ii) unsupervised learning, (iii) reinforcement learning and (iv) semi-supervised learning [11, 41, 44, 79]. Many researchers were intrigued to utilise ANN on machines to solve a wide range of problem. A literature review on some of these researches was provided in previous chapters. Over the years, many types of ANN have been developed for various applications and needs. Feedforward, feedback, deep neural network, convolutional, recurrent, dynamic, long/short term memory, auto encoder and residual are several highly popular categories of Neural Network developed, with some of them have its own sub-categories [1, 5, 11, 41, 42, 44, 79, 80].

4.4 Lookup Table

The type of Lookup Table (LUT) used for this research is an 1×8 index mapping, where each element stores the angle that a joint need to rotate to. LUT is a well-known technique, which was used even before the invention of computers to help speed-up hand calculations of complex functions. The LUT is often used in low processing-power devices to help the device when floating-point calculations and trigonometrical functions are required [91, 92]. The LUT can be applied to an evolutionary controller by connecting the sensory inputs to the table axes and the parameters within the table to the actuators [57, 93]. The disadvantages of the LUT include its quantization of the input-outputs and the difficulties in scalability. But when it is adapted into evolutionary computation, the LUT parameters and size can be encoded into a chromosome and modified by the genetic algorithm.

In programming, LUT can be an array or matrix that replaces runtime computation with an array-indexing operation. By using LUT the system can save a significant amount of processing time and utilise the extra resources on other essential operations and /or speed up convergence rate to the desired results.

Examples of LUT used as evolvable controllers include, Currie et al. [94] who evolved the walking gait of a hexapod robot using a two-dimensional LUT with nine distinct leg positions and eighteen motor angles. The utilisation of a LUT based controller can quickly converge to a workable solution when a fault occurs in the system, especially if the controller uses a multi-dimensional LUT that links the state of the system obtained

from its input sensors to the required output for the system to function correctly [50, 91, 92].

4.5 Design and Architecture for ANN Based Fault-Tolerant Controller

This research uses GA or PSO to evolve the weightings of the ANN. The population consists of 100 individuals, known as particles for PSO and chromosomes for GA. Each particle or chromosome has the length of 80, and each element represents a weighting of the ANN which are randomly initialised initially.

The activation function chosen for this ANN-based Fault-Tolerant Controller are:

- linear function for the input layer.
- linear function for the hidden layer.

The fitness function chosen for this controller is the Euclidean Error function, given in equation (4.6):

$$fitness = \sqrt{(x_{best} - x_{current})^2 + (y_{best} - y_{current})^2} \tag{4.6}$$

where,

x_{best} is the x coordinate that produce the swarm's best known fitness output.

y_{best} is the y coordinate that produce the swarm's best known fitness output.

$x_{current}$ is the x coordinate for individuals particle best known fitness output.

$y_{current}$ is the y coordinate for individuals particle best known fitness output.

The fitness function of each individual particle/ chromosome is calculated by dividing the calculation of the forward kinematics for the robotic arm and updating of the ANN's weightings into 10 steps, with each step allowing each individual joint to revolve no more than $-9° \sim 9°$ and no more than $-90° \sim 90°$ in total. The comparison and update of the swarm's collective best fitness is made at the final step, while the comparison and update of the individual's local fitness are made independently at each step. The termination criteria for this algorithm is either the controller had found a global fitness that is equal or smaller than the desire fitness function inputted by the user

and/ or iterations of the algorithm had exceeded 100,000 iterations. The termination criteria is considered to be met, if the a fitness function is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1mm apart.

4.5.1 Design and Architecture of PSO – ANN Fault-Tolerant Controller

Figure 4.7 shows the design and architecture of PSO-ANN as a Fault-Tolerant controller. The particles population for the PSO were initially initialised at random, then it is passed into the ANN to calculate the fitness for each individual particle and the swarm's collective best fitness. Each individual fitness is then used to compare against the swarm's collective best fitness. If an individual particle's fitness is better than the swarm's collective fitness, that particle becomes the latest swarm's collective best solution and its fitness becomes the swarm's collective best fitness. If the swarm's collective best fitness has met the termination criteria or the computation cycle has reached 100,000 iterations, the solution that is the current swarm's collective best fitness solution becomes the solution for the Fault-Tolerant Controller uses; otherwise the particle's velocity and position are updated using its associated equation discussed in previous chapter. The computation cycle repeats itself until either the desired fitness is met, or 100,000 computation cycles are reached. The termination criteria is considered to be met, if the fitness function of the best particle is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1mm apart. At each iteration the swarm's collective best fitness is plotted, when the computation cycles complete the numerical results are exported to a CSV file and graphical results of the fitness function are saved as a PNG[3] file.

[3] PNG stands for Portable Network Graphics

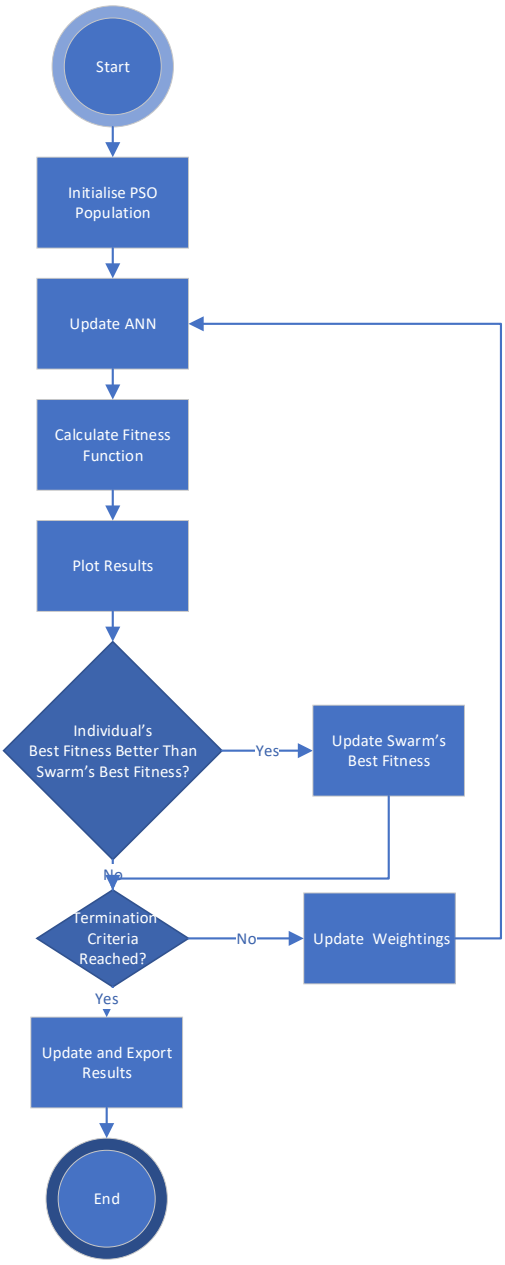


Figure 4.7. Architecture of PSO-ANN Fault-Tolerant Controller

4.5.2 Design and Architecture of GA – ANN Fault-Tolerant Controller

Figure 4.8 shows the design and architecture of GA-ANN as Fault-Tolerant controller. The chromosome population for the GA was initially initialised at random, then it is passed into the ANN to calculate the fitness of each chromosome, the average fitness for the entire population and best global fitness. Each chromosome fitness is then used to compare against the best fitness, if a chromosome's fitness is better than the best fitness, that chromosome becomes the latest best solution and its fitness becomes the best fitness. If the best fitness has met the termination criteria or the computation cycle has reached 100,000 iterations, the best fitness solution becomes the solution for the

Fault-Tolerant Controller; otherwise selection and reproduction will occur to produce a new set of children chromosomes, and the parent chromosome solutions are updated accordingly. The computation cycle repeats itself until either the desired fitness is met, or 100,000 generations is reached. The termination criteria is considered to be met, if the fitness function of a chromosome is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1 mm apart. At each iteration the average and best fitness are plotted, when the computation cycles complete the numerical results are exported to a CSV [4] file and graphical results of the fitness function are saved as a PNG file.

[4] CSV stands for Comma Separated Value

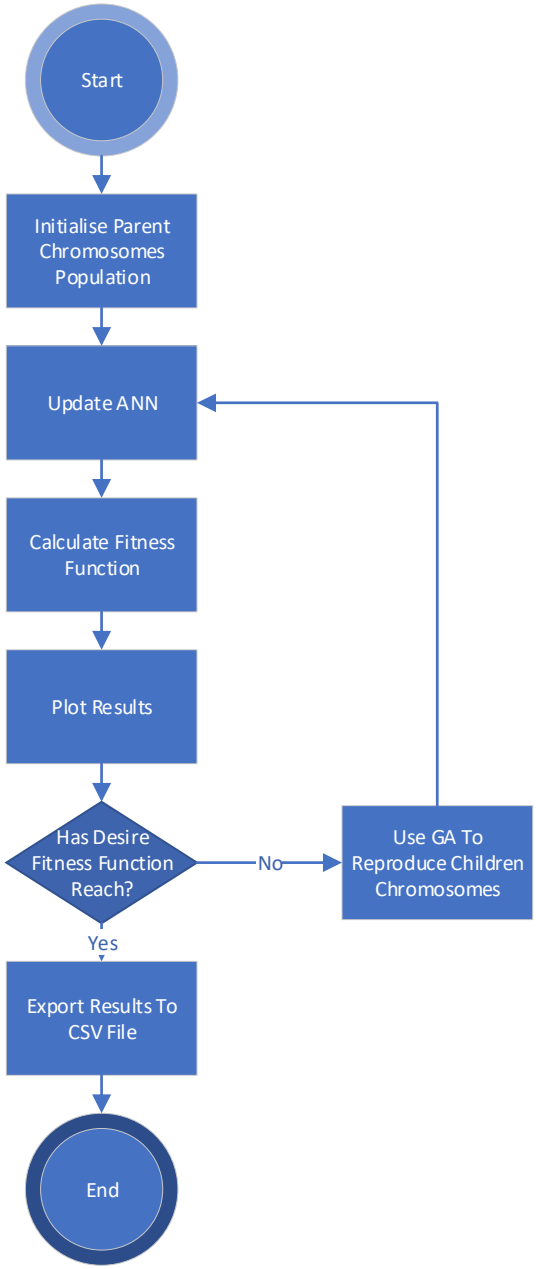


Figure 4.8. Architecture of GA-ANN Fault-Tolerant Controller

4.6 Design and Architecture for LUT based Fault-Tolerant Controller

This research uses GA or PSO to evolve the element in the LUT, where each element represents the angle for each joint needs to revolve. The population consists of 100 swarms of eight individual particles/chromosomes that were first randomly initialised, with each individual particle/chromosome representing a joint angle for the simulated robotic arm.

For the simulation to more accurately represent the physical system, for each output joint angle stored the LUT is bounded by the piecewise function given in equation (4.7):

$$output\ joint\ angle = \begin{cases} -45, & joint\ angle \leq -45^{\circ} \\ joint\ angle, & -45^{\circ} < joint\ angle < 45^{\circ} \\ 45, & joint\ angle \geq 45^{\circ} \end{cases} \quad (4.7)$$

The fitness function chosen for this controller is the Euclidean error function, given in the equation (4.8):

$$fitness = \sqrt{(x_{best} - x_{current})^2 + (y_{best} - y_{current})^2} \quad (4.8)$$

where,

x_{best} is the x coordinate that produce the swarm's best known fitness output.

y_{best} is the y coordinate that produce the swarm's best known fitness output.

$x_{current}$ is the current x coordinate for individual's particle.

$y_{current}$ is the current y coordinate for individual's particle.

The fitness function of each individual particle is calculated by dividing the calculation of the forward kinematics for the robotic arm and updating of the output angles into 10 steps, with each step allowing each individual joint to revolve no more than $-9° \sim 9°$ and no more than $-90° \sim 90°$ in total. The termination criterion for this algorithm is either the controller had found a global fitness that is equal or smaller than the desire fitness function inputted by the user and/ or iterations of the algorithm had exceed 100,000 iteration. The termination criteria is considered to be met, if a fitness function is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1mm apart.

4.6.1 Design and Architecture of PSO – LUT Fault-Tolerant Controller

Figure 4.9 shows the design and architecture of PSO-LUT as Fault-Tolerant controller. The particles population for the PSO were initially initialised at random, then it is used to update elements in the LUT and calculate the fitness for each individual particle and the swarm's collective best fitness. Each individual fitness is then used to compare against the swarm's collective best fitness, if an individual particle's fitness is better than the swarm's collective fitness, that particle becomes the latest swarm's collective best solution and its fitness becomes the swarm's collective best fitness. If the swarm's collective best fitness has met the termination criteria or the computation cycle has reached 100,000 iteration, the solution that is the current swarm's collective best fitness solution becomes the solution for the Fault-Tolerant Controller uses. Otherwise, the particle's velocity and position are updated using its associated equation discussed in previous chapter. The computation cycle repeats itself until either the desired fitness is met, or 100,000 computation cycle is reached. The termination criteria is considered to be met, if the fitness function of the best particle is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1mm apart. At each iteration the swarm's collective best fitness are plotted, when the computation cycles complete the numerical results are exported to an CSV file and graphical results are saved as a PNG file.

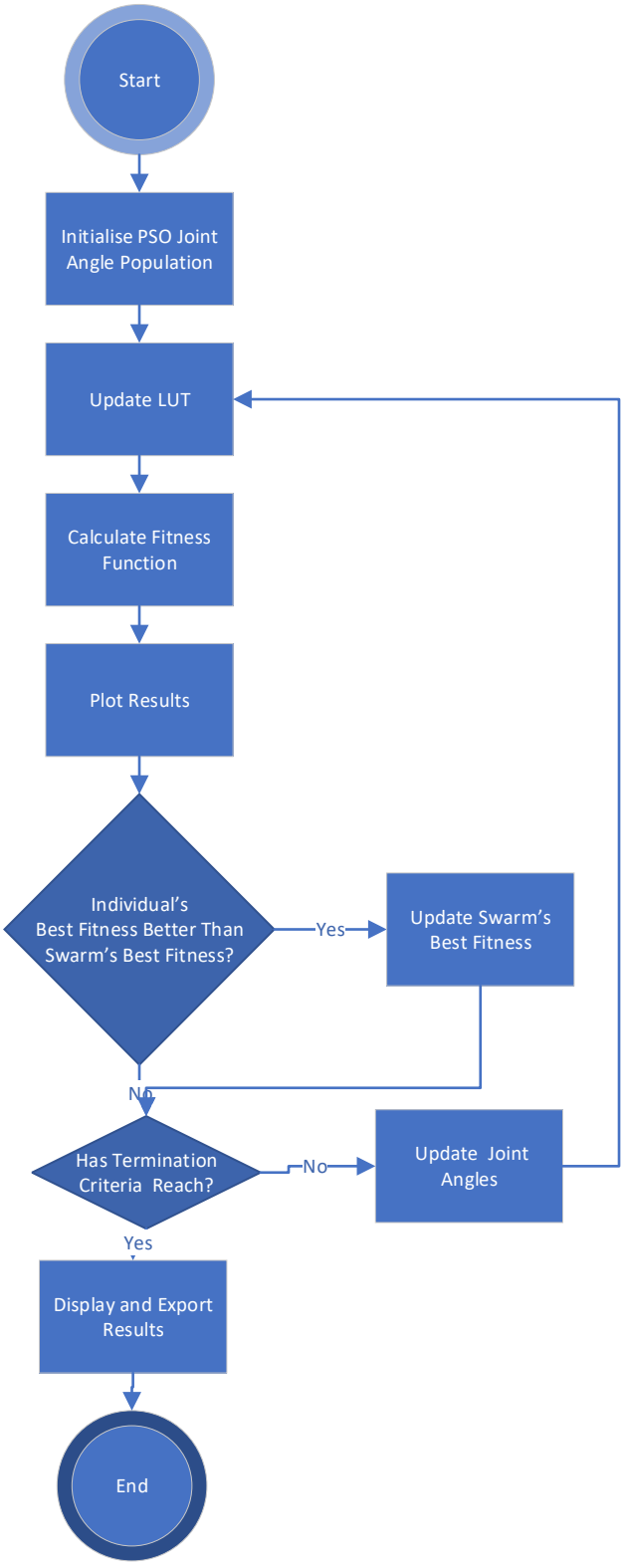


Figure 4.9. Architecture of PSO-LUT Fault-Tolerant Controller

4.6.2 Design and Architecture of GA – LUT Fault-Tolerant Controller

Figure 4.10 shows the design and architecture of GA-LUT as Fault-Tolerant controller. The chromosome population for the GA was initially initialised at random, then it is passed into the LUT to calculate the fitness of each chromosome, the average fitness for the entire population and best global fitness. Each chromosome fitness is then used to compare against the best global fitness. If a chromosome's fitness is better than the global best fitness, then that chromosome becomes the latest best global solution and its fitness becomes the best global fitness. If the global best fitness has met the termination criteria or the computation cycle have reach 100,000 iteration, the global best fitness solution becomes the solution for the Fault-Tolerant Controller uses. Otherwise, selection and reproduction will occur to produce a new set of children chromosomes, and the parent chromosome solutions are updated accordingly. The computation cycle repeats itself until either the desired fitness is met, or 100,000 computation cycle is reach. The termination criteria is considered to be met, if the fitness function of a chromosome is smaller or equal to the value chosen by the user. For the experiments of this thesis, the termination criteria is considered to be met, if the fitness function scored less than 0.1, i.e. the Euclidean error between the ideal location and the calculated location of the arm of the arm is less than 0.1mm apart. At each iteration the average and global best fitness are plotted, when the computation cycles complete the numerical results are exported to an CSV file and graphical results of the fitness function are saved as a PNG file.

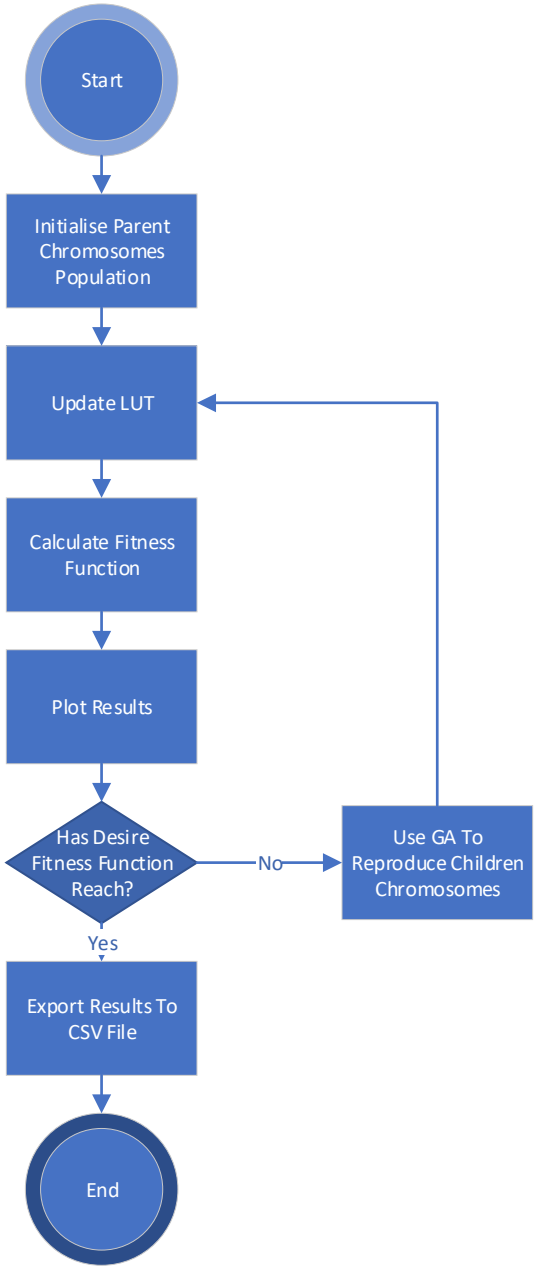


Figure 4.10. Architecture of GA-LUT Fault-Tolerant Controller

In summary, this chapter provides detail descriptions of the optimisation algorithm and controllers used to construct the Fault-Tolerant Controller developed for this research. In addition, this chapter provides the flowchart and detail description of the architecture for the four Fault-Tolerant controllers developed for this research. The next chapter provides a summary and comparisons of results generated by the Fault-Tolerant Controllers.

Chapter 5
Results

This chapter presents the graphical and numerical results for all four Fault-Tolerant Controllers developed for this research, and each controller is evaluated with five different fault combinations, ranging from zero to four faults. The test was repeated 100 times for each fault.

The first half of this chapter will compare and analyse the performance of each controller and the effects that different parameters and fault combinations had on the controller's convergence rate. The second half of this chapter provides a comparison and analysis of the performance between the four controllers. Results are plotted using the box-and-whisker diagram for each feature to show the data in terms of quartiles. The box represents the data that lies between the lower and upper quartile, which is the interquartile range of the data. The horizontal line in the middle of the box represents the median of the data. The whiskers on the plot mark the full range of the data and are drawn on either side of the box, representing the minimum and maximum values.

5.1 Effects of Using Different Parameter Settings on the Controllers

In order to determine whether different parameter settings have a significant effect on the convergence rate of the controllers, each of the four controllers is experimented on with different parameters settings and results of hundred trials are gathered and analysed.

5.1.1 Results for Different Acceleration Coefficient Combinations on PSO

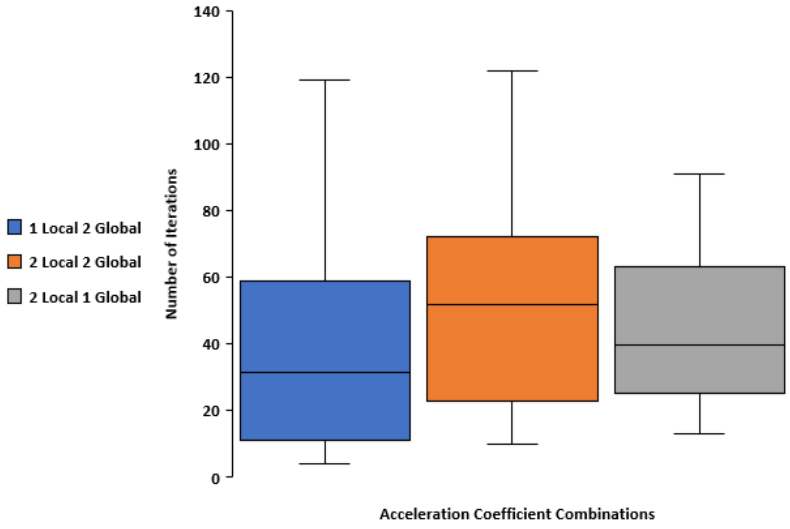
For PSO the different combinations of acceleration coefficient has the following effects:

- Two Local Acceleration and One Global Acceleration, the swarm emphasis more on finding and converging the individual particle's optimum more than converging towards the swarm's collective optimum.
- One Local Acceleration and Two Global Acceleration, the swarm emphasis more on converging towards the swarm's collective optimum more than individual particle's optimum.

- Two Local Acceleration and Two Global Acceleration, the swarm emphasis equally on converging towards the swarm's collective optimum and individual particle's optimum.

The maximum acceleration coefficient value of two is chosen since the reviewed articles [58-61, 87, 95], all suggested that neither the local nor the global acceleration coefficient should not exceed a value of two. The remainder of this section presents the effects that different acceleration coefficient combinations had on the convergence rate on a system with no faults, the acceleration coefficients combinations chosen to be investigated in this research are: (i) 1 Local and 2 Global Acceleration, (ii) 2 Local and 2 Global Acceleration, and (iii) 1 Local and 2 Global Acceleration.

Figure 5.1 shows that for PSO-ANN, a global acceleration coefficient of two and local acceleration coefficient of one produces the fastest median convergence rate.



	1 Local 2 Global	2 Local 2 Global	2 Local 1 Global
Upper Extreme	119	122	91
Upper Quartile	59	72	63
Median	32	52	40
Lower Quartile	11	23	25
Lower Extreme	4	10	13
Mean	50	50	48

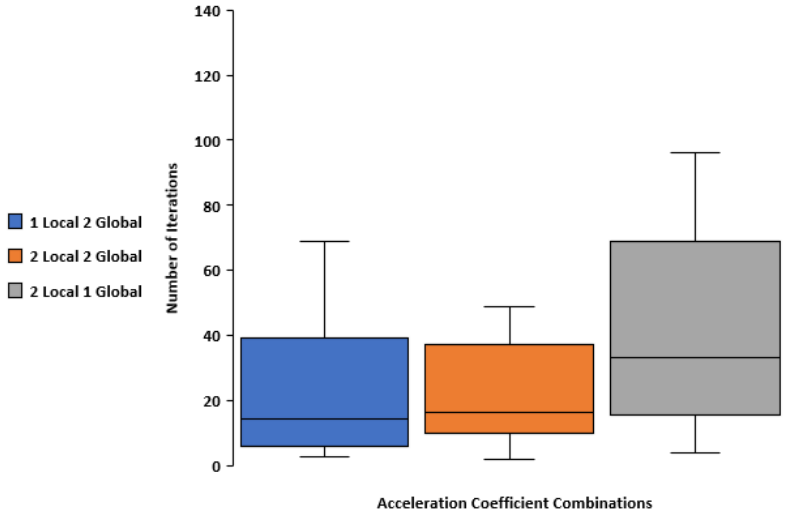
Figure 5.1 Results for Different Acceleration Coefficient Combination for PSO-ANN

For PSO-ANN, the acceleration coefficient combination Two Local and Two Global, and Two Local and One Global have similar mean, lower quartile and lower extreme results; but have worst performance when compared to One Local and Two Global. The acceleration coefficient combination Two Local and Two Global has a poorer

median, upper quartile and upper extreme convergence rate comparing to the other Two acceleration coefficient combination. In addition, results for Two Local and Two Global appears to be left skewed, where the other two acceleration coefficient combinations appear to be right skewed. If the results are left skewed, it meant that the results has a lower mean than the median, on the contrary if the results is right skewed it meant that the results has a lower median than the mean. A left skewness in the results suggests that majority of the time the controller is able to coverage to an acceptable result with iterations around the median, with some outliers that converge to a result significantly faster than others and thus it lowers the mean of the convergence rate. Whereas a right skewness of the results, indicates that majority of the of the time the controller is able to converge to an acceptable results with iterations indicated by the median with some outliers that took significantly longer to converges to a results and thus increase the mean of the convergence rate. However, if the upper and lower extremes were taken out of considerations, results from Two Local and Two Global tends to appear to be more uniformly distributed than the other two, whereas right skewedness within the results is still observable for the other two acceleration coefficient combinations.

Figure 5.2 shows that for PSO-LUT: (i) local acceleration coefficient of one and global acceleration coefficient of two and (ii) local acceleration coefficient of two and global acceleration coefficient of two both have the same mean convergence rate, however local acceleration coefficient of two and global acceleration coefficient of two has a better median convergence rate, thus it is considered to be a better configuration to be use.

While, the means for all three combinations have similar values, the combination that has a stronger emphasis on converging the results towards the swarm's collective best solution has the least variation between upper quartile and lower quartile, while the combination that has a stronger emphasis on converging the results towards the individual's best solution has the largest variation.



	1 Local 2 Global	2 Local 2 Global	2 Local 1 Global
Upper Extreme	69	49	96
Upper Quartile	39	38	69
Median	15	17	34
Lower Quartile	6	10	16
Lower Extreme	3	2	4
Mean	26	29	53

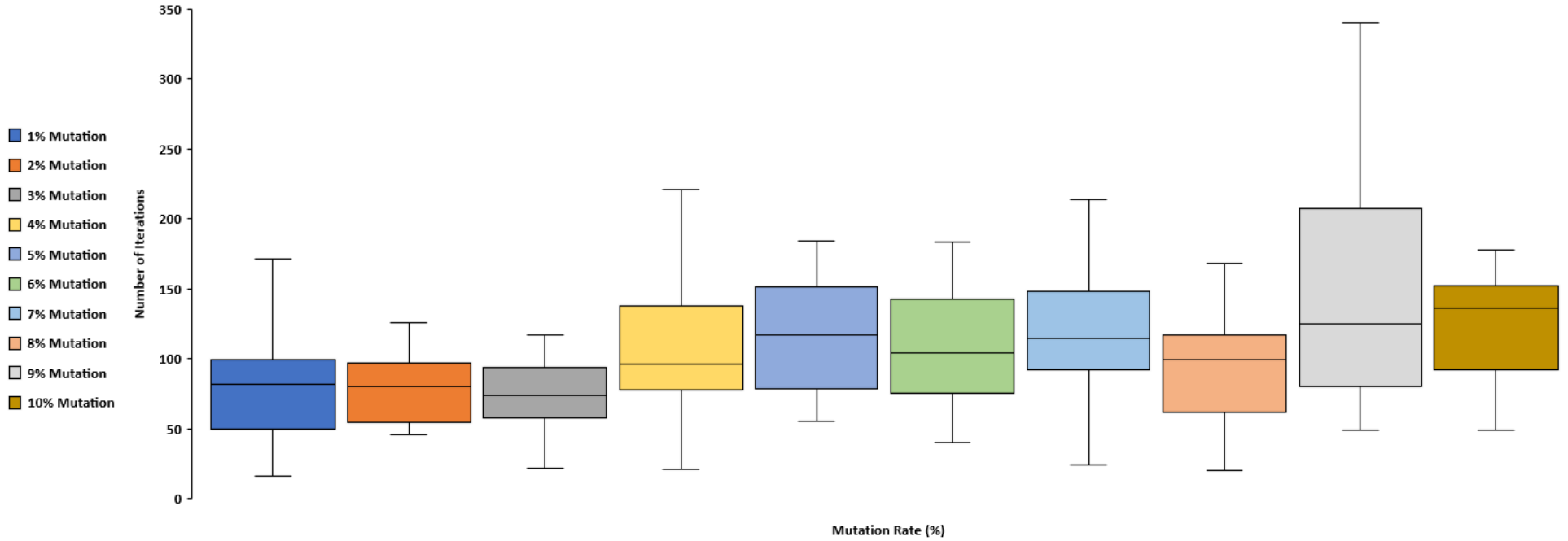
Figure 5.2 Effects on PSO-LUT Using Different Acceleration Coefficient Combination

For PSO-LUT, results have shown that the acceleration coefficient combination One Local and Two Global and, Two Local and Two Global are similar to each other and outperforms the acceleration coefficient combination Two Local and One Global. The acceleration coefficient combination One Local and Two Global have a slightly better lower quartile and median convergence rate than Two Local and Two Global, however it has a worst upper extreme and no difference in upper quartile convergence rate. Despite for PSO-LUT all three-acceleration coefficient combination showed right skewedness in its results, the acceleration coefficient combination Two Local and Two Global has a significant lower variation between upper and lower quartile, and between upper and lower extreme. Thus, it can be considered that have a more predictable convergence rate than the other two acceleration coefficient.

In conclusion, the acceleration coefficient combination One Local and Two Global is considered to have the best performance comparing to the other two acceleration coefficient combination. Therefore, for the remainder of this research it is chosen to be the default acceleration coefficient combinations to be used.

5.1.2 Results for Different Mutation Rates on GA

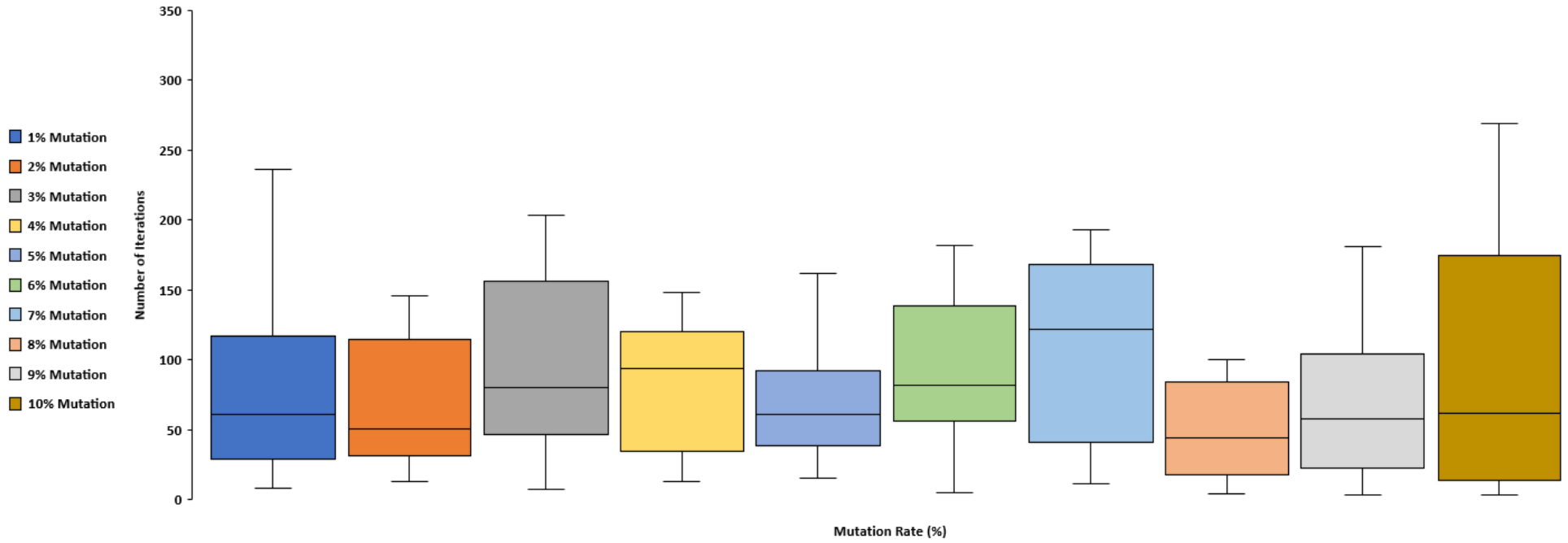
Figure 5.3 shows the results for GA-ANN using different mutation rates on the system with no fault in the system. Although the median convergence rate is similar between various mutation rates, the 3% mutation rate has produced the fastest median convergence rate.



	1% Mutation	2% Mutation	3% Mutation	4% Mutation	5% Mutation	6% Mutation	7% Mutation	8% Mutation	9% Mutation	10% Mutation
Upper Extreme	171	126	117	221	184	183	214	168	340	178
Upper Quartile	100	97	94	138	152	143	148	117	207	153
Median	82	81	74	96	117	104	115	99	125	137
Lower Quartile	50	55	58	78	78	76	92	62	80	92
Lower Extreme	16	46	22	21	55	40	24	20	49	49
Mean	81	82	76	107	117	107	119	100	145	131

Figure 5.3 Effects on GA-ANN Using Different Mutation Rate

Figure 5.4 shows the effects of different mutation rates had on the GA-LUT with zero fault. Similar to the results observed from GA-ANN, the mean and median between the mutation rates are close to each other, except for the median for 7% mutation rate where it is observably higher than the other mutation rates. An obvious difference between the results in GA-LUT and GA-ANN is, for each mutation rates in GA-LUT produces a larger variation between the upper quartile and lower quartile. The overall performance on convergence rate for each mutation rate in GA-LUT is observably better than GA-ANN. Since GA-LUT has lower mean, median, upper quartile and lower quartile values than GA-ANN across all mutation rate. Another difference between GA-ANN and GA-LUT, is that for GA-LUT 5% and 8% mutation rate tends to have the best performance, whereas GA-ANN 2% and 3% mutation rate appears to have the best performance rate.



	1% Mutation	2% Mutation	3% Mutation	4% Mutation	5% Mutation	6% Mutation	7% Mutation	8% Mutation	9% Mutation	10% Mutation
Upper Extreme	236	146	203	148	162	182	193	100	181	269
Upper Quartile	117	115	156	120	92	138	169	85	104	175
Median	61	51	80	94	61	82	122	45	58	62
Lower Quartile	29	32	45	35	39	57	41	18	23	14
Lower Extreme	8	13	7	13	15	5	11	4	3	3
Mean	87	85	98	82	74	113	106	66	65	92

Figure 5.4 Effects on GA-LUT Using Different Mutation Rate

In conclusion, from the experimentation results discussed above for both ANN and LUT, for GA it is ideal to choose a mutation rate between 1% and 5%. For GA-ANN Fault-Tolerant Controller with zero fault using the mutation rate between 1% and 3%, tends to produce the fastest convergence rate. While for GA-LUT Fault-Tolerant Controller the mutation rate 2%, 5% and 8% tends to converges the quickest. When the results of effects of mutation rate had on both ANN and LUT are compared side by side, it is observable that the mutation rates between 1% and 3% has similar median and lower quartile convergence rate. However, for LUT the effects have a larger variation for upper quartile and upper extreme when compared to ANN. In addition, despite 5% mutation rate has a slightly worst median convergence rate for both ANN and LUT, however it also produces a more stable result. That is, it has a more stable and observably lower variations between the results, i.e. lower variation between upper and lower quartile, and between upper and lower extremes. Therefore, for the remainder of this research 5% mutation rate are chosen to be used as the default mutation rate.

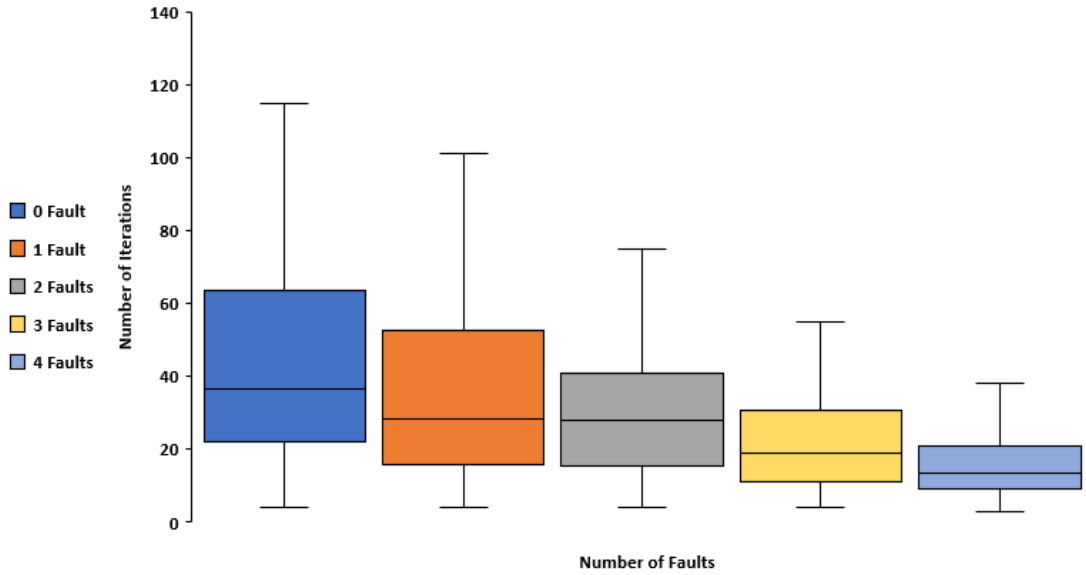
5.2 PSO – ANN Fault-Tolerant Controller Results

Figure 5.5 shows the performance of PSO-ANN as a Fault-Tolerant Controller, when it is tested across all five fault combinations, ranging from no fault to four faults. The mean and median shown in Figure 5.5 suggested that as the number of faults within the system increases, the amount of iterations the controller required to converge to a solution decrease. Moreover, as the number of faults increases, variation (i.e. difference between the upper and lower quartile, and upper and lower extreme) of convergence rate amongst the hundred trials decreases as well.

In summary, the PSO-ANN as a Fault-Tolerant controller has the following characteristic:

- For zero fault, it has slowest convergence rate.
- For one fault, it has a faster convergence rate compared to zero fault, yield a similar convergence rate to two faults, and has slower convergence rate compared to three and four faults.
- For two faults, it is statistically faster convergence rate compared to zero fault, yield a similar on convergence rate to one and three faults and has slower convergence rate compared to four faults.

- For three faults, it has a faster convergence rate compared to zero and one fault and are yield a similar in convergence rate to two and four faults.
- For four faults, it has a faster convergence rate compared to zero to two faults and yield a similar in convergence rate compared to three faults.



	0 Fault	1 Fault	2 Faults	3 Faults	4 Faults
Upper Extreme	115	101	75	55	38
Upper Quartile	64	53	41	31	21
Median	37	29	28	19	14
Lower Quartile	22	16	15	11	9
Lower Extreme	4	4	4	4	3
Mean	48	37	30	22	16

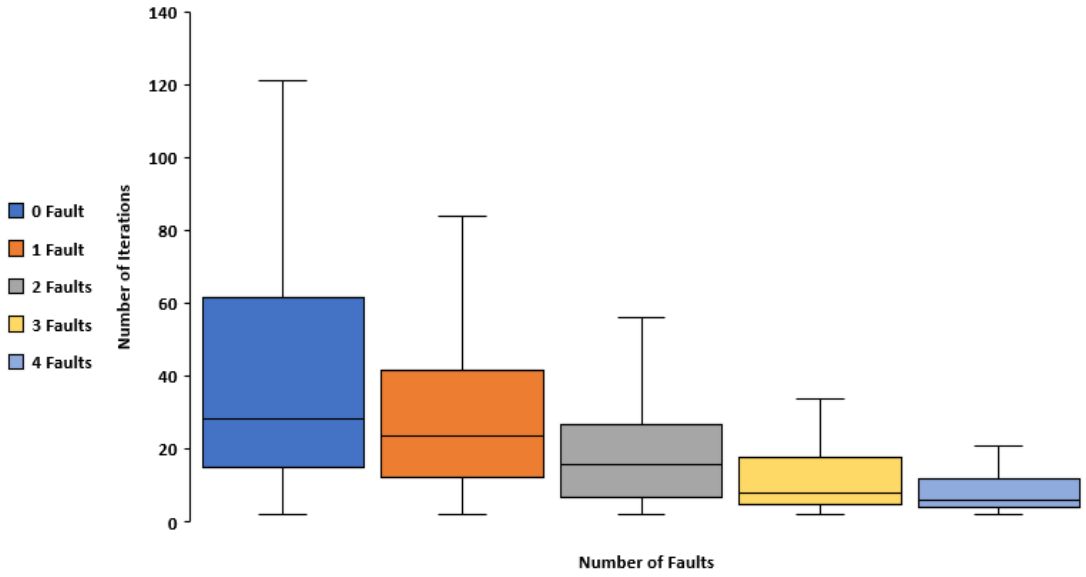
Figure 5.5 Results for PSO-ANN Across All Faults Combination

5.3 PSO – LUT Fault-Tolerant Controller Results

Figure 5.6 shows the performance of PSO-LUT as a Fault-Tolerant Controller, when it is tested across all five fault combinations ranging from no fault to four faults. Results shown in Figure 5.6 resemble results from PSO-ANN, where the mean and median suggested that as the number of faults within the system increases, the amount of iterations the controller required to converge to a solution decreases. As the number of faults increases, variation (i.e. difference between the upper and lower quartile, and upper and lower extreme) of convergence rate amongst the hundred trials decreases as well. However, the overall performance of the PSO-LUT is comparatively better than PSO-ANN as the median, mean, upper extremes and the quartile ranges across all five fault combinations are numerically better.

In summary, the PSO-LUT as a Fault-Tolerant controller has the following characteristics:

- For zero fault, it has slowest convergence rate.
- For one fault, it has statistically faster convergence rate compared to zero fault, but has slower average convergence rate compared to two, three and four faults.
- For two faults, it has statistically faster convergence rate compared to zero and one fault, statistically indifferent on convergence rate compared to three faults, and has slower convergence rate compared to four faults.
- For three faults, it has statistically faster convergence rate compared to zero to two faults and is statistically indifferent in convergence rate four faults.
- For four faults, it has statistically faster convergence rate compared to zero to two faults and is statistically indifferent in convergence rate compared to three faults.



	0 Fault	1 Fault	2 Faults	3 Faults	4 Faults
Upper Extreme	121	84	56	34	21
Upper Quartile	62	42	27	18	12
Median	29	24	16	8	6
Lower Quartile	15	12	7	5	4
Lower Extreme	2	2	2	2	2
Mean	42	31	20	14	9

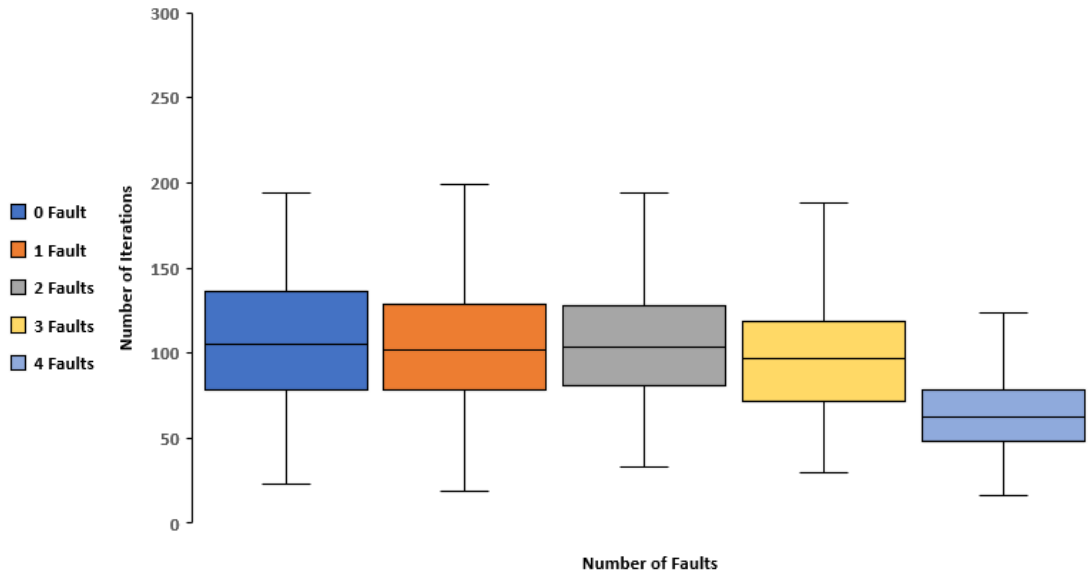
Figure 5.6 Results for PSO-LUT Across All Faults Combinations

5.4 GA – ANN Fault-Tolerant Controller

Figure 5.7 shows the performance of GA-ANN as Fault-Tolerant Controller, when it is tested across all 5 fault combinations ranging from zero fault to four faults. Results from Figure 5.7 show that for 0 to 3 faults the mean, median, extremes and quartile range are highly similar, whereas for 4 faults it is observably better than the other four fault combinations.

In summary, the GA-ANN as a Fault-Tolerant controller has the following characteristics:

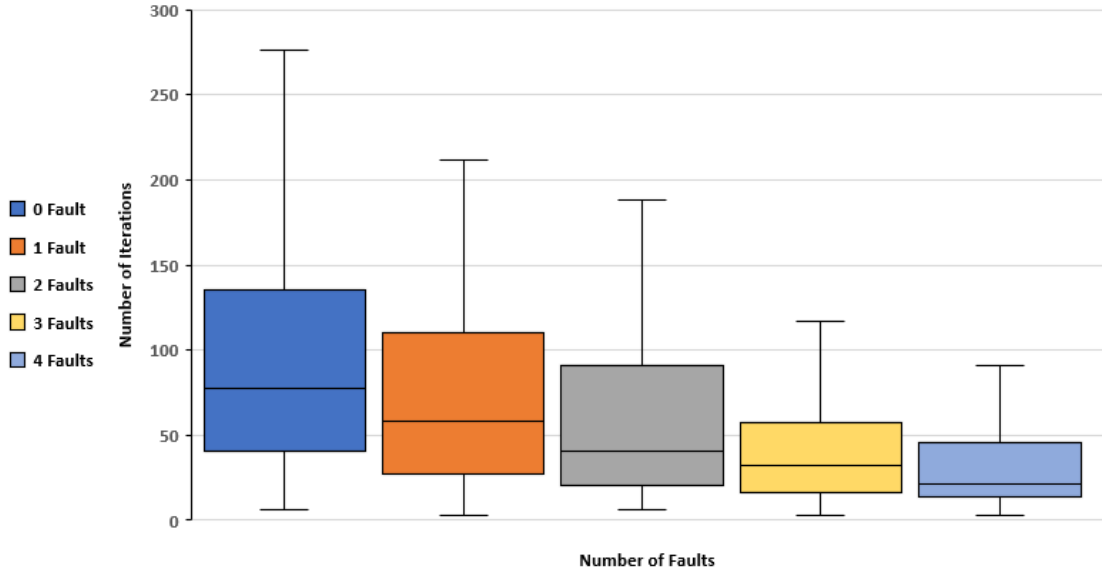
- Convergence rate between zero to three faults are not statistically different on a significant level.
- Convergence rate for four faults are statistically faster on a significant level, compared to the other four fault combinations.



	0 Fault	1 Fault	2 Faults	3 Faults	4 Faults
Upper Extreme	194	199	194	188	124
Upper Quartile	137	129	128	119	79
Median	106	102	104	97	63
Lower Quartile	78	79	81	72	48
Lower Extreme	23	19	33	30	16
Mean	106	104	105	100	66

Figure 5.7 Results for GA-ANN Across All Faults Combinations

5.5 GA – LUT Fault Tolerant-Controllers



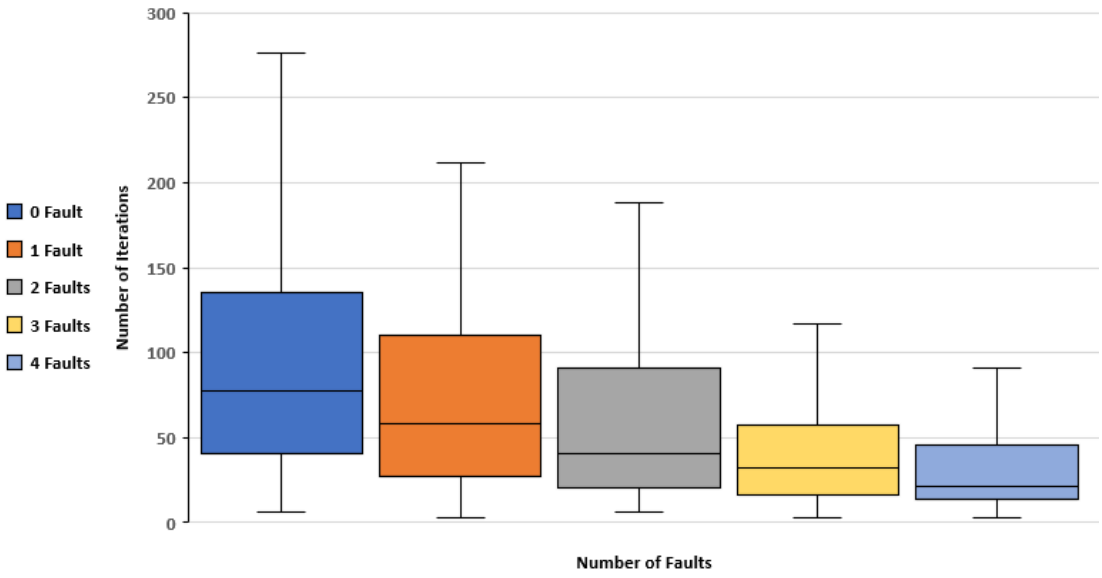
	0 Fault	1 Fault	2 Faults	3 Faults	4 Faults
Upper Extreme	276	212	188	117	91
Upper Quartile	136	111	91	57	46
Median	78	58	41	33	22
Lower Quartile	41	27	20	16	14
Lower Extreme	6	3	6	3	3
Mean	106	85	59	39	37

Figure 5.8 shows the performance of GA-LUT as a Fault-Tolerant Controller, when it is tested across all five fault combinations ranging from no fault to four faults. The results show a trend from that in PSO-ANN and PSO-LUT, whereas the number of faults within the system increases, the iterations the controller required to converge to a solution decrease. As the number of faults increases, the variation (i.e. difference between the upper and lower quartile, and upper and lower extreme) of convergence rate amongst the hundred trials decreases as well. However, the overall performance of the GA-LUT is comparatively worse than PSO-ANN and PSO-LUT as the median, mean, upper extremes and the quartile ranges across all five fault combinations are numerically larger.

In summary, the GA-LUT as a Fault-Tolerant controller has the following characteristics:

- It has the similar trend as PSO-ANN and PSO-LUT, that is as the number of faults increases, convergence rate improves.

- For zero fault, convergence rate is slower compared to two to four faults, while it yield a similar convergence rate is compared to the convergence rate with 1 fault.
- For one fault, convergence rate is slower compared to two to four faults, while it yields a similar convergence rate compared to the convergence rate with 0 fault.

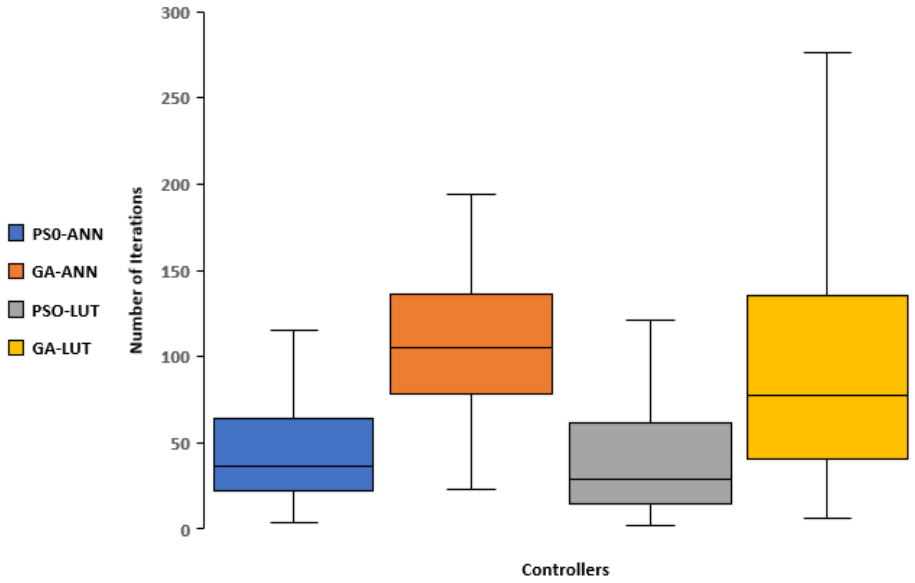


	0 Fault	1 Fault	2 Faults	3 Faults	4 Faults
Upper Extreme	276	212	188	117	91
Upper Quartile	136	111	91	57	46
Median	78	58	41	33	22
Lower Quartile	41	27	20	16	14
Lower Extreme	6	3	6	3	3
Mean	106	85	59	39	37

Figure 5.8 Results for GA-LUT Across All Faults Combination

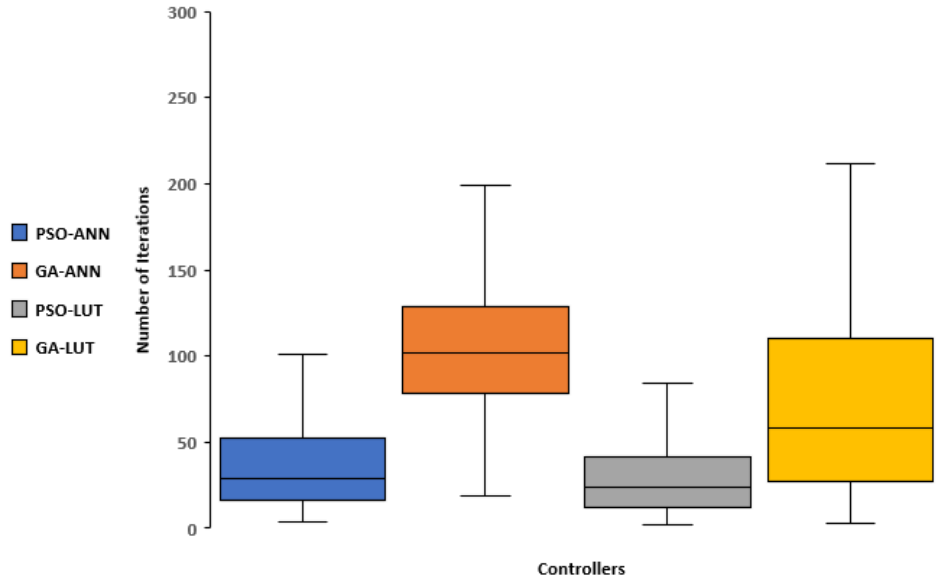
5.6 Comparison of All Four Fault-Tolerant Controllers

Figure 5.9 to Figure 5.13, shows a side-by-side comparison of the convergence rate of the five fault combination used to test the four Fault-Tolerant Controllers developed for this work. The results show that PSO-LUT has the fastest convergence rate for all five fault combinations, followed by PSO-ANN then GA-LUT and finally GA-ANN. In addition, all four controllers appear to have improved convergence rate and reduce the variations between upper and lower quartile and variation between upper and lower extremes as the number of faults in the system increase.



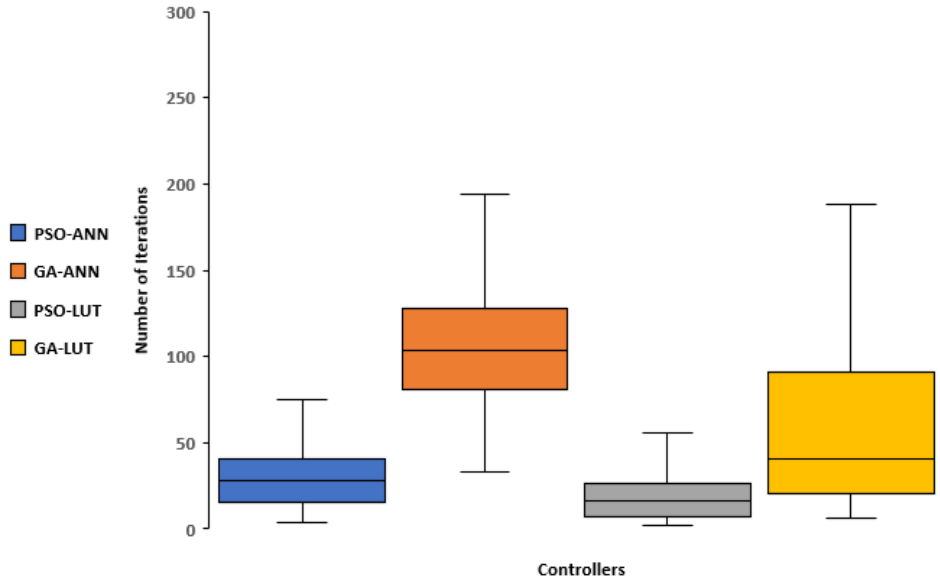
	PS0-ANN	GA-ANN	PSO-LUT	GA-LUT
Upper Extreme	115	194	121	276
Upper Quartile	64	137	62	136
Median	37	106	29	78
Lower Quartile	22	78	15	41
Lower Extreme	4	23	2	6
Mean	48	106	42	106

Figure 5.9. Side-by-side Comparison of Controllers on 0 Fault



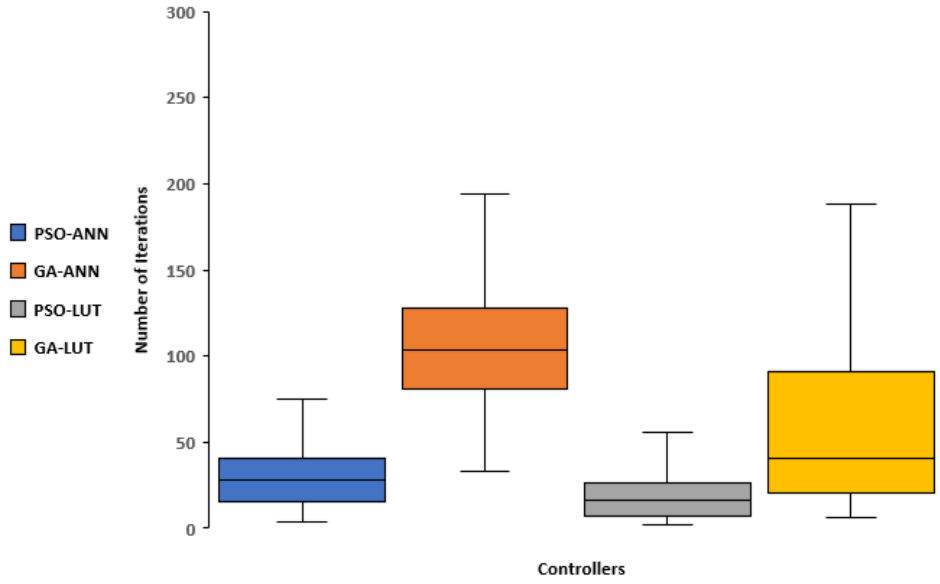
	PSO-ANN	GA-ANN	PSO-LUT	GA-LUT
Upper Extreme	101	199	84	212
Upper Quartile	53	129	42	111
Median	29	102	24	58
Lower Quartile	16	79	12	27
Lower Extreme	4	19	2	3
Mean	37	104	31	85

Figure 5.10. Side-by-side Comparison of Controllers on 1 Fault



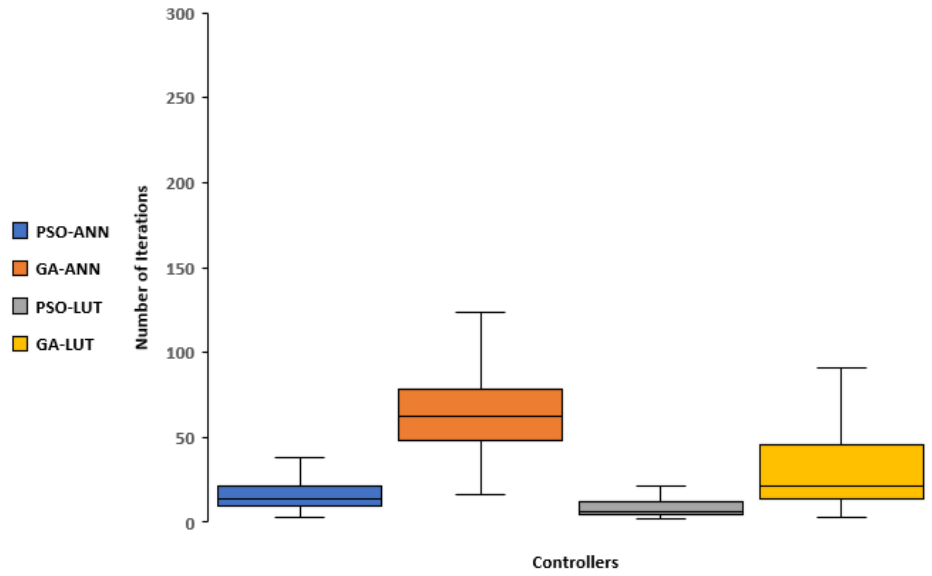
	PSO-ANN	GA-ANN	PSO-LUT	GA-LUT
Upper Extreme	75	194	56	188
Upper Quartile	41	128	27	91
Median	28	104	16	41
Lower Quartile	15	81	7	20
Lower Extreme	4	33	2	6
Mean	30	105	20	59

Figure 5.11. Side-by-side Comparison of Controllers on 2 Faults



	PSO-ANN	GA-ANN	PSO-LUT	GA-LUT
Upper Extreme	75	194	56	188
Upper Quartile	41	128	27	91
Median	28	104	16	41
Lower Quartile	15	81	7	20
Lower Extreme	4	33	2	6
Mean	30	105	20	59

Figure 5.12. Side-by-side Comparison of Controllers on 3 Faults



	PSO-ANN	GA-ANN	PSO-LUT	GA-LUT
Upper Extreme	38	124	21	91
Upper Quartile	21	79	12	46
Median	14	63	6	22
Lower Quartile	9	48	4	14
Lower Extreme	3	16	2	3
Mean	16	66	9	37

Figure 5.13. Side-by-side Comparison of Controllers on 4 Faults

Chapter 6
Conclusion and Discussion

The overall aim of the research reported in this thesis was to investigate, design, develop and compare four Fault-Tolerant Controllers for a multi-joint planar robotic arm as FTC, by combining well-studied optimisation algorithms and robotic controllers. Analysis of convergence rate of multiple trials for different fault combinations will assist in determining the effectiveness of the developed controllers.

The literature reviewed categorised faults into five major categories, which are: Internal, Sensory, Operational, Design, and Combinational. In addition, a fault could occur: Abruptly, Incipiently, Permanently, Transiently, Intermittently, and Hiddenly. Thus, it is impossible for an engineer to anticipate every possible fault and take preventative steps when designing the system, as a result it gave rise to Fault-Tolerant Controller. A Fault-Tolerant Controller allows the system to adapt the fault and enables it to fail gracefully. instead of immediate system failure. The literature reviewed in chapter 2 describes the architecture of Fault-Tolerant Controller typically consisting of two parts, which are (i) Fault Diagnosis and (ii) Fault-Tolerant Control and can be either passive or active.

This research primarily focuses on investigating and development of active FTC, after a thorough review of existing literature, four Fault-Tolerant Controllers were designed and built for this research, which are: (1) PSO-ANN, (2) PSO-LUT, (3) GA-ANN, and (4) GA-LUT.

This research developed and compared the effectiveness of four Fault-Tolerant Controllers, which were designed and implemented only using active FTC, by combining PSO and GA with ANN and LUT to control a simulated robotic arm system. The robotic arm is tasked to move from a set starting point to a location set by the user within its workspace, with or without occurrence of fault in the system. The controllers observed to find how fast it can adopt itself to changes in the number of faults (ranging from no fault to four faults) presented in the system.

Results in Chapter 5 have shown that the Fault-Tolerant Controller using PSO-LUT has the fastest overall convergence rate across all the fault combinations tested, followed by PSO-ANN, GA-ANN and finally GA-LUT. In addition to having the fastest overall

convergence rate, PSO-LUT also has the lowest variation between the upper and lower quartiles from the results of each trial it was experimented on. For the controller GA-ANN the overall convergence rate remains consistent between zero to three faults, and it performs better at a statistically significant level for four faults. However, for the controllers PSO-ANN, PSO-LUT and GA-LUT a common feature was observed, that is with the increasing number of faults presented in the system, up to four faults, there is an observable increase of convergence rate. The reason for this is because, as the number of faults increases the number of solutions and ideal solution in the solution space decreases. A solution in the solution space, is a location where the robotic arm can reach within its workspace, whereas ideal solutions in the solution space, are solutions that can drive the robotic arm system to the desired destination. However, the number of solutions in the solution space decreases faster than the number of ideal solutions in the solution space. Therefore, the controller is more likely to find a solution it needs to drive the robotic arm to its desired destination as the number of faults increase. Yet, the convergence rate only increases up to four faults in the system and will decrease or no solution could be found from five faults onwards. This is because from five faults onwards it becomes physically impossible for the robotic arm system to reach the desired destination.

Experiment with no fault in the system, suggested that by using different parameters for PSO (different combinations of local and global acceleration coefficient) and GA (different mutation rate) appears to affect the performance of the Fault-Tolerant Controller.

For PSO based Fault-Tolerant Controllers: PSO-ANN and PSO-LUT, three combinations of acceleration coefficient combination were tested, which are: a) One local acceleration and Two global acceleration, b) Two local acceleration and Two global acceleration, and c) Two local acceleration and One global acceleration. Results from both PSO-ANN and PSO-LUT had shown that the acceleration coefficient combination One local and Two global has the fastest median convergence rate, while Two local and Two global has the slowest median convergence rate. However, the mean convergence rates for all three acceleration coefficient combinations are very similar. In addition, the acceleration coefficient combination One local and Two global, has the smallest variation between the upper and lower quartile, the lowest upper extreme and the highest lower extreme. This suggests that different acceleration combination has effects on the convergence rate of the PSO based Fault-Tolerant

Controller. In addition, it suggest future work on finding the combination of acceleration coefficients that optimised the convergence rate for the controllers for various faults situations.

For GA based Fault-Tolerant Controllers (a)GA-ANN and (b)GA-LUT, the results for no fault identifies mutation rate had an effect on the convergence rate of the Fault-Tolerant Controller. Results from GA-ANN shows that the 1% to 3% mutation rate appears to have similar mean, median and upper quartile values that are lower than the other mutation rates. In addition, these three mutation rates appear to yield a better convergence rate, than other mutation rate. However, for GA-LUT results have shown that 5% and 8% mutation rate yield the fastest convergence rate and has the least variation between upper and lower quartile. In addition, 8% mutation rate has observably smaller upper extreme value than other mutation rate. This suggest future work could be carried out to investigate the optimal mutation rate for each of the developed GA based controller to be used across various fault situations.

References

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Berlin: Springer, 2016.

[2] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature,* vol. 521, pp. 503-507, 05/27/2015.

[3] T. L. Theodore, "The Twenty Lost Years of Solid-State Physics," *analog science fact and science fiction,* vol. 75, p. 5, 1965.

[4] C.-S. Lee, M.-H. Wang, S.-J. Yen, T.-H. Wei, I. C. Wu, P.-C. Chou*, et al.*, "Human vs. Computer Go: Review and Prospect," vol. 11, ed, 2016, pp. 68-73.

[5] R. V. Yampolskiy, "AI-Complete, AI-Hard, or AI-Easy-Classification of Problems in AI," in *Midwest Artificial Intelligence and Cognitive Science Conference (MAICS)*, University of Cincinnati, Cincinnati, Ohio, USA, 2012, pp. 94-101.

[6] R. H. Taylor, J. Funda, B. Eldridge, S. Gomory, K. Gruben, D. LaRose*, et al.*, "A Telerobotic Assistant for Laparoscopic Surgery," *IEEE Engineering in Medicine and Biology Magazine,* vol. 14, pp. 279-288, 1995.

[7] M. Hirose and K. Ogawa, "Honda humanoid robots development," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 365, pp. 11-19, 2006.

[8] P. B. Wigley, P. J. Everitt, A. van den Hengel, J. Bastian, M. A. Sooriyabandara, G. D. McDonald*, et al.*, "Fast machine-learning online optimization of ultra-cold-atom experiments," *Scientific reports,* vol. 6, p. 25890, 2016.

[9] B. Lussier, M. Gallien, J. Guiochet, F. Ingrand, M. Killijian, and D. Powell, "Fault Tolerant Planning for Critical Robots," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 144-153.

[10] A. C. Emmanuel and H. Inyiama, "A Survey of Controller Design Methods for a Robot Mainpulator in Harsh Environments," *European Journal of Engineering and Technology Vol,* vol. 3, pp. 64-73, 2015.

[11] T. M. Mitchell, *Machine Learning*: New York : McGraw-Hill, 1997.

[12] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-tolerant Control*: Heidelberg ; New York : Springer, Third edition, 2016.

[13] S. X. Ding, *Data-driven design of fault diagnosis and fault-tolerant control systems*: London : Springer, 2014.

[14] R. Isermann, *Fault-Diagnosis Applications: Model-Based Condition Monitoring: Actuators, Drives, Machinery, Plants, Sensors, and Fault-tolerant Systems*. New York: Springer, 2011.

[15] H. Noura, D. Theilliol, J.-C. Ponsart, and C. Abbas, *Fault-Tolerant Control Systems: Design and Practical Application*. Dordrecht: Springer., 2009.

[16] L. Li, *Fault Detection and Fault-tolerant Control for Nonlinear Systems*: Wiesbaden : Springer Vieweg, 2016.

[17] M. Witczak, *Fault Diagnosis and Fault-Tolerant Control Strategies for Non-Linear Systems: Analytical and Soft Computing Approaches*. New York: Springer, 2014.

[18] A. Zolghadri, D. Henry, J. Cieslak, D. Efimov, and P. Goupil, *Fault Diagnosis and Fault-Tolerant Control and Guidance for Aerospace Vehicles: From Theory to Application*. London: Springer, 2014.

[19] P. Goel, G. Dedeoglu, S. I. Roumeliotis, and G. S. Sukhatme, "Fault Detection and Identification in a Mobile Robot Using Multiple Model Estimation and Neural Network," presented at the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 2000.

[20] K. Kawabata, S. Okina, T. Fujii, and H. Asama, "A Study of Self-diagnosis System for an Autonomous Mobile Robot: Coping with Self-condition Using Internal Sensory Information," *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society,* p. 381, 29/11 - 01/12 2001.

[21] Q. Shen, B. Jiang, and P. Shi, *Fault Diagnosis and Fault-tolerant Control Based on Adaptive Control Approach*: Cham, Switzerland : Springer, 2017.

[22] P. Mhaskar, J. Liu, and P. D. Christofides, *Fault-Tolerant Process Control: Methods and Applications*. London: Springer, 2013.

[23] E. Dubrova, *Fault-Tolerant Design*: New York, NY : Springer, 2013.

[24] H. Dong, Q. Hu, and G. Ma, "Adaptive fault-tolerant controller for satellite proximity operations with finite-time convergence," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, 2015, pp. 3162-3167.

[25] X. Li, F. Zhu, A. Chakrabarty, and S. H. Żak, "Nonfragile Fault-Tolerant Fuzzy Observer-Based Controller Design for Nonlinear Systems," *IEEE Transactions on Fuzzy Systems,* vol. 24, pp. 1679-1689, 2016.

[26] J. Mora and E. de la Torre, "Accelerating the evolution of a systolic array-based evolvable hardware system," *Microprocessors and Microsystems,* vol. 56, pp. 144-156, 2/1/February 2018 2018.

[27] G. Á, J. Mora, A. Otero, E. d. l. Torre, and T. Riesgo, "A Scalable Evolvable Hardware Processing Array," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2013, pp. 1-7.

[28] R. Dobai and L. Sekanina, "Low-Level Flexible Architecture with Hybrid Reconfiguration for Evolvable Hardware," *ACM Trans. Reconfigurable Technol. Syst.,* vol. 8, pp. 1-24, 2015.

[29] E. B. Sedrine, J. Ojeda, M. Gabsi, and I. Slama-Belkhodja, "Fault-Tolerant Control Using the GA Optimization Considering the Reluctance Torque of a

Five-Phase Flux Switching Machine," *IEEE Transactions on Energy Conversion,* vol. 30, pp. 927-938, 2015.

[30] A. R. Merheb, H. Noura, F. Bateman, and J. Al-Jaroodi, "Fault severity based Integrated Fault Tolerant Controller for quadrotor UAVs," in *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015*, 2015, pp. 660-668.

[31] M. Li and Y. Chen, "Sliding Mode Fault-tolerant Control for Switched Systems with Disturbance and Faults," in *2017 11th Asian Control Conference (ASCC)*, 2017, pp. 2884-2888.

[32] P. Zhang and Z. Gong, "Design of H8 Fault-tolerant Controller for Networked Control System," in *2015 Seventh International Conference on Advanced Communication and Networking (ACN)*, 2015, pp. 64-67.

[33] W. K. Clifford, "Preliminary Sketch of Biquaternions," *Proceedings of the London Mathematical Society,* vol. 25, pp. 381-395, 01 / 01 / 1871.

[34] Z. Man, A. P. Paplinski, and H. R. Wu, "A Robust MIMO Terminal Sliding Mode Control Scheme for Rigid Robotic Manipulators," *IEEE Transactions on Automatic Control,* vol. 39, pp. 2464-2469, 1994.

[35] M. P. Aghababa, N. Z. Moghadam, and V. E. Balas, "Nonlinear fault tolerant control design for multimachine power systems," in *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, 2017, pp. 512-517.

[36] S. Jiantao, Y. Yuhao, S. Jun, H. Xiao, Z. Donghua, and Z. Yisheng, "Fault-tolerant formation control of non-linear multi-vehicle systems with application to quadrotors," *IET Control Theory & Applications,* vol. 11, pp. 3179-3190, 2017.

[37] S. Mallavalli and A. Fekih, "A fault tolerant control approach for a quadrotor UAV subject to time varying disturbances and actuator faults," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 2017, pp. 596-601.

[38] G. Wafa, G. Hajer, and B. Mohamed, "Fault Tolerant Control Based on PID-type Fuzzy Logic Controller for Switched Discrete-time Systems: An Electronic Throttle Valve Application," *Advances in Science, Technology and Engineering Systems, Vol 2, Iss 6, Pp 186-193 (2017),* p. 186, 2017.

[39] P. Yang, Z. Gao, J. Zhao, Z. Zhou, and P. Cheng, "Fault Tolerant PI Control Design for Satellite Attitude Systems with Actuator Fault," in *2017 Chinese Automation Congress (CAC)*, 2017, pp. 2026-2030.

[40] A. Machmudah, S. Parman, A. Zainuddin, and S. Chacko, "Polynomial joint angle arm robot motion planning in complex geometrical obstacles," *Applied Soft Computing,* vol. 13, pp. 1099-1109, 2013/02/01/ 2013.

[41] J. R. Anderson, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: an artificial intelligence approach*. Saint Louis, Missouri: Morgan Kaufmann Publishers, Inc., 1983.

[42] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine learning: Algorithms and Applications*. Boca Raton: CRC Press, 2017.

[43] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*: New York : Springer, 2010.

[44] M. Negnevitsky, *Artificial intelligence : A Guide to Intelligent Systems*, Third ed.: Harlow, England; New York: Addison Wesley/Pearson, 2011.

[45] X. Naidenova, *Machine Learning Methods for Commonsense Reasoning Processes: Interactive Models*. Hershey, PA: Information Science Reference, 2010.

[46] J. Miller, *Cartesian Genetic Programming*: New York : Springer, 2011.

[47] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*: MIT Press, 1992.

[48] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," *Data & Knowledge Engineering,* vol. 25, pp. 161-197, 1998.

[49] O. Kramer, *Genetic Algorithm Essentials*: Cham, Switzerland : Springer, 2017.

[50] S. Bandyopadhyay and S. K. Pal, *Classification and Learning using Genetic Algorithms: Applications in Bioinformatics and Web Intelligence*. New York: Springer, 2007.

[51] J. H. Holland, "Genetic Algorithms," *Scientific American,* vol. 267, pp. 66-73, 1992.

[52] B. Bhattarai, M. Shrestha, R. S. Aygun, and M. L. Pusey, "Optimizing Genetic Algorithm for Protein Crystallization Screening using an Exploratory Fitness Function," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2017, pp. 2083-2090.

[53] L. A. Denisova and V. A. Meshcheryakov, "Control System Synthesis Based On Multicriteria Optimization Using Genetic Algorithm," in *2017 Dynamics of Systems, Mechanisms and Machines (Dynamics)*, 2017, pp. 1-5.

[54] Z. Jinghui, H. Xiaomin, Z. Jun, and G. Min, "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, 2005, pp. 1115-1121.

[55] Y. a. Zhang, M. Sakamoto, and H. Furutani, "Effects of Population Size and Mutation Rate on Results of Genetic Algorithm," in *2008 Fourth International Conference on Natural Computation*, 2008, pp. 70-75.

[56] K. A. De Jong, *Evolutionary Computation: A Unified Approach*: Cambridge, Mass. : MIT Press, 2001.

[57] M. Beckerleg and R. Hogg, "Evolving a Lookup Table Based Motion Controller for a Ball-plate System with Fault Tolerant Capabilites," in *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 2016, pp. 257-262.

[58] Z. Yudong, W. Shuihua, and J. Genlin, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," *Mathematical Problems in Engineering,* 2015.

[59] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A Comprehensive Review of Swarm Optimization Algorithms," *PLOS ONE,* vol. 10, 2015.

[60] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, pp. 1942-1948 vol.4.

[61] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Théraulaz, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*: Oxford university press, 1999.

[62] D. Karaboga and B. Akay, "A Comparative Study of Artificial Bee Colony Algorithm," *Applied Mathematics and Computation,* vol. 214, pp. 108-132, 2009.

[63] K. Slany, "Cartesian Genetic Programming in a Changing Environment," in *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, 2015, pp. 204-211.

[64] K. S. Babu and N. Balaji, "Approximation of Digital Circuits Using Cartesian Genetic Programming," in *2016 International Conference on Communication and Electronics Systems (ICCES)*, 2016, pp. 1-6.

[65] G. I. Balandina, "Control System Synthesis by Means of Cartesian Genetic Programming," *Procedia Computer Science,* vol. 103, pp. 176-182, 2017.

[66] J. Kushida, A. Hara, T. Takahama, and N. Mimura, "Cartesian Ant Programming Introducing Symbiotic Relationship Between Ants and Aphids," in *2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA)*, 2017, pp. 115-120.

[67] Y. Zhangyi and Z. Sanyou, "Using Cartesian Genetic Programming to Design Wire Antenna," *International Journal of Computer Applications in Technology,* vol. 43, pp. 372-377, 2012.

[68] J. A. Walker and J. F. Miller, "The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation,* vol. 12, pp. 397-417, 2008.

[69] L.-W. Tsai, *Robot Analysis and Design: The Mechanics of Serial and Parallel Manipulators*: John Wiley & Sons, Inc., 1999.

[70] S. Briot and W. Khalil, *Dynamics of parallel robots : from rigid bodies to flexible elements*: Cham : Springer, 2015.

[71] J. Wittenburg, *Kinematics : Theory and Applications*. Heidelberg: Springer, 2016.

[72] A. Novitarini, Y. Aniroh, D. Y. Anshori, and S. Budiprayitno, "A closed-form solution of inverse kinematic for 4 DOF tetrix manipulator robot," in *2017 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, 2017, pp. 25-29.

[73] J. Choi, D. Park, H. Shin, and S. Rhim, "A New Approach to Generate the DH Parameters of Modular Robots," in *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, 2017, pp. 37-41.

[74] P. Reed, B. Minsker, and D. E. Goldberg, "Designing a competent simple genetic algorithm for search and optimization," *Water Resources Research,* vol. 36, p. 3757, 12/ 1/ 2000.

[75] J. A. Vasconcelos, J. A. Ramirez, R. H. C. Takahashi, and R. R. Saldanha, "Improvements in genetic algorithms," *IEEE Transactions on Magnetics,* vol. 37, pp. 3414-3417, 2001.

[76] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics, Systems, Man and Cybernetics, IEEE Transactions on, IEEE Trans. Syst., Man, Cybern.,* p. 656, 1994.

[77] J. Sun, C.-H. Lai, and X.-J. Wu, *Particle swarm optimisation : classical and quantum perspectives*: Boca Raton [Fla.] : CRC Press, 2012., 2012.

[78] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942-1948 vol.4.

[79] S. Shanmuganathan and S. Samarasinghe, *Artificial Neural Network Modelling*: Cham : Springer, 2016.

[80] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*: Cambridge, Mass. : MIT Press, [1996], 1996.

[81] N.-K. Im and V.-S. Nguyen, "Artificial neural network controller for automatic ship berthing using head-up coordinate system," *International Journal of Naval Architecture and Ocean Engineering,* 8/1 2017.

[82] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-13, pp. 834-846, 1983.

[83] F. Mondada and D. Floreano, "Evolution of Neural Control Structures: Some Experiments on Mobile Robots," *Robotics and Autonomous Systems,* vol. 16, pp. 183-195, 1995.

[84] A. N. Mohammed and I. Kostanic, "Stochastic Gradient Descent for Reducing Complexity in Millimeter Wave Hybrid Precoding Design," ed: IEEE, 2018, p. 1.

[85] M. Duarte, S. Oliveira, and A. L. Christensen, "Hierarchical Evolution of Robotic Controllers for Complex Tasks," in *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, 2012, pp. 1-6.

[86] G. E. U. Faelden, J. M. Z. Maningo, R. C. S. Nakano, A. A. Bandala, and E. P. Dadios, "A Neural Network Approach to a Cooperative Balancing Problem in Quadrotor-unmanned Aerial Vehicles (QUAVs)," in *2015 International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, 2015, pp. 1-5.

[87] V. Subramanyam, D. Srinivasan, and R. Oruganti, "A Dual layered PSO Algorithm for evolving an Artificial Neural Network controller," in *2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 2350-2357.

[88] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A Time-delay Neural Network Architecture for Isolated Word Recognition," *Neural Networks,* vol. 3, pp. 23-43, 1990.

[89] S. P. Knerr, L.; Dreyfus, G, "Handwritten Digit Recognition by Neural Networks with Single-Layer Training," *IEEE Transactions on Neural Networks, Neural Networks, IEEE Transactions on, IEEE Trans. Neural Netw.,* p. 962, 1992.

[90] N. Yadav, A. Yadav, and M. Kumar, *An introduction to neural network methods for differential equations*: Dordrecht : Springer, 2015.

[91] K. Jinwook, K. Yoon-Gu, and A. Jinung, "A Fuzzy Obstacle Avoidance Controller Using a Lookup-Table Sharing Method and Its Applications for Mobile Robots," *International Journal of Advanced Robotic Systems, Vol 8, Iss 5 (2011),* 2011.

[92] S. Skjong and E. Pedersen, "Model-based control designs for offshore hydraulic winch systems," *Ocean Engineering,* vol. 121, pp. 224-238, 7/15/15 July 2016 2016.

[93] Y. J. Huang and S. H. Chang, "Table look-up variable structure control for robotic arms," in *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*, 2004, pp. 585-591 Vol.1.

[94] J. Currie, M. Beckerleg, and J. Collins, "Software Evolution Of A Hexapod Robot Walking Gait," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 305-310.

[95] T. Bäck, P. Krause, and C. Foussette, *Contemporary Evolution Strategies*. Heidelberg: Springer, 2013.