University of Southern Queensland
Faculty of Health, Engineering & Sciences

Optimal fuel cost controller design for a helicopter/twin rotor system

A dissertation submitted by

A. Coutts

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Electrical & Electronic Engineering

Submitted: October, 2021

Abstract

The TRMS (Twin Rotor Multi-Input-Multi-Output [MIMO] System) 33-949 system is a small-scale model of a helicopter, exhibiting many similar characteristics, except for limited degrees of freedom. The non-linearities, as well as cross-couplings between the inputs and outputs of such a system make the model a useful basis for designing, testing, and deploying a broad range of control algorithm implementations, and allow for experimentation and exploration of various optimisation techniques.

Many Machine and Deep Learning techniques have been discovered that allow researchers to optimise control algorithms based on multiple objectives. Research was conducted to find a method of optimisation that allowed the search of a large space of candidate solutions to minimise an objective function of error and cost. The evolutionary-based genetic algorithm was suitable, demonstrating good results on the TRMS system. Memetic Algorithms (MA) were discovered to have been applied in system identification, but not yet applied in finding optimal control parameters for a TRMS system. A MA is of the same family of algorithms as a Genetic Algorithm (GA), so the performance of the memetic algorithm is directly comparable to the established research on GA and its application to the TRMS.

Implementation and testing has revealed that the MA does outperform an equivalent GA (the same algorithm, with the Local Search removed) in every test case, with typical fitness value improvements of anywhere from 5% to 120% (and greater improvements are likely as observed from the results data). Of particular interest was the capability of MA for finding simultaneous decoupling and control solutions. Test results have indicated however, that significant cross-couplings still exist, particularly in the pitch-to-yaw path. Therefore, the MA has been useful for designing a cost/error optimal controller, especially for Single-Input-Single-Output (SISO) applications, but as with many developments in

AI, it certainly is not without limitations.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

A. Coutts

Acknowledgments

Thanks are in order to my supervisors Prof. Paul Wen and Dr. Bo Song, for their support and guidance throughout this project.

I would like to specially thank my little brother. The youngest sibling of two who was undeniably helpful in all ways for the completion and success of this thesis, except that through his good-hearted, yet questionably helpful discussion, added unnecessary stress in the final months of completing this thesis. His intentions and actions remained pure, yet the outcome was a hindrance. Thus, 'twas a burden of love to withhold my fist from his larynx. Nevertheless, I thank him for his support, and for being a bearable housemate.

In all seriousness, I would like to thank all my friends and family for their unwavering support, not only through this thesis, but throughout my entire university journey. Thanks to the uni crew for all the great (and questionably productive) study sessions we had, and thank you for dragging me away from the studies when I clearly needed it. Thank you to Jacob, Josh, Kieran, Nathan and Tim - the close friends that I've held since primary and high-school - for always making time to be there for me, even though I may drop off the map for extended periods of time. Finally, thank you to my family for all the love, help and support at every single step of the way.

A. Coutts

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The twin rotor aircraft, most commonly known as the Helicopter, has many applications across various transport, industrial and military operations. Helicopters have an advantage over most fixed-wing aircraft, with the ability to take off, hover, and land vertically. When compared to fixed-wing counterparts, rotary-wing aircraft operational aerodynamic environments are often complex, introducing a plethora of issues that have become the focus of much research (Wilbur et al. 2018).

An optimal flight pitch control system in a rotor-wing aircraft is important to many operational aspects. High frequency system characteristics and high overshoot can induce increased wear and damage to the structure of the aircraft and can cause discomfort and potential harm to passengers. Furthermore, because these characteristics are undesirable, the energy consumed performing such actions is wasted, thus reducing useful fuel and increasing operational costs. The conventional PID controller is frequently utilised in industry for multiple applications due to the simple architecture, and relative simplicity in tuning for given control objectives. More than 95% of industrial control systems are PID-based (Serradilla et al. 2020). While the PID is simple and spans many applications, complications arise when controlling nonlinear dynamical systems, such as the twin rotor system. Because of the widespread expertise in implementation of the PID controller, much research has focused on methods to optimise the capability of PID controllers in regard to nonlinear dynamical systems (Norsahperi & Danapalasingam 2020).

Sarvart (2001) outlines the design and implementation of control for contemporary systems such as the twin rotor system. System design and requirements definition, followed by modelling and analysis are the first two steps within the overarching process. With a physical system created, and a model derived, a controller can then be designed, analysed and implemented.

System Design and Requirement, as the name suggests, involves identifying requirements and designing the physical system. Often, through this stage, various control engineers are involved to provide expert input, ensuring the system is compatible with control techniques to be implemented later in the process. However, it is the task of many engineers to provide control systems for a physical system or plant which has already been designed and built.

The next step in the process, **Modelling**, is often the most difficult, and equally one of the most important. The model must accurately represent the real-world system, as this is used as the basis in designing, simulating and implementing the control law. Models can be obtained either by mathematical modelling of the system's physical properties, or through iterative system identification methods.

In the third step, **Control Design**, the operating conditions of the plant must be determined, and an appropriate design methodology selected to suit. Many control paradigms are available, with approaches dependent on plant type, i.e. either Single-Input Single-Output (SISO) or Multi-Input Multi-Output (MIMO). Classical techniques such as frequency domain methods are typical candidates for SISO systems, whereas modern control techniques such as state-space approaches are often better suited to MIMO systems. **Control Analysis** may also be viewed as the validation phase. This step allows the control design to be verified against requirements, and if unsatisfactory, a different control paradigm may be chosen, or, if possible, the system design itself may be altered.

Control Implementation, the final step in the process, is the operational deployment of the control algorithm to the physical plant. Although implementation is the last step, it is often not the final step to be performed. Real operational environments are far from perfect, and real, non-ideal conditions can affect the real-world performance of the control algorithm. Many disturbances such as noise and non-linearities are present in practice, and because imperfections are often difficult to realise mathematically, these can often be unaccounted for. Therefore, if the control does not perform as per requirements, the

process is reiterated until the system control is satisfactory.

The development of highly sophisticated Unmanned Aerial Vehicles (UAV) is a byproduct of the surge in research performed on Artificial Intelligence (AI) techniques, such as genetic algorithms, neural networks and fuzzy-logic, all of which address core issues such as system modelling and optimal control. Whilst work surrounding AI is continuously gaining more traction, such techniques are yet to be widely accepted within the control systems industry, with historical data, literature and track record still growing. Further study within the fields of AI and optimal control are of great importance, as a better understanding of such technologies are crucial to widespread adoption across many industries.

1.2 The Twin Rotor MIMO System

1.2.1 Overview

The 33-949S Twin Rotor MIMO System (TRMS) is an excellent candidate for performing experiments and simulations due to the striking similarities to real-world twin rotor systems (Sarvart 2001). However, the TRMS has some notable simplifications; the system is attached to a tower, offering only 2 degrees of freedom (DOF), and the position and velocity are controlled through varying the speed of the rotor (Feedback Instruments Ltd. n.d.). Real-world helicopters are not attached to a tower, and as indicated by Salazar Alvarez (2010), offer six DOF. Further, a real helicopter's rotor speed is generally constant, with propulsion controlled by varying blade angles. Further similarities between the TRMS and a real helicopter lie within the ability to capture most dynamical characteristics. Perhaps, of particular interest, is the cross-coupling between both rotors; an activation of one input, e.g. vertical position, will also cause an activation of movement in the horizontal plane (Feedback Instruments Ltd. n.d.).

Both rotors of the TRMS are mounted on a beam with a counter balance, fixed to a tower mounted on a base containing the electrical unit, which is important for interfacing between the model and a PC. There are two inputs on the 33-949S system: the voltages supplied to each of the two rotors, and two outputs: vertical and horizontal angles, and angular velocity (Feedback Instruments Ltd. n.d.).

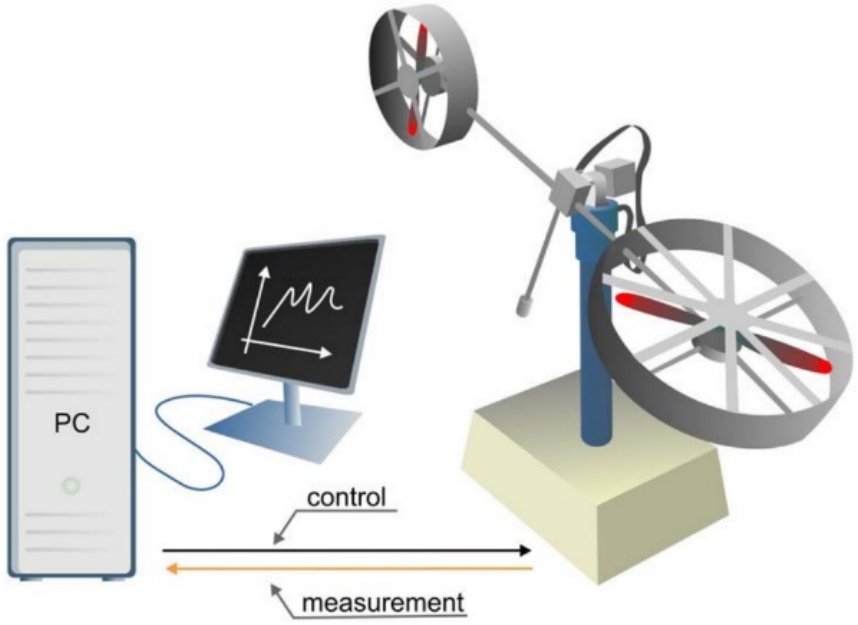

Figure 1.1: The manufacturer's image of the overall TRMS control system (Feedback Instruments Ltd. n.d.)

1.2.2 TRMS Model

Feedback Instruments Ltd. (n.d.) have included the derived equations for the parameters of the TRMS system within the 33-949S manual. It is important to note that some parameters are arguments of nonlinear functions, and that to form a transfer function of the system, the system must be linearised. The manual also presents experimentally-chosen values for the parameters, which may be substituted into the equations to present a semi-phenomenological representation of the system to be used in simulation.

A simplified system schematic is also provided in the manual, presented in Figure 1.2, which demonstrates the simple mapping between inputs and outputs, and the cross coupling modes, where ψ is the pitch and φ is the azimuth (yaw) rotation.

1.3 Project Aims and Objectives

Considering that classical control design methods have been extensively researched, and are already widely accepted within industry, the focus of this project is drawn to investigating AI techniques, with particular interest drawn to methods that may have not yet

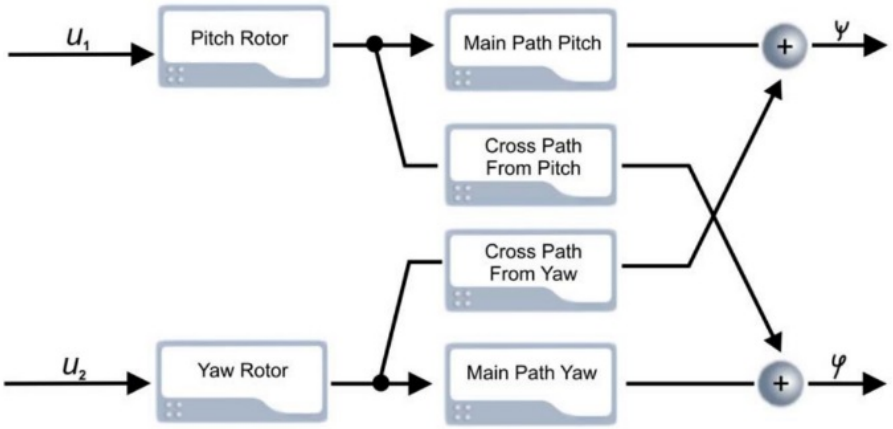

Figure 1.2: The manufacturer's simplified system schematic of the TRMS control system (Feedback Instruments Ltd. n.d.)

been applied to the TRMS system.

The aim stated for the project is to *Optimise fuel performance to reduce operational costs of helicopter twin rotator systems*. It is desirable to continue research into potential algorithms within AI to achieve a fuel-optimal system, but as indicated in previous sections, classical methods are indeed widely accepted, and methods which incorporate popular components such as the aforementioned PID controller may be more easily accepted within industry. AI is an important tool that should be harnessed to achieve continuous improvements, and solve problems that are otherwise very difficult, or even impossible with classical methods.

To ensure these aims are realised, the following objectives are in place for this project:

- Research and analyse classical methods of control to determine if any immediate improvements could be made on these.
- Research and investigate AI methods used to solve similar problems. The literature review presented a few key pieces, but there are plentiful methods to be researched.
- Research Deep Learning algorithms in particular and determine if these may be useful for solving control problems.
- Apply a novel algorithm, or make improvements to existing methods to achieve the aims of better fuel efficiency, without sacrificing performance.
- Investigate the applicability of the results and real-world inferences. In practice,

this will include testing on a test rig in the lab.

The scope of this project is broad, as a scan of much existing literature is required to evaluate different potential methods, and apply the most appropriate to achieve the aims and objectives of this project. The work completed in this project may be extendable to systems of higher DOF, but preliminary priority is placed on ensuring that a functional method is employed for a 2 DOF system, such as the 33-949S TRMS.

1.4 Dissertation Overview

An overview of the dissertation goes here.

Chapter 2

Literature Review

This chapter aims to provide a broad review of the existing literature in the control of TRMS systems, ranging from traditional methods to AI techniques. A more comprehensive study is conducted of the evolutionary Genetic and Memetic Algorithms, which is integral to the work conducted in this research project.

2.1 Robust Deadbeat Decoupling & Control

System decoupling, especially with strongly-coupled systems such as the TRMS, is vital to ensuring that control algorithms can perform optimally. A paper by Wen & Lu (2008) investigates decoupling of a twin rotor system using the robust deadbeat control technique. A thorough analysis of the system's mathematical model is developed to simulate the system as accurately as possible. In the deadbeat control system, the design could be insensitive to parameter variations of up to 50%. The robust deadbeat control method is PID-based, and separates each of the input-output paths into two SISO 1 DOF paths and models, each according to the structure outlined in Figure 2.1

The representation between the output and input is known as the transfer function, and for this system is derived as:

$$\frac{C(s)}{R(s)} = \frac{G_c(s)G(s)}{1 + G(s)H_2(s) + G_c(s)G(s)H_1(s)} \tag{2.1}$$

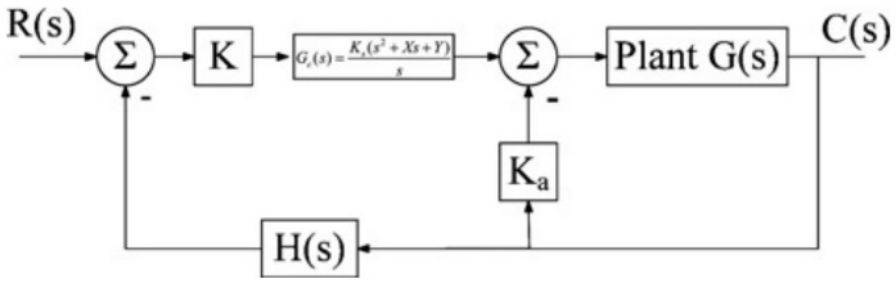

Figure 2.1: Typical control loop for a Robust Deadbeat Control System (Wen & Lu 2008)

Where the $H(s)$ feedback path is $H_1(s)$ and the K_a feedback path is $H_2(s)$. The plant is represented as $G(s)$ and the PID control portion, which is the segment cascading between the two summing junctions, is represented by $G_c(s)$.

Substituting the values of each of these functions yields a fourth order closed-loop transfer function of the form:

$$T(s) = \frac{\omega_n^4}{s^4 + \alpha\omega_n s^3 + \beta\omega_n^2 s^2 + \gamma\omega_n^3 s + \omega_n^4} \tag{2.2}$$

This can be normalised to the form:

$$T(s) = \frac{1}{\bar{s}^4 + \alpha\bar{s}^3 + \beta\bar{s}^2 + \gamma\bar{s} + 1} \tag{2.3}$$

where

$$\bar{s} = \frac{s}{\omega_n} \tag{2.4}$$

In this form, the coefficients α, β, γ and $T_{s'}$ can be obtained from the robust deadbeat control lookup table. The desired settling time was chosen by the designers to be 2 seconds and with K initially set to 1, all unknown values in the transfer function were able to be calculated by virtue of simultaneous equations, and substitution.

With the parameters in the control loop for both the main rotor and tail rotor determined, the designers were able to form the robust deadbeat control MIMO system as shown in Figure 2.2.

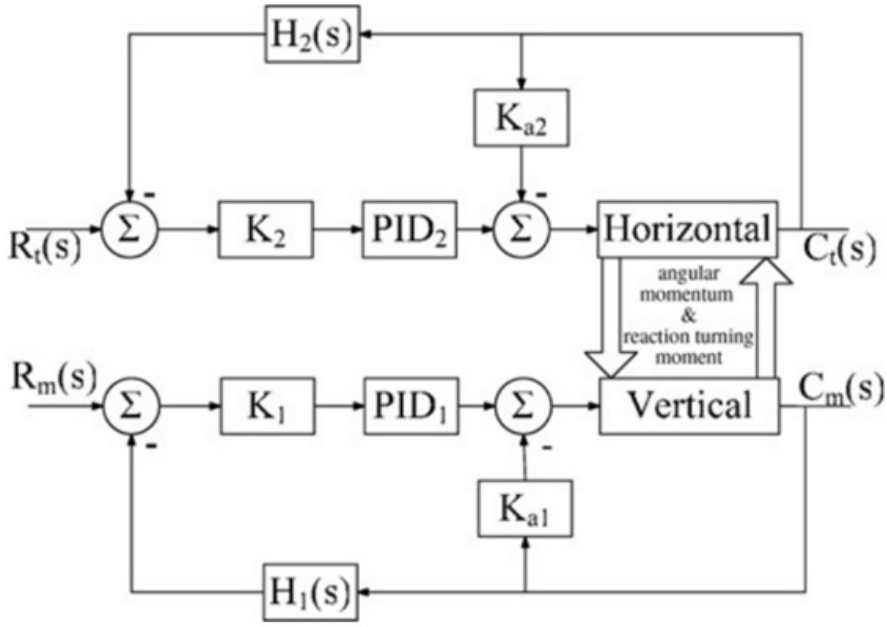

Figure 2.2: Final Deadbeat Robust Control MIMO model (Wen & Lu 2008)

Both 1 DOF cases of tail and main rotor are simulated individually, ignoring the cross coupling. The K parameters K_1 and K_2, which mentioned previously were set to 1 initially, are both tuned to achieve a deadbeat response. The deadbeat control was then applied to the 2 DOF case, which includes the full model of the TRMS system – including the cross-couplings. The desired settling time in this setting is 4 seconds for both tail and main rotors.

In all cases, the control system met or exceeded all specifications and proved to suppress the effects of cross-coupling. The results were compared to a simple PID control of the system, and demonstrated that the robust control method shows a much more accurate response to a step input than the simple PID control. Simple PID control demonstrated drastic overshoot and settling oscillation, whereas the robust deadbeat control method response follows the step envelope well, with very little overshoot and oscillation.

For the tail rotor SISO system, a settling time of about 6 seconds is achieved, and overshoot amount reduced by 20%. For the SISO main rotor, a settling time of 12 seconds was achieved, and overshoot was fully eliminated. Similarly for the full 2 DOF system with cross-coupling, settling time was reduced to 20 seconds and the amount of overshoot was also reduced. Although strong effects of nonlinear components was predicted, the system demonstrated resilience to parameter variations, as predicted by the insensitivity to 50% plant parameter variations (Wen & Lu 2008).

2.2 System Identification

Chalupa et al. (2015) aimed to design a higher-accuracy, valid model of the Twin Rotor MIMO system to be used as the basis of further research. The work primarily utilises a 'grey box' approach, which involves first deriving the model from first principles (white box modelling), and then performing measurements of the system to further refine the model. The white box modelling from first principles is similar in nature to that presented in Wen & Lu (2008). The first enhancement that was performed to the model, was the measurement of certain parameters, such as the length of the counter-weight beam, to ensure that system parameters were as accurate to the real system as possible. Any nonlinear static functions from the mathematical model were then determined using a phenomenological approach and polynomial approximation. Linear approximation was used to model the cross-coupling of tail motor to elevation, whereas due to the complexity associated with modelling the azimuth rotations caused by the main rotor, an exponential function of moment was used. Static characteristic measurements of the main motor show a piecewise linear relationship between elevation and control voltage. When changing the sign of the control voltage, and thus the rotation direction of the propeller, the gain of the system is greater in the positive control voltage range, as compared to the negative range. Thus the system can be considered linear when operating within each of these ranges exclusively.

The research showed that enhancing the non-linear model using grey box identification did produce a system model more accurate than white box modelling alone. It must be noted however, that tests completed on the tail and main rotors were each isolated, meaning with one input set, the other is set to 0. The paper did not indicate that tests were completed whilst setting both inputs.

The following plant model equation was obtained for the main rotor:

$$G_v(s) = \frac{111.2}{0.3954s^3 + 0.3835s^2 + 1.463s + 1} \tag{2.5}$$

The plant model equation obtained for the tail rotor:

$$G_h(s) = \frac{700.7}{5.61s^2 + 3.992s + 1} \tag{2.6}$$

Sarvart (2001) performs extensive work in black box identification for the TRMS MIMO system. Sarvart acknowledges that mathematical modelling is best suited to simple systems, and that complex systems are much more difficult to analyse using such methods. Often, specialist knowledge is required for the accuracy of details such as the behaviour of electro-mechanical components, for example. Another important point that is raised, is that mathematical models often do not account for environmental disturbances or dynamics that are often ignored, and thus it is inferred that mathematical models are simply limited. System identification through black box methods are often almost necessary for new aircraft in which parameters and structural details are unknown.

Whilst the work by Sarvart focuses on determining rigid body modes and other details necessary for vibration control (which is the aim of the study), the researchers were able to determine a black box model with a high degree of confidence, obtained through validation of both time and frequency domain analyses. A discrete-time transfer function was identified for 1 DOF main rotor input to pitch angle output. An analysis of the poles in the transfer function indicate directly-related physical properties of the structural material, such as the existence of critically stable oscillatory modes and state variables such as rigid body motion. Strong couplings were identified between main rotor and pitch, main rotor and yaw, tail rotor and yaw, however, very little interaction between tail rotor and pitch was observed and was thus omitted from a model fit. The interaction between main rotor and yaw was also identified as imperfect, however the resulting model was still deemed acceptable for use in controller design.

2.3 Artificial Intelligence Techniques

Thus far, traditional methods of control, decoupling techniques and system identification have been briefly discussed. These methods are widely accepted in industry and are reasonably straightforward in implementation. However, whilst these methods are reliable and produce good results, AI techniques have continuously shown through many fields, that better results are achievable. Plentiful research has been conducted in the application of AI to solve problems in control, such as optimisation and modelling. Increased research is critical for increased industrial confidence in implementing methods based on AI techniques.

Serradilla et al. (2020) investigates the use of AI in tuning parameters of controllers for DC motors based on two objectives: high productivity and efficiency. Meta-heuristic AI techniques, based on genetic algorithms, were utilised to adjust PID controller parameters to achieve the desired objectives. The meta-heuristics approach searches for optimal solutions within a given problem space, which is know as exploration. Due to the highly-dimensional nature of the problem space, it is generally not feasible to explore all potential solutions. Therefore, reasonably good solutions are found and then further refined.

The first step in the paper presented by Serradilla et al. (2020) involves gaining a description of the model through mathematical modelling. This step is synonymous with system identification through white box modelling. A relevant point to note is that the energy expenditure of the DC motor is modelled. A model of the energy expenditure of the system is necessary if this is to be optimised.

It is helpful, and also necessary, that a brief review of PID control is presented in the paper. A set of performance indexes are useful for determining the accuracy of the controller, and provide a means of assessment for the choice of gains and parameters selected in implementation. Whilst the details are not presented here as these are reasonably well-known and are learnt in most undergraduate engineering control courses, the most widely used performance indexes are settling time, decay ratio, overshoot and steady-state error. These performance indexes are the most common and are determined against step changes in set point response. If the reader is unfamiliar with these, it is recommended to consult the referenced paper for the definitions, as well as standard or typical values. Serradilla et al. highlights that whilst there are frequently used approaches to designing PID controllers such as the root-place and Ziegler-Nichols methods, these hardly guarantee the most optimal solution.

The next step presented in the paper by Serradilla et al. (2020) is optimisation with meta-heuristics. An objective function must be constructed to satisfy the requirements of an optimisation problem. First, weights were assigned to each of the performance indices, using prior work by (Naranjo et al. 2020). The four weighted parameters, plus a weighted energy parameter, are used to formulate an equation of weights, which can then be used to construct an error equation. The equation of weights that is formulated in the paper

is as follows:

$$\epsilon = W_e \frac{\alpha + \beta + \gamma + \delta}{100 - W_e} \tag{2.7}$$

α, β, γ and δ are the weighted parameters of settling time, decay ratio, overshoot and steady-state error, respectively. W_e is the weighted energy parameter, which is set dependant on importance.

The error function that is used is a weighted average function as follows:

$$error = \frac{1}{s}\frac{1}{5}\sum_{s=1}^{s} \alpha \cdot tss_s + \beta \cdot d_s + \gamma \cdot o_s + \delta \cdot ess_s + \epsilon \cdot energy_s \tag{2.8}$$

The error is then utilised in the fitness function, which is defined as:

$$fitness = \frac{1}{1 + error} \tag{2.9}$$

The paper also specifies the Genetic Algorithm architecture that is utilised for evolutionary computation optimisation. Ultimately the fitness function is optimised to obtain a value as close as possible to 1, so that, satisfying Equation 2.9, the error approaches 0.

Results of the research saw the ability to find optimal control parameters whilst considering both performance and energy criterion. Furthermore, it was discovered that increasing the weight of energy results in poorer performance of settling time, decay ratio, overshoot and steady-state error, although some combinations were identified that demonstrated good energy savings, with little impact to performance. A final note is that the importance of energy consumption is specific for each real motor. Thus, an analysis of a motor, or in the general case the plant, must be performed before applying the strategy (Serradilla et al. 2020).

The focus of work in the paper by Doğruer & Tan (2019) concerns using FOPID optimisation techniques to perform decoupling of a twin rotor MIMO system. Genetic Algorithms are again utilised in the optimisation process, due to heuristic methods performing better than classical design methods. The fitness function used in the genetic algorithm architecture is the Integral of Time-weighted Squared Error (ITSE). The FOPID is similar

to a PID controller, except there are 2 additional parameters required to be tuned. The transfer function of a conventional PID controller is:

$$C(s) = K_p + \frac{K_I}{s} + K_d s \tag{2.10}$$

where each of the K terms are the Proportional, Integral and Derivative terms, respectively. The FOPID controller introduces a degree of integral term λ and a degree of derivative term μ, such that the equation then becomes:

$$C(s) = K_p + \frac{K_I}{s^\lambda} + K_d s^\mu \tag{2.11}$$

It is important to note that the FOPID controller is a suitable candidate when designing for performance criteria such as robustness and stability.

Doğruer & Tan (2019) achieves System Identification through the black box method. The researchers make use of MATLAB's System Identification toolbox. The identification yielded 4 mathematical models; one for each of the paths from input to output (including cross-couplings). With models obtained for each path, decoupled transfer functions can be obtained by decoupling techniques covered in the work by Ben Hariz & Bouani (2015). Using the ITSE performance criterion in the genetic algorithm, FOPID controller designs were created for each of the decoupling transfer functions to yield decoupling controllers. When compared to a generic PID tuned using the Simulink model that the researchers obtained, an 8% difference between maximum percent overshoot was observed for pitch position. Whilst a 1.8% overshoot was observed with the FOPID for yaw position control, and 0% overshoot was observed for the generic PID, the settling time of the FOPID was 7.87 seconds, as opposed to 50.37 for the generic PID.

In a paper by Norsahperi & Danapalasingam (2020), the tuning of Fractional-Order PID controllers is investigated, examining methods such as particle swarm-based FOPID (PSOFOPID) and neuro-based FOPID (NNFOPID). The motivation behind the research is similar to that found in the work by Serradilla et al. (2020), in that whilst the controller is industry-standard, the methods of tuning do not guarantee optimal solutions.

For the PSOFOPID research performed, a new method was proposed for setting the initial search range which prevented the algorithm from being trapped in local optima,

reducing the problem search space. For the NNFOPID research, the algorithm was able to tune more practical controller parameters without the need for deep knowledge of the system, and resulted in a lighter network. Five criteria points were used to assess the performance of each algorithm: Square-wave characteristics, reference to disturbance ratio, evaluation time, energy consumption of control signal and tracking performance. Notably, the test environment was controlled using 3 different cases: no coupling effect and wind disturbance, coupling effect only (no wind disturbance) and wind disturbance only (using a wind velocity of 4.01 m/s) (Norsahperi & Danapalasingam 2020). The researchers compared the results of each algorithm against a PID controller, optimised by pre-searched genetic algorithms (Juang et al. 2008). Energy consumption performance of the designed controllers is also assessed for each tuning algorithm. The energy of control input was measured and showed 31% reduction for the NNFOPID-tuned controller, and 5% reduction for the PSOFOPID-tuned controller. The NNFOPID method also offers accurate system positioning through reducing the steady-state error by 34% in the cross-coupling-only case. PSOFOPID control is useful for a 27% reduction in tracking error and yields the lowest oscillation in the cross-coupling-only case. Both methods were also demonstrated to be robust and efficient in the wind-disturbance-only case (Norsahperi & Danapalasingam 2020).

Ant Colony based model prediction of a Twin Rotor System is investigated in a paper by Toha et al.(2012). The focus is drawn to system identification, and inferring an accurate mathematical model. The components of an Ant Colony Optimisation (ACO) meta-heuristic are sets of ant-like agents, usage of memory, stochastic decisions, and collective and distributed learning strategies. The results of the research indicate that a stable and satisfactory model can be extracted using the algorithm. The algorithm was also 99.33% accurate in predicting outputs at each time step, as compared to the actual output.

2.4 Evolutionary Algorithms

This section and subsequent sections aims to draw the reader's attention closer to Evolutionary Algorithms, and in particular Genetic & Memetic Algorithms, for which is the primary focus of this paper.

In 1859, Charles Darwin published the book *Origin of Species*, in which he revealed

the theory of evolution by natural selection (Ayala 2009). Since then, much work has progressed within science that further validate and verify the theories of evolution. It is from this work that scientists and engineers alike have been inspired by evolution theory's more abstract applications, such as those within mathematics and computing. The development of Evolutionary Algorithms is a product of inquiry over recent decades, and as the name conveniently suggests, covers a broad range of programming algorithms which are based on the theory of evolution.

Through studying the applications of evolutionary theory to computing, various subsets of algorithms have been discovered. Evolutionary Algorithms (EA) have birthed various subsets of traditional algorithms such as, Evolutionary Strategy, Genetic Algorithms (GA), Genetic Programming, Genetic Improvement, Grammatical Evolution, Linear Genetic Programming, Cartesian Genetic Programming, Differential Evolution and Gene Expression Programming. Specialised techniques include Auto-constructive Evolution, Deep Neuroevolution, Self-replicating Neural Networks, Markov Brains, PushGP, Simulated Annealing, Tanlged Program Graph, Tabu search and Animal inspired algorithms (Sloss & Gustafson 2020). Each technique have appropriate problem domains in which they are most effective to be employed.

2.4.1 Genetic Algorithms

Of all EAs available, the Genetic Algorithm (GA) is the most popular (Sloss & Gustafson 2020). Genetic Algorithms utilise methods that are analogous to genetic evolutionary processes to search for optimal solutions in an often wide solution space that is unsolvable by existing efficient algorithms or methods. Alam et al. (2020) summarises the components of a Genetic Algorithm in their paper covering the review, implementations and applications of Genetic Algorithms. The concepts will be extrapolated using a PID tuning scenario to better illustrate the concepts involved. The first consideration is usually the objective function; this is an equation or set of constraints that combines target characteristics, such that the equation can either be minimised or maximised to obtain the best candidate solutions. An example of this has been covered in a previous section of the literature review when analysing the work by Naranjo et al. (2020), where some of the desired variables were step response characteristics such as settling time, overshoot and energy. Candidates are the variables that are required to be tuned in order to achieve

the desired fitness of the objective function. In the case of this example, the candidates are PID control variables K_p, K_i and K_d. The candidates are usually encoded to a suitable representation, i.e. binary strings, however, for this case, it is most appropriate to keep the PID variables as floating-point numbers, as these are the type in which they are implemented.

To start the Genetic Evolution, a random population of potential candidates must be generated. Each individual candidate is a solution to the target problem. The population is propagated through each iteration, better known as a generation in evolutionary nomenclature. For each iteration, each candidate in the population is evaluated against the objective function and a random selection of the fittest individuals is chosen to be part of the next generation. A random sample of these individuals undergo crossover, in which parent candidates combine attributes to produce offspring candidates. Another randomly chosen sample set is subject to mutation, in which the individual is randomly altered to introduce diversity in the population. The former process is used to exploit good characteristics in candidates and encourage convergence to an optimum, whereas the latter ensures that a local optima is not overly-exploited, ensuring that more ground is covered in the hypothetical solution landscape, and that other potential areas of global optima are discovered. The balancing of both exploitative and exploratory procedures is also the subject of much research (Zhao et al. 2008). The final step in a generation involves what could be essentially termed as 'survival of the fittest', in which, through a semi-stochastic method (where fitness is used as a weighting), individuals are selected to become part of the following generation. Through iterating this process through a number of generations, the population will become 'fitter', and when generated offspring demonstrate no significant differences to previous populations, the candidates can be said to have converged to an optimum. In some instances, convergence may not occur, and other termination criteria may be met, causing the algorithm to conclude. A common termination may simply be that the maximum number of generations has been exceeded. A general overview in the form of a flowchart is depicted in Figure 2.3.

2.4.2 Genetic Algorithms and Control

GAs are certainly not scarce within existing literature, and a brief survey is presented here to demonstrate the breadth of research, especially as applied to tuning PID parameters.

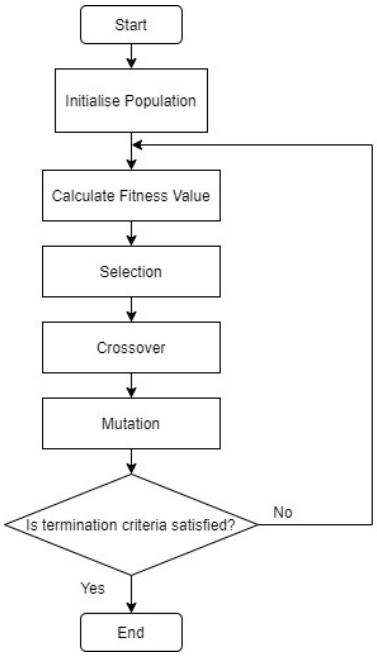

Figure 2.3: Basic General Flowchart of a Genetic Algorithm (Albadr et al. 2020)

In a paper by Jayachitra & Vinodha (n.d.), a GA is applied to a PID controller for a non-linear Continuous Stirred Tank Reactor plant. The researchers found that the optimised PID allowed the plant to operate within the entire operating range, overcoming limitations of the linearity of the PID, with satisfactory set point tracking and disturbance rejection. The objective function to optimise in the genetic algorithm was the weighted sum of Integral Square Error (ISE), Integral Absolute Error (IAE) and Integrated Time Absolute Error (ITAE), as the weighted sum indices are more appropriate than the performance of each standalone index. The definition for each of these functions is given in Equations 2.12, 2.13 and 2.14

$$ISE = \int_0^\infty [e(t)]^2 \, dt \tag{2.12}$$

$$IAE = \int_0^\infty |e(t)| \, dt \tag{2.13}$$

$$ITAE = \int_0^\infty t \, |e(t)| \, dt \tag{2.14}$$

where $e(t)$ is the difference between the input setpoint signal $r(t)$ and the output signal

$y(t)$, i.e.

$$e(t) = y(t) - r(t) \tag{2.15}$$

The weighted sum equation takes the form:

$$J(K_p, K_I, K_D) = w_1\ (ISE) + w_2\ (IAE) + w_3\ (ITAE) \tag{2.16}$$

where the weights w_1, w_2 and w_3 were chosen to be 0.4, 0.2 and 0.4 respectively. The researchers utilise Binary-String Encoding for representing the parameters of the PID (the genes in evolutionary terms). The reproduction probability formula used is:

$$P_{ri} = \frac{F_i(\theta)}{\sum_{i=1}^{P_l} F_i(\theta)} \tag{2.17}$$

The crossover method, population and generation sizes were left unspecified, and the mutation was briefly explained to be achieved through randomly flipping a bit in a gene at a uniformly random chosen index.

Mirzal et al. (2012) investigates using a genetic algorithm to tune the gains of a PID controller for a First Order Lag plus Time Delay (FOLPD) system. The researchers' approach involved comparing the algorithm against multiple objective functions, the Iterative Method and the Ziegler-Nichols Method, and selecting the best one. The objective functions were ISE, IAE, ITAE, as defined previously, with another two common objective functions included in the comparison, namely MSE and ITSE, as defined in Equations 2.18 and 2.19

$$MSE = \frac{1}{t} \int_0^\tau [e(t)]^2\ dt \tag{2.18}$$

$$ITSE = \int_0^\tau t\ [e(t)]^2\ dt \tag{2.19}$$

Additionally, the fitness value for each chromosome is defined as:

$$fitness = \frac{1}{performance\ index} = \frac{1}{J} \tag{2.20}$$

where performance index is the objective function used (usually defined as J). It appears that the researchers used MATLAB's implementation of genetic algorithms. For the genetic algorithm, the termination criteria was set to 300 maximum generations, the population was chosen to either be 80 or 100 chromosomes, floating-point was used to encode the genes, Normalised Geometric Selection was the preferred parent selection technique, arithmetic crossover for floating point numbers were used, with 4 crossover points selected and the mutation rate set to 0.1%. Standard performance measures were analysed: percentage overshoot, settling time, rise time, peak time and stability margin. Criteria for settling time was chosen to be 5% and 0-95% criterion was used for rise time. As predicted, the GA methods performed better than the Iterative and Ziegler-Nichols Method. Perhaps unsurprisingly, each objective function yielded different characteristics, with strong performance in some indices, whilst weaker in others.

The majority of a paper by Devanshu (2017) describes the GA configuration used to tune a PID controller for a process control. The population was chosen to be about 80, as research suggests that a population size between 30 to 100 is usually optimal, and the researcher's own experimentation revealed that out of experiments with 40, 60 80 and 90 chromosomes, sizes of 90 and above do not result in much improvement. For reproduction selection, the Roulette Wheel method was preferred for its simplicity. Multi-point crossover was the preferred choice for the recombination operator, as Uniform crossover was believed to be capable of dismantling perfectly fit genes, thus rendering it useless in the next generation. It is unclear whether 0.1% or 0.01% was chosen for the mutation probability, however it is assumed that one of these was chosen for implementation. Elitism is implemented in the GA by preserving the best individual in each iteration, and ensuring that it is propagated to the following generation if the current generation fails to yield a better solution. No detailed mention is made of the selected objective function in this particular paper. The GA algorithm was demonstrated to have a visually better step response than the Ziegler-Nichols method.

In an article by Mahfoud et al. (2021), the GA is applied to tune the PID controller for the Direct Torque Control of a Doubly Fed Induction Motor (DFIM). The GA used was

encoded using real numbers, as this appeared to be the most efficient, and reasonably simple to implement. The first population was generated heuristically to ensure that the algorithm evolves 'good' genes. Maximum values of K_P, K_I and K_D were set to 100, 10 and 1 respectively and all parameters had a minimum value of 0. It was noted that research discovered optimal population sizes between 10 and 160, and that 20 was chosen for initial implementation. Equations 2.12, 2.13 and the weighted function in Equation 2.16 was used, and the performance of each is compared. The objective functions are each converted to the fitness function given in Equation 2.9. The Tournament Selection method was used for parent selection, as it yielded the best results through experimentation. The probability of crossover is suggested to be within range [0.6, 0.99], and was therefore chosen as 0.8. The probability of mutation should be between [0.001, 0.01], and was chosen as 0.001. The GA successfully improved response time by 82.67% and rejection time by 72.21%, reduced overshoot and electromagnetic torque ripples by 100% and 16.16% respectively, and minimised Total Harmonic Distortion in the stator and rotor currents by 53.76% and 34.55% respectively.

Alharbi & Gomm (2017) conducted research to optimise PID controllers for a Multivariable (MIMO) Process. In its simplest representation, the control loop is reminiscent to the MIMO model of the TRMS system as depicted in Figure 1.2. A review of the general methodology is worthwhile, as the methods presented could be well applied to the TRMS. To allow the reader to better illustrate the system architecture and follow this analysis better, the Shell oil process model overview is shown in Figure 2.4.

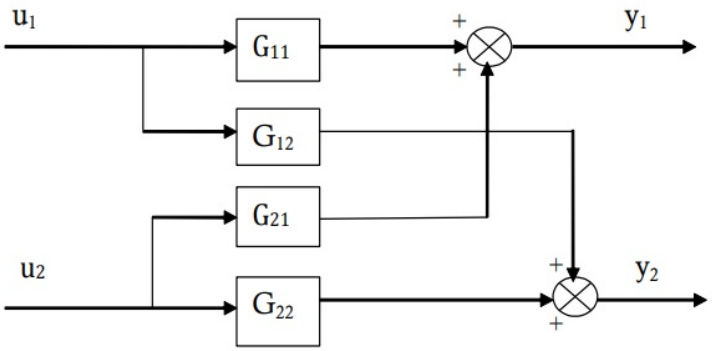

Figure 2.4: Shell process oil system overview (Alharbi & Gomm 2017)

It may benefit the reader for the sake of contextual application to this project, to relate G_{11} to the Main Path for Pitch, G_{12} to the Cross Path from Pitch to Yaw, G_{21} to the Cross Path from Yaw to Pitch and G_{22} to the Main Path for Yaw.

Alharbi & Gomm first optimise controllers for the system plant of each SISO-reduced path: from u_1 to y_1 and from u_2 to y_2, corresponding to the plant transfer functions of G_{11} and G_{22} respectively. To achieve this, the researchers ran optimisation for a PID wrapped around each of these two transfer functions so that the controllers were considered for its own plant individually, without any cross coupling considerations between the two input lines. This was tested both by simulating each path separately, i.e. u_1 to y_1 and u_2 to y_2, and by testing the entire system with cross-couplings present. When testing the entire system, u_1 was first set to a step function of $r(t) = 1$ and u_2 set to 0, then vice versa to test the u_2 to y_2 line. The plots for both outputs were obtained for both scenarios. It must be noted at this point that the objective function chosen was ITAE, as given in Equation 2.14.

Alharbi & Gomm then perform optimisation on each of the two PIDs for the whole-system configuration, as in Figure 2.4. To do this, the researchers first set the parameters of the PID controller for G_{22} equal to the parameters found in the SISO instance. The PID for G_11 is then optimised, using the error term:

$$e_1(t) = y_1(t) - r_1(t) \tag{2.21}$$

in the GA's ITAE objective function. The PID for G_22 is optimised by using the error term:

$$e_2(t) = y_2(t) - r_2(t) \tag{2.22}$$

in the GA's ITAE objective function. The testing steps used to obtain the step response results are the same as detailed for the SISO-optimised case.

Most importantly to review, however, is the novel calculation of the objective function using both the error terms $e_1(t)$ and $e_2(t)$, so that the objective function becomes:

$$ITAE = \int_0^\infty t \; |e_1(t) \; + \; e_2(t)| \; dt \tag{2.23}$$

This allows the simultaneous optimisation of ensuring that the path supplied with $r(t) = 1$ follows a unit step response, and the other path, with $r(t) = 0$, remains at 0. Inspection

of the ITAE function and the corresponding error terms can verify this; in the situation where $r_1(t) = 1$ and $r_2(t) = 0$, if $e_1(t)$ and $e_2(t)$ are minimised, the response will tend towards a step response for $y_1(t)$, and will tend toward 0 for $y_2(t)$. The process for optimisation is the same as the previous method that the researchers conducted. This method of optimisation performed comparatively better than the other methods that were conducted by the researchers, demonstrating improvements in most of the standard performance criteria such as rise time, overshoot and settling time, whilst also achieving the lowest ITAE value.

More sophisticated GAs have also been applied in research. An article by Lin & Liu (2010) focuses on tuning the parameters of a PID controller using an adaptive GA. The Adaptive Genetic Algorithm (AGA) used is different to a classical genetic algorithm in that the rate of mutation (P_m) and crossover (P_c) are automatically increased when the fitness of individuals in the population indicates a convergence toward a local optimum, and are decreased when the population fitness tends toward dispersive characteristics. The mutation (Equation 2.25) and crossover (Equation 2.24) probability are adjusted with the following equations:

$$P_c = \begin{cases} p_{c1} & f' < f_{avg} \\ p_{c1} - \frac{(p_{c1}-p_{c2})(f'-f_{avg})}{f_{max}-f_{avg}} & f' \geq f_{avg} \end{cases} \tag{2.24}$$

$$P_m = \begin{cases} p_{m1} & f < f_{avg} \\ p_{c1} - \frac{(p_{m1}-p_{m2})(f_{max}-f_{avg})}{f_{max}-f_{avg}} & f \geq f_{avg} \end{cases} \tag{2.25}$$

The fixed maximal cross probability is p_{c1}, fixed minimum crossover probability is p_{c2}, fixed maximum mutation probability is p_{m1}, fixed minimum mutation probability is p_{m2}, f_{max} and f_{avg} are maximal fitness and average fitness of population respectively, f' is the fitter of the individuals to be crossed and f is the fitness of the chromosome to be mutated. In the work previously examined by Mahfoud et al. (2021), the crossover probability interval of [0.6, 0.99] and mutation probability interval of [0.001, 0.01] could be assigned to the respective minimum and maximum crossover and mutation values, if an adaptive algorithm were to be designed.

Lin & Liu also clearly outline the following GA configuration of PID parameter bounds: 0 to 20 and 0 to 1 respectively, chromosome encoding method: binary number encoding,

population size: 50, generation size: 100, crossover rate: 0.6, mutation: 0.06, reproduction method: roulette wheel, crossover: single point crossover, mutation: single point mutation. The objective function used is a weighted version of the ITAE, equation, such that:

$$J = \int_0^\infty (w_1|e(t)| + w_2 u^2(t))\ dt + w_3 \cdot t_u \tag{2.26}$$

where w_1, w_2 and w_3 are the weights, $e(t)$, as defined in Equation 2.15 is the error between the reference setpoint signal and the system output, $u(t)$ in this context is the controller effort (output) and t_u is rise time. The penalty function in Equation 2.27 is also used to reduce overshoot when the error is less than 0:

$$J = \int_0^\infty (w_1|e(t)| + w_2 u^2(t) + w_4|e(t)|)\ dt + w_3 \cdot t_u \tag{2.27}$$

Where w_4 is the weight for the added penalty term. The fitness function is as defined in Equation 2.20.

The AGA method applied shows good results when compared to the Ziegler-Nichols and Classical Genetic Algorithm (CGA) methods. The authors also claim that the simple structure and low computational complexity make the algorithm suitable for online adaptation.

The Real-coded GA (RGA) is used to tune the PID controller for the TRMS, using predetermined search range, in a paper by Prasad et al. (2013). To determine the initial search, Simulink's Design Optimisation Library CDO tool was attached to the signal to be optimised, and desired step response characteristics were specified. The gradient descent method was then chosen as the optimisation method. The CDO was then run to provide an initial range for the PID parameters.

To run the RGA process, 12 chromosomes with three genes each were used for 1-DOF systems and 20 chromosomes with 12 genes each for the 2-DOF system. Roulette wheel selection was used, where the probability for selection for each chromosome is:

$$P_i = \frac{\xi_i}{\sum_{t=1}^{\lambda} \xi_t} \tag{2.28}$$

where ξ is the fitness of the individual chromosome. Crossover method used is whole arithmetic recombination, using the following two equations:

$$\begin{cases} x_i = (1-\beta)a_i + \beta b_i \\ y_i = (1-\beta)b_i + \beta a_i \end{cases} \tag{2.29}$$

where $A = [a_1, a_2, ..., a_i]$ and $B = [b_1, b_2, ..., b_i]$ are the resultant offspring as a result of the cross over, and β is a random number in the range [0, 1]. It should be noted that $\beta = 0.5$ will produce identical offspring. For mutation, the randomly selected genes are modified so that they take the value determined by Equation 2.30

$$x_k = L_k + r(U_k - L_k) \tag{2.30}$$

where L_k U_k are lower and upper bounds respectively of the values that a gene can take, and r is from [0,1]. The objective function used was the integral of absolute error and squared control energy, or ITE, as given in Equation 2.31

$$I_{ITE} = (T^2 \int_0^T |e(t)| \, dt) + \int_0^T u^2(t) \, dt \tag{2.31}$$

where T is the simulation time. The fitness function used was:

$$fitness = \frac{30000}{I_{ITE}} \tag{2.32}$$

The value in the numerator is explained in a work referenced later in this section. Results showed better performance as compared to the CDO optimisation.

A similar strategy is seen in the work by Juang et al. (2008), in that to determine the initial search range, the Nonlinear Design Technique (NCD) was used. This yielded initial values of approximately 1 for all PID controller parameter gains, therefore, the search range was initially set to [0, 2]. A comprehensive, modified ITSE objective function was

used for control over multiple performance indices:

$$
\begin{aligned}
I_{PITSE} = T^2 \Bigg(& \int_{t_1}^{t_2} (y(t) - 1.1r(t))\ dt\big|_{M_p\ \geq\ y(t)\ \geq\ 1.1r(t)} \\
& + 2\int_{t_3}^{t_4} (y(t) - 1.5r(t))\ dt\big|_{M_p\ \geq\ y(t)\ \geq\ 1.5r(t)} \\
& + \int_{0}^{t_r} 0.9r(t)\ dt\big|_{0.9r(t)\ \geq\ y(t)} \\
& + \int_{t_{ss}}^{T} |y(t) - r(t)|\ dt\big|_{t\ \geq\ steady-state\ time,\ |e(t)|\leq\ 0.05r(t)} \Bigg) \\
& + \int_{0}^{T} u^2(t)\ dt
\end{aligned}
\tag{2.33}
$$

where T is the total running time, t_1 to t_2 is the time period when $y(t) > 1.1r(t)$, t_3 to t_4 is the time period when $y(t) > 1.5r(t)$, t_r is rise time, t_{ss} is steady-state time and M_p is the maximum overshoot of $y(t)$. The fitness function was chosen to be:

$$
fitness = \frac{10000}{I_{PITSE}} \tag{2.34}
$$

The numerator of this equation, often referred to as A, is generally chosen to be a large constant, which reportedly enhances the differences between chromosomes. For the GA, population size was set to 10, number of generations was 200, simulation time was from 0 to 50 seconds and sampling time was 0.05 seconds. Roulette wheel selection was used for Selection and Reproduction, mutation is achieved through using Equation 2.30 with a rate of 0.025, and crossover was applied by using a modified crossover process with a rate of 0.025. This crossover method is implemented in this project, therefore implementation is covered in Chapter 3.

The optimisation method shows to reduce error by 21% in the horizontal plane and 20% in the vertical plane for the 2DOF test, as compared to conventional RGA optimisation. Juang also contributed to another paper (Juang & tu 2013), which compares a similar implementation of the GA with other optimisation methods. In this work, algorithms were implemented on a Field Programmable Gate Array (FPGA), to achieve hardware-in-the-loop testing.

A paper by Sivadasan & Iruthayarajan (2018) focuses on applying different evolutionary algortithms (including the RGA) to the TRMS system, using a Nonlinear PID controller.

Both coupling ignored (2 PID controllers) and coupling considered (2 PID controllers + 2 cross coupling controllers) cases are investigated in this paper. The following objective function - a variant of ISE - is used:

$$ISE = \int_0^\infty e_m^2 \, dt + \int_0^\infty e_t^2 \, dt \tag{2.35}$$

which considers both the error signals of the main and tail rotor positions, where e_m is the error of main rotor position, and e_t is error of tail rotor position. 20 independent trials were conducted in this paper to draw useful conclusions of the performance of each algorithm. It was concluded that PSO performs best, with the best fitness and convergence characteristics.

An application of the Genetic Algorithm is also seen used to tune a Model Predictive Controller (MPC) for the TRMS (Kumar & Narayan 2016). The objective function strategy revolves around simultaneously optimising the infinity norm of sensitivity function and the ISE objective function.

Literature also demonstrates that the GA is suitable for system identification of the TRMS, using real-coded genes (Toha & Tokhi 2009).

To conclude a rather exhaustive section, it can be seen that there are many applications of the GA within existing literature. It is beneficial to be familiar with the existing research, to build a solid foundation of understanding and methodology for developing GAs, and to better apply methods to the development of new algorithms, such as the Memetic Algorithm (MA), as introduced in the following section.

2.4.3 Memetic Algorithms

A crude, but suitably simple definition of an MA, is that it is an Evolutionary Algorithm, such as a Genetic Algorithm, with an added Local Search (LS) component. The representation of the general flowchart of a GA in Figure 2.3 can be extrapolated to include the LS component, as shown in Figure 2.5.

As with the core philosophy of the GA, the MA also finds its inspiration from evolutionary theory. Memes describe cultural knowledge and ideas which often propagate and mutate

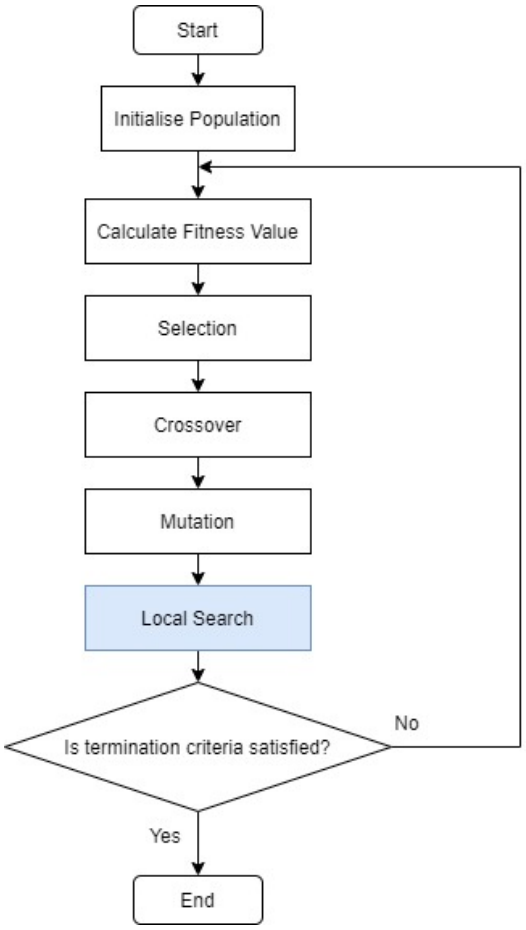

Figure 2.5: Basic General Flowchart of a Memetic Algorithm (Assiroj et al. 2021)

within generations of the human species. Within these cultures, strong ideas are often refined and preserved, whilst weaker ideas dissipate quickly. The local search process in a MA is then analogous to this example, in that the fittest individuals are refined and propagated to future generations, whereas weaker individuals disappear quickly (Neri & Cotta 2012).

For the sake of brevity, the mathematical foundation for MA is not included here; it is anticipated that the information presented in this paper regarding MAs will be accessible to an audience with no prior knowledge of the topic. However, if the reader is interested in acquiring solid foundational knowledge, the article titled "A Gentle Introduction to Memetic Algorithms" by Moscato & Cotta (2003) provides good background and understanding for the key elements of LS strategy. Aside from the introduction to MA, the chapter also includes guidelines for the genetic operators.

For instance, the authors elaborate on the recombination operator and provide guidelines in considering an appropriate strategy. Blind recombination, which does not consider

individual fitness, is usually justified as the choice operator as it supposedly prevents sub-optimal convergence through the introduction of excessive bias in the search algorithm. However, hybrid/heuristic recombination operators are also presented, which are monotonic in nature (where the child generated is at least as good as the best parent), such as Dynastically Optimal Recombination, Patching-by-forma-completion, and a specialised operator for the Travelling Salesman Problem - the Edge Assembly Crossover. An important conclusion of this particular investigation is that blind crossover follows $O(N \log N)$ time complexity, where N is the size of the input. Heuristic recombination is more expensive, but the solutions provided are usually better than the former, thus requiring to be invoked a lesser number of times. The authors also make mention of random seeding or injection of high-quality configurations to generate the initial population.

Another important consideration in MAs is Lamarckian Evolution versus the Baldwin Effect. Both are strategies which describe the behaviour of improved individual(s) when reintegrating with the population. In Lamarckian Evolution, the improved individuals replace the original solutions that were selected for improvement, whereas the Baldwin Effect describes the improved individuals joining the population alongside the individuals chosen for improvement. Both strategies can present issues; for example, Lamarckian Evolution can reduce the diversity of the population, leading to premature convergence. To counter this, a percentage of the total improved individuals may be selected to replace the original individuals, however, this in itself becomes another heuristic which must be fine-tuned for the efficacy of the algorithm (Bereta 2019). In a paper by Krasnogor & Smith (2005), the authors advocate the Lamarckian approach for running the local search to an optimum. Coarse-grain schedulers are suggested as a simple means of avoiding the loss of diversity caused by the Lamarckian Evolution approach, through monitoring and application of vigorous mutation to the whole population. It must be noted that these two strategies may also be combined, and have been done so in other research, with some notable benefits (Choi & Moon 2005).

There are a plethora of different metaheurstics available for performing the LS operation. A paper by Thainiam (2019) investigates four of the most common: Hill Climbing (HC), Tabu Search (TS), Simulated Annealing (SA) and Iterated Local Search (ILS). HC stochastically iterates through neighbour solutions of a selected individual, and only accepts the neighbour if it is a better solution than the original candidate. In TS, neighbours are also visited as in HC, but with two specific alterations; the first being that

individuals with worse fitness can be accepted if no improved individuals are available, to avoid stalling in strict local minima, and the second is that the search is prohibited from visiting previously-searched solutions, which is achieved by utilising a memory structure. SA considers neighbouring solutions, but uses probability to decide if the new solution should be selected. Ultimately, the probabilistic nature of such a system should move to a lower energy state (meaning the search becomes less stochastic), in which the optimum should be located. The final method considered, ILS, is essentially an extension of the Multi-Restart LS, in that a chosen LS method (in this paper it is HC) is performed iteratively, but upon restart, the solutions to improve are not random, but rather are variants of the previous locally optimal solutions, modified through perturbation.

2.4.4 Memetic Algorithms and Control

Whilst the MA enjoys similar popularity to the GA for applications in many areas of optimisation, only a small handful of articles can be found that investigate its application to control systems. Four articles are presented here; the first two apply MA for nonlinear system identification, and the remaining two use MA for determining the gains of a PID controller.

In the first article (Maliński & Figwer 2015), the authors focus on nonlinear input-output dynamic system identification using a MA for both very small and large numbers of input and output measurements. The authors follow common memetic frameworks for implementing the MA. In the initial stage, variables such as population size and standard deviation for mutation are set, and procedures are implemented to adjust adaptive parameters when the search stagnates. The population is then processed by selecting parents through Roulette Wheel Selection (RWS), performing crossover and mutation (if required – this in itself was made adaptive by the authors), and then improvement through local search. The algorithm tracks the best performing individual, which the authors label the 'Current Leader'. The individual is kept separate from the population to ensure that it is propagated to future generations. Crossover was chosen to be optional, and used only on user demand. When crossover is enabled, the operation involves choosing two parents, and then performing a one-dimensional memetic algorithm search to find a global optimum along a line formed between the two parents. The mutation method involves randomly selecting individuals using Gaussian distribution with a mean of 0 and a user

defined standard deviation. The rate of local search is also chosen to be adaptive, and the local search method used is random hill-descension.

The next article by Subudhi & Jena (2009) covers an application of nonlinear system identification using MA to the TRMS. The authors exploit three different global search methods; GA, PSO and Differential Evolution (DE) for comparison and utilises a Back Propagation (BP) gradient descent method for the local search component.

The third article investigates using the MA to design an optimal PID controller for a servo-motor system (Shyr et al. 2002). The authors perform the following steps: initialise a population of integers, use roulette wheel selection to select pairs for mating, apply mutation and crossover to produce new individuals for the next generation, apply a hill-climber algorithm for the local search to refine each individual, then terminate when a solution cannot be improved upon within the nominated number of iterations. The authors show that the MA converges faster, and to a better minimum than a Genetic Algorithm that was used for comparison.

In the final article, a MA is used to tune a PID controller for an AVR system, where the algorithm is DE-based (Mandal et al. 2011). The authors use a custom objective function defined as in Equation 2.36

$$\min f(\vec{K}) = e^{-\alpha} \cdot (t_s + t_r) + (1 - e^{-\alpha}) \cdot (M_p + E_{ss}) \tag{2.36}$$

where the control parameters $\vec{K} = [K_p, K_i, K_d]$ and α is a weighting factor, designed such that a value of $\alpha < 0.7$ allows the designer to preference a minimisation of rise time and settling time, and $\alpha > 0.7$ preferences a reduction of overshoot and steady state error. Each individual is also evaluated against the Routh-Hurwitz stability criterion, in which all values that do not pass are penalised with a very large positive constant value.

The competitive variant of DE that the authors employ is combined with a Hybrid Mutation Strategy, which the authors claim reduces the likelihood of converging to local optima, rather than the global optimum.

For LS, the authors use Solis and Wet's algorithm, which could be categorised as a randomised hill climber. A derivative of this algorithm is implemented in this project, thus implementation details will be presented in the following chapters.

Chapter 3

Methodology

3.1 Chapter Overview

This chapter aims to outline the research problem and the components required to fulfil the requirements of the project. These sections will provide the software design and decisions made for implementation, and will highlight any limitations observed throughout the process.

3.2 The Research Problem

As indicated from the literature review, not many papers have investigated the applicability of the MA to control systems, and a knowledge gap has been identified in that the MA has not yet been used to optimise the PID controllers for control and decoupling of the TRMS. It is of particular interest to test if the MA is capable of optimising parameters to the extent that sufficient decoupling is achievable, removing the need for the 2 additional cross-coupling controllers.

Thus, this research aims to implement the MA to identify and analyse its suitability for efficiently and successfully finding optimal PID parameters to both control and decouple the system using only two PID controllers. The algorithm is analysed to determine if the results are at least as good as, but ideally better, than existing GA implementations that do not utilise LS methods.

The key performance metrics that are analysed in this research are:

1. Total algorithm execution time
2. Sum of total absolute error between desired input setpoints and output response. This should be minimised.
3. Sum of total absolute control effort. This should be minimised.
4. Key step response performance metrics (Doğruer & Tan 2019)
   (a) Percentage overshoot
   (b) Rise Time
   (c) Settling Time

Items 2 and 3 can be effectively analysed by comparing fitness, as this is inversely proportional to the objective function. Therefore, the objective function will be a controlled variable in both SISO and MIMO cases separately.

3.3 The Optimisation Process

This project follows a similar process that was employed by Alharbi & Gomm (2017) for optimising both SISO (1DOF) and MIMO (2DOF) representations of a multi-process plant, with cross-couplings similar to the TRMS.

The linearised MIMO model can be accessed from the system manufacturer, and is simplified and depicted in Figure 3.1 (Didactic 2021). The following subsections will provide an overview of the general optimisation process to determine PID parameters suitable for linear SISO and MIMO representations of the TRMS.

3.3.1 SISO Optimisation Process

To obtain initial starting parameters for both main (pitch) and tail (yaw) rotor PIDs, optimisation is first performed on a SISO model of each individual pitch and yaw paths. The SISO model for both pitch and yaw is simply the isolated path directly from input to output, and can be extracted from the given MIMO model, as depicted in Figure 3.2.

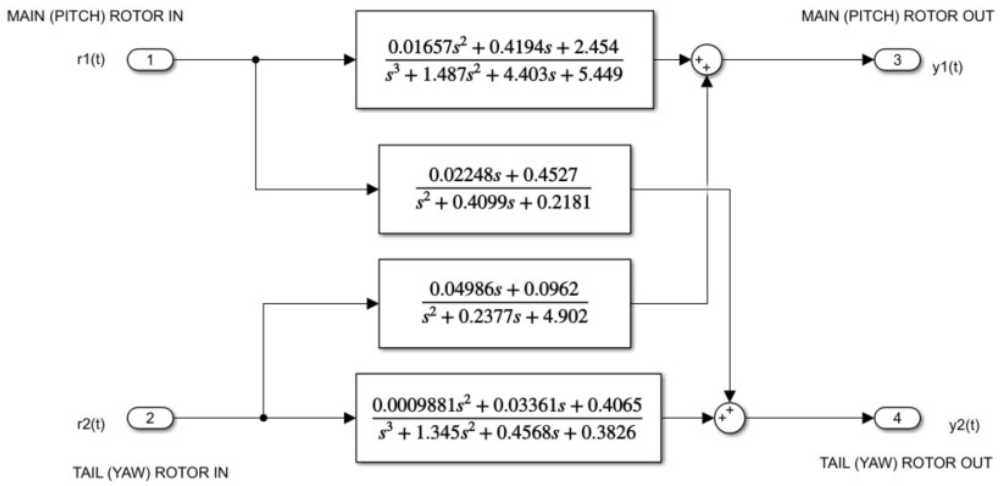

Figure 3.1: Simple Representation of TRMS MIMO System with Transfer Functions

For the SISO model, the PID is cascaded with the plant, and a negative feedback loop is placed from the output to the input, so that the input to the PID controller is the error term of $e(t) = y(t) - r(t)$. The process of building this programmatically is outlined in a following section. Therefore, the system model for both pitch and yaw paths becomes as depicted in Figure 3.3.

The optimisation process is run for both pitch and yaw SISO systems individually to determine good initial starting parameters, before proceeding to the MIMO optimisation process.

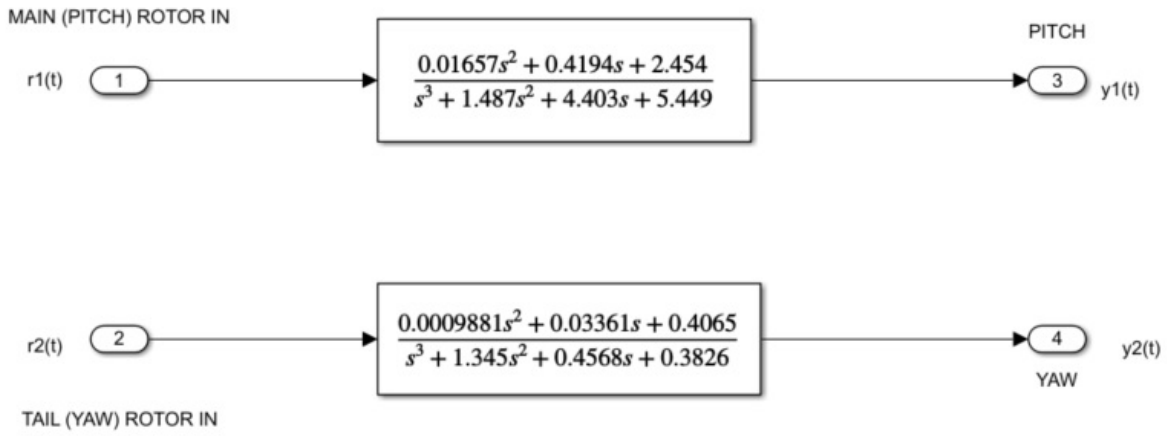

Figure 3.2: Simple Representation of TRMS SISO System with Transfer Functions

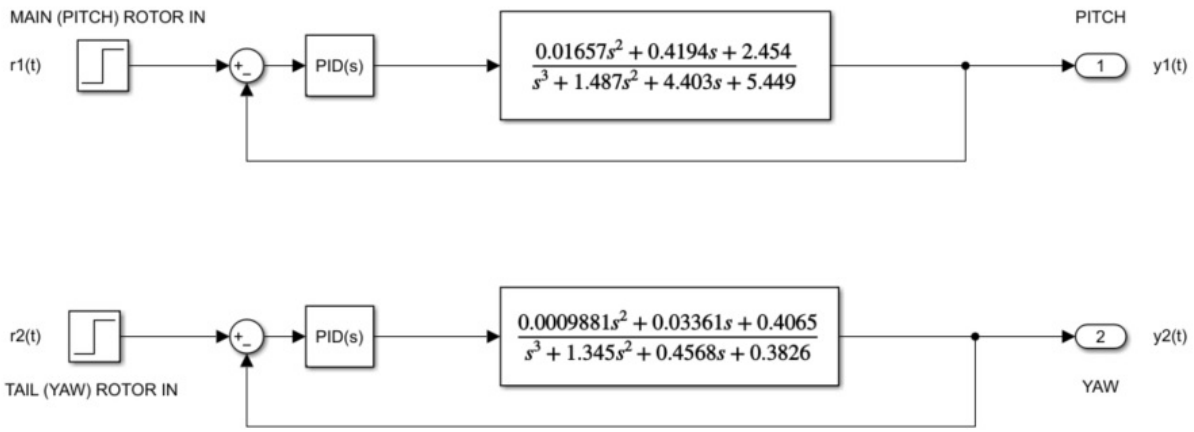

Figure 3.3: TRMS SISO Control System with added PID controllers

3.3.2 MIMO Optimisation Process

With good starting parameters determined from the optimisation of each SISO system, the optimisation of the MIMO system, which takes into consideration the cross-coupling paths, can be performed. The MIMO paths are as depicted in Figure 3.1. Similarly, a PID controller can be cascaded with the main paths for both the pitch and yaw plant representations. However, it is important to note that the PID should be cascaded before the cross-coupling path in each instance; for example, the pitch path PID should be placed before the parallel path split to the pitch path plant and yaw cross-coupling plant. The reverse process can then be performed for yaw path PID controller.

This approach is implemented so the PID can control the errors of both the main path, and the cross-coupling path simultaneously. This is as depicted in Figure 3.4.

When optimising one path, the input/output errors of both paths are incorporated into the objective function. The error for the pitch path, having input $r_1(t)$ and output $y_1(t)$ is:

$$e_1(t) = y_1(t) - r_1(t) \tag{3.1}$$

The error for the yaw path, having input $r_2(t)$ and output $y_2(t)$ is:

$$e_2(t) = y_2(t) - r_2(t) \tag{3.2}$$

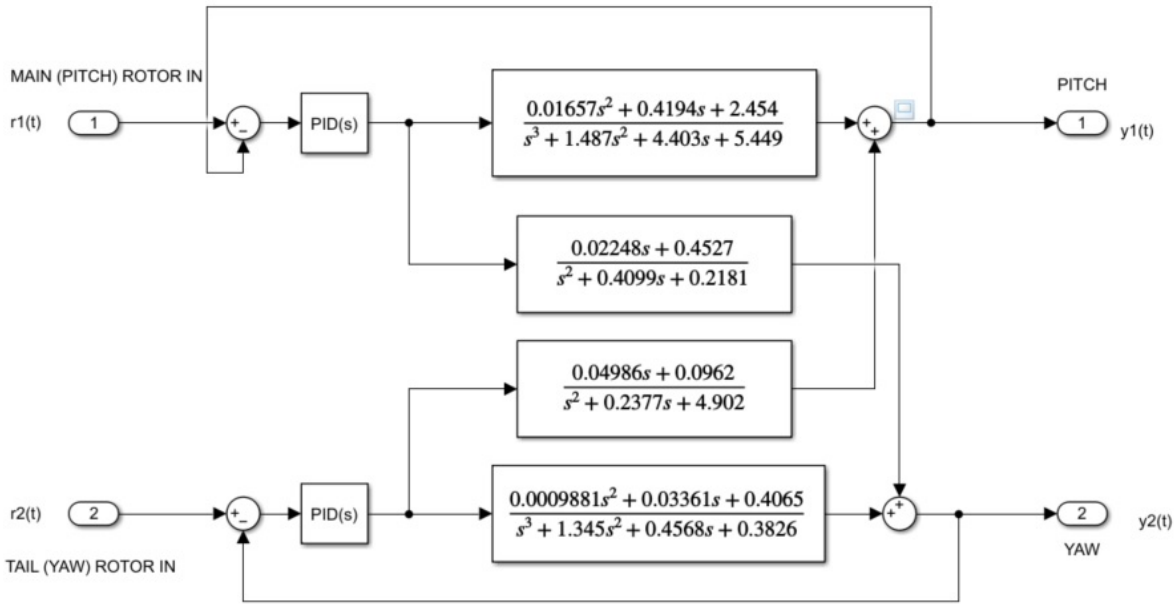

Figure 3.4: TRMS MIMO Control System with added PID controllers

The objective function, as discussed in further detail in a following section, is then tailored to minimise both error functions $e_1(t)$ and $e_2(t)$ simultaneously. This is useful as a means of using two PID controllers to simultaneously control the main path, and decouple the cross-coupling path. This can be achieved by setting the input for the path under optimisation to a step input of $r(t) = 1$, and setting the other input to $r(t) = 0$. If optimisation drives the errors $e_1(t)$ and $e_2(t)$ to 0, then the main path output should be driven towards the expected output of $y(t) = r(t) = 1$, and the cross-coupled path should be driven towards the expected output of $y(t) = r(t) = 0$.

The following is a point form summary of the steps taken for optimising the MIMO representation of the TRMS:

1. Incorporate both errors for both input/output paths in the objective function.
2. Fix the tail rotor PID with the parameters found in the SISO tail rotor PID optimisation
3. Optimise the main rotor PID, setting the main rotor input to $r_1(t) = 1$, and the tail rotor input to $r_2(t) = 0$.
4. Using the results from the previous step, fix the main tail rotor PID parameters.

5. Optimise the tail rotor PID, setting the tail rotor input to $r_2(t) = 1$, and the tail rotor input to $r_1(t) = 0$.

3.4 Memetic Algorithm Software Design

A thorough review of existing literature is very useful in the design of any EA, as many of the configurations and parameters for the algorithm operators rely on heuristic hyperparameters, having values that are often determined solely by pre-existing research. Therefore it must be noted, that for many of the parameters outlined in the following sections, prior research is exploited for reasonable starting values, and these are fine-tuned or adjusted where necessary through experimentation. Thus, this highlights the first limitation of EAs - that many potential parameter values may be used and that many must be chosen *posteriori*.

The GA component of the MA is built using best practices discovered through a literature review of the existing research applied to PID control tuning, and where possible the TRMS test rig. However, there will be some modifications; whilst GA aims toward convergence to a global optimum, most of the parameters for the operators are designed for a balance between exploitation and exploration. In the case of the MA, this should be mainly maintained, however a slight preference toward diversity is desired, as the added local search component itself will search local optima using exploitation and exploration. It is therefore desirable for the GA to diversify candidate solutions (whilst of course maintaining sensible and feasible solutions), and for the MA LS component to intensively focus on optimising elite individuals provided by the GA.

All code created to implement the MA is written in MATLAB, and utilises a few notable toolboxes, such as the Control Systems Toolbox and Parallel Computing Toolbox. If the reader wishes to execute the code in their own MATLAB environment, MATLAB should automatically detect and prompt to install the required toolboxes. If MATLAB fails to detect the missing toolboxes automatically, the reader may need to seek help directly from Mathworks.

3.4.1 Population & Parameter Representation

Parameter Representation

As per many of the research articles outlined in Chapter 2, Section 2.4.2 the PID controller parameters K_P, K_I and K_D are encoded by real-number representation. Thus, each individual in the population is represented as a vector, so that:

$$\vec{K} = [K_P, K_I, K_D] \tag{3.3}$$

where k_1 is the Proportional gain, k_2 is the Integral gain, and k_3 is the Derivative gain. Each PID parameter is referred to individually as the 'Gene'.

This representation is chosen primarily for ease of implementation, and to maintain the highest resolution of values as possible. Often, binary representations are more efficient than real-number encoding, but it must be noted that the performance gains from using such representation is problem-specific, and do not translate to all problem domains. This is discussed further in the work by Mahfoud et al. (2021), and as such, this research follows suit.

Parameter Bounds and Initialisation

Whilst injection of high-quality configurations can be used for generating the initial population (Krasnogor & Smith 2005), the population is generated by random seeding for initial testing. To select suitable bounds for each of the PID parameters, the literature is reviewed to identify three parameter bounds from relevant research on the TRMS, and one from MA research on a foreign system. The last two configurations included in the paper are to test the limits of the bounds configuration. A trial run is conducted with each of the parameter bound configurations, using a generation size of 30 and a population size of 80. The average percentage of stable systems across all generations is recorded and the final fittest value at the end of the algorithm is also recorded. The results are presented in Table 3.1. It should be noted firstly that these values are recorded from one run of the algorithm each, therefore, the values may not be truly representative of a final averaged value across multiple runs, but for the purpose of guiding the selection of values, this is

sufficient.

PID Bounds ([P, I, D])	**Average Percentage of Stable Systems in Generation**	**Fitness**
[0, 4] (Prasad et al. 2013)	99.0833%	3.634
[0, 5] (Sivadasan & Iruthayarajan 2018)	99.2083%	3.7285
[0, 2] (Juang & tu 2013) (Juang et al. 2008)	98.9167%	3.7571
[0, 30] (Shyr et al. 2002)	98.8333%	3.7035
[0, 100]	97.8333%	0.50825
[0, 100], [0, 10], [0, 1]	73.875%	3.6941

Table 3.1: Table of different average percentages of stable systems across all generations and fitness values for different parameter bound settings

The results of this brief experiment demonstrate that more unstable systems are generated, and the population struggles to converge to optimal fitness with larger parameter bounds. Conversely, it is ideal to have reasonably large bounds, as this allows the algorithm to operate over a larger solution space, and more potential solutions may be identified. Most solutions however, from the trial runs, indicate that most optimal parameters have low values, thus the range of [0, 5] has been chosen as bounds for all of the parameter gains.

Population Size

The number of individuals for the population size was determined through partial heuristics. Research suggests that optimal population sizes lie between 10 and 160 individuals. Small population sizes result in premature convergence due to the lack of diversity, and large populations drastically increase computation time (Mahfoud et al. 2021). This has been verified through experimentation in this project, therefore a value of population size = 80 was determined to be a reasonably good value. The parameter *popSize* will be used to denote the population size in the following sections, as it appears in the code.

Generation Size

A review of the collected literature in Chapter 2 is used to determine an appropriate generation size. In the work by Kumar & Narayan (2016), 1000 generations were used to tune a MPC. In tuning PID controllers using GAs, Lin & Liu (2010) and Mirzal et al. (2012) use 100 and 300 generations respectively. A generation size of 30 was used for tuning the parameters of a PID using a MA Shyr et al..

Prasad et al. (2013) and Juang et al. (2008) both performed their work on the TRMS, and chose generation sizes of 200. Considering this value is reasonably average within the sample of papers presented, and the work is focused on the TRMS, this value was initially adopted in the final implementation. However, it was found that the MA regularly converged at approximately 100 generations. It was deemed appropriate to reduce the number of generations, as long as this variable was kept consistent over all test cases. Furthermore, the smaller generation size is beneficial for a reduced general runtime. The final value chosen for the number of generations is 120. The variable *maxGenerations* is used to denote the generation size in the written code, and will therefore be used when discussed in the following sections.

3.4.2 Objective Function and Fitness

Constructing Model for Calculation

The objective function is implemented in code as a MATLAB function named *objective_function()*. The function takes two input parameters: K_arr and testRH. K_arr is an array of the PID parameters to calculate the objective function for, and testRH is a boolean to control whether or not the Routh-Hurwitz criterion is tested. The function returns two values; fitness, which is the calculated fitness value, and stable, a boolean which indicates if the system is stable using the Routh-Hurwitz (RH) criterion.

Before calculating the value of the objective function, a model must be first created by cascading the system plant with the PID constructed by utilising the input K_arr in the Control System Toolbox function *pid*. The plant and PID are cascaded using another Control System Toolbox function *series()*, and the *feedback()* function is used to form a negative feedback loop, so that a complete model of the final system is formed.

To simulate the system with a step response, using the *step()* function, a time vector must be supplied. To construct the time vector, the total time used is $T = 50s$, and a sampling time/resolution of $0.05s$ (Juang & tu 2013)(Juang et al. 2008). With the step response calculated, the control effort $u(t)$ can be obtained from using another toolbox function *lsim()*, by passing the controller as the system to analyse, $1 - y$ - the input to the controller, where y is the output of the system, and the time vector t.

With all variables calculated, y - the output to a step input and u - the controller effort, the objective RH criterion and objective/fitness function can be determined.

A similar process can be followed for the MIMO representation of the system. In this instance, however, care was taken to ensure that the controllers are cascaded before the main and coupling path splits by first programmatically creating a representation of the system as a subsystem with 2 inputs and 2 outputs, and cross-couplings handled internally. The diagrammatic representation of this is depicted in Figure 3.5.

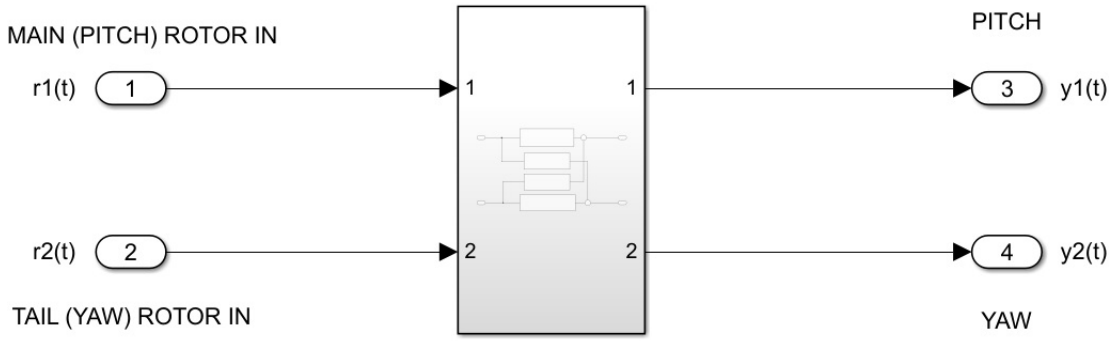

Figure 3.5: Diagrammatic representation of programmatic open-loop subsystem

The MIMO capability of the *series* Control Systems Toolbox function is employed to cascade the output of each PID controller created for the pitch and yaw paths to the respective pitch and yaw inputs of the subsystem. With that completed, negative feedback lines from each output to the input are created by passing a 2x2 identity matrix to the *ss* function; this ensures feedback lines are only formed for the main paths, and not for cross-coupled paths. The final diagrammatic representation of the system formed programmatically can be seen in Figure 3.6.

A helper function was created to build the required SISO or MIMO system, called *build_control_system()*. Source code for the function can be perused in Appendix F.

Routh-Hurwitz Criterion

The RH criterion is an important mechanism which allows a designer to determine if a given system is stable or not. If any roots lie within the right-half complex plane (i.e positive-real), a dynamic system is unstable, therefore, all roots must lie in the open left-half of the complex plane (negative-real) to ensure system stability (Ho et al. 1998). As

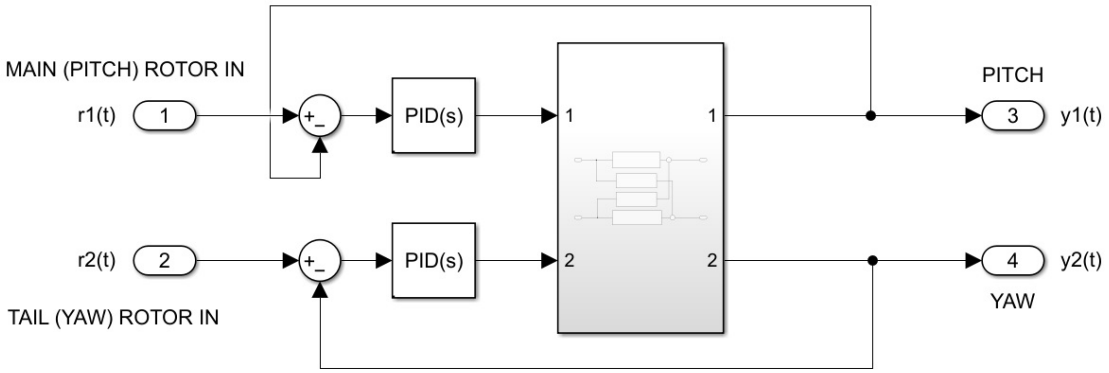

Figure 3.6: Diagrammatic representation of programmatic closed-loop subsystem with PID controllers

Mandal et al. (2011) included the RH criterion in their objective function, it was decided that this criterion should also be included this project's objective function to guarantee that proposed solutions are stable.

Thus, various Routh-Hurwitz algorithms have been constructed to analyse the poles of a given transfer function. Most RH Algorithms are designed to construct the Routh table, in which the values in the first column can be inspected. If all values have the same sign (i.e., when iterating through each value in the first column, the sign does not change from one value to the next), then the RH criterion is passed, and the system will be stable.

To construct the Routh table, a script by Sagharchi (2021) that automatically calculates the stability by constructing the Routh table and checking the signs of the first column, is modified into a function that returns the Routh table only, and the signs are checked within a dedicated function *is_system_stable()*, which may be called as necessary elsewhere in the code. Initially, a function created by Rivera-Santos (2021) was implemented, called *routh()*, which returns the Routh table as a symbolic array. This was discarded, as the function included calculating the determinate of an array, which was measured to be computationally expensive. The *routh()* function appears to handle special cases which may arise for certain system dynamics that may occur, and it is unclear whether the function by Sagharchi (2021) is capable of handling such special cases. Both functions have received high reviews in the MATLAB File Exchange, therefore the computationally expensive *routh()* function was discarded and the modified function of the script by Sagharchi (2021) was kept for implementation. Results have shown no exceptions or errors being thrown by the use of the *rhStabilityCriterion()* function through experimentation

and testing.

The Objective Function & Fitness Function

As can be observed from the Literature Review chapter, there are many objective functions implemented across different research domains. Many objective functions possess different treatments of the error term, whilst others take a weighted approach to performance criteria such as rise time, settling time and overshoot. Different weighted approaches were implemented and tested, but these contributed to greater inefficiency in the calculation of the objective function, primarily due to the computational complexity in obtaining step information (through either standard calculations or using SIMULINK's *stepinfo()* function). It was deemed inappropriate to add further computational overhead, when programmatically constructing a model, analysing, and using response information is already computationally expensive when multiple calls are made to the objective function. Weighted approaches are also subject to being yet another heuristic, in which further tuning or prior knowledge is required.

Since one of the objectives of this project is to optimise for efficient controllers, the chosen objective function is the ITE, as given by Prasad et al. (2013). This appeared to be one of the only functions in which both errors and control effort are considered in the existing literature, without the use of weighting coefficients. Another point of merit for choosing the ITE objective function, is that the authors were performing their research specifically on the TRMS. The objective function equation was presented in the Literature Review chapter, but is included again here for convenience:

$$J = (T^2 \int_0^T |e(t)| \, dt) + \int_0^T u^2(t) \, dt \tag{3.4}$$

In the same paper, the fitness function is calculated as:

$$fitness = \frac{30000}{J} \tag{3.5}$$

This was also used for implementation in this project. The fitness function converts the problem from a minimisation to a maximisation problem, which permits the algorithm

to remain analogous to the 'survival of the fittest' philosophy. Numerically, due to the proportionality between the objective function and the fitness function, the objective function is still ultimately minimised, as the population becomes fitter. The constant scaling factor in the numerator is not critical, and this value was taken directly from the paper. Combined with the objective function, the fitness values yielded are around the range of [1,10], which is easier to discriminate than the low values that would be produced without the numerator scaling factor.

Any values that do not pass the RH criterion have a large number added to the result of the objective function, so that when the fitness function is calculated, the resultant fitness value is drastically lower than the fitness of individuals that pass the RH criterion.

3.4.3 Parent Selection

A large consensus of the Literature Review advocates for the use of Roulette Wheel Selection (RWS) when selecting parents for mating, thus this has been implemented in this work also. RWS ensures that the probability of selecting individuals is proportional to their fitness. The probability of selection is as follows:

$$P_i = \frac{\xi_i}{\sum_{t=1}^{\lambda} \xi_t} \tag{3.6}$$

where the numerator is the individual fitness, and the denominator evaluates to the total fitness of the population.

Parent selection is implemented in the *parent_selection()* function, which takes two inputs; *pop* - the population, and *fitVec* - a vector of fitness values corresponding to each individual in the population. The function returns the selected parent individual.

The code for parent selection is listed in Appendix F.

3.4.4 Crossover (Recombination)

The following modified crossover method as outlined in the paper by Juang et al. (2008), is implemented for this work. The genes in the individual are selected and modified using

the following equation:

$$\begin{cases} x_{o1} = (1-\beta)x_{pm}^{\alpha} + \beta x_{pn}^{\alpha} \\ x_{o2} = (1-\beta)x_{pn}^{\alpha} + \beta x_{pm}^{\alpha} \end{cases} \tag{3.7}$$

where x_{pm} and x_{pn} are genes of two parents respectively, α is the crossover position, randomly selected from range $[1, \mu]$, μ is the number of genes in an individual, and β is random number within range [0,1]. Crossover is performed by firstly splitting both parent chromosomes at the randomly chosen index. The two child chromosomes each inherit the same genes from their respective parents up to, but not including the specified index. The genes for both children at the specified random index are calculated using x_{o1} and x_{o2}, in the given equation above. All genes after the index are then the swapped values of both parents.

Crossover is implemented in the *crossover()* function, which takes three inputs; *pop* - the population, *fitVec* - a vector of fitness values corresponding to each individual in the population, and *crossover_rate* - a ratio value between [0, 1] to calculate the number of total offspring produced. To elaborate on the third input further, a crossover rate of 0.65 would yield $0.65 \times popSize$, so that if popSize is 80, the number of offspring produced would be a vector with dimensions 52x3. The number of offspring is always at least 2 offspring, The following formula is also used to ensure that the number of offspring is always a multiple of 2:

$$number\ of\ offspring = floor(crossover\ rate \cdot \frac{popSize}{2}) \cdot 2 \tag{3.8}$$

The function returns the offspring vector, and a corresponding vector of fitness values for each of the offspring individuals.

The rate of crossover is another heuristic which has been chosen based on research. Research by Mahfoud et al. (2021) suggests that a rate of crossover should be chosen from the interval [0.6, 0.99], and chooses the average of these two boundaries: 0.8. Therefore, for this research a rate of 0.8 is also used.

The adaptive approach outlined by Lin & Liu (2010) was considered for adjusting the crossover rate as the population tends toward convergence, however due to the addi-

tional complexity and time constraints of this project, this approach was not adopted for implementation. This is a potential area for future research.

The code for crossover is listed in Appendix E.

3.4.5 Mutation

To perform mutation, a number of individuals determined by the mutation rate are chosen at random. For each individual, a gene is chosen at random, and mutated with the following equation (Prasad et al. 2013):

$$x_k = L_k + r(U_k - L_k) \tag{3.9}$$

where L_k U_k are lower and upper bounds respectively of the values that a gene can take, and r is from [0,1].

The rate of mutation is also covered in the paper by Mahfoud et al. (2021). The recommended rate of mutation is in the interval of [0.001, 0.01] and was chosen to be 0.001. In the work by Mirzal et al. (2012), a value of 0.01 is used. The rate of mutation was chosen to be 0.01, as for smaller population sizes, such as the population size of 80 in this project, rates of 0.01 and 0.001 will both result in 1 mutated individual, because $0.01 \times 80 < 1$ and $0.001 \times 80 < 1$, and the minimum number of mutations to occur is configured to always be at least 1 if the rate of mutation is non-zero.

Mutation is implemented in the *mutation()* function, which takes three inputs; *pop* - the population, *crossover_rate* - a ratio value between [0, 1] to be used to calculate the number of total offspring produced, and *gene_bounds* - a cell vector of bounds corresponding to each of the Proportional, Integral and Derivative genes. The function returns both a vector of the offspring and associated fitness values.

The adaptive approach outlined by Lin & Liu (2010) was considered for adjusting the mutation rate as the population tends toward convergence, however due to the additional complexity and time constraints of this project, this approach was not adopted for implementation. This is a potential area for future research.

The code for mutation is listed in Appendix E.

3.4.6 Local Search

Following from the work by Mandal et al. (2011), in which the original Solis & Wet's algorithm is implemented, this project implements the modified Solis & Wet's algorithm by Zhang et al. (2012), albeit with a very minor alteration. In Zhang et al.'s work, the algorithm is implemented in tandem with an electromagnetic-like global optimisation scheme. The Solis & Wet's algorithm is a stochastic hill-climbing algorithm. Before starting the algorithm, the fittest candidate is typically chosen for improvement. A deviation is then chosen from a normal distribution with a mean b_i and standard deviation ρ. The randomly chosen deviation value is added to all genes of the candidate solution. The value of the mean, b_i, is biased in a certain search direction (i.e. the value will become positive or negative) dependent on whether the algorithm finds a better solution. The standard deviation ρ is increased to broaden the search space if the algorithm detects a convergence to local optima, and is decreased to focus the search if the algorithm fails to find better local solutions.

The modified algorithm by Zhang et al. (2012) removes a maximum iteration limit in favour of a global variable *threshold* that shrinks at the end of the LS algorithm, if a better candidate is successfully found. The standard deviation, ρ - another global variable, is likewise increased at the end of the algorithm. Since both variables are global, each time the LS algorithm is run, the current search will have access to the previous search's ρ and *threshold* variables. Both shrinking of the threshold and expansion of the standard deviation, as well as the global capability of both variables, ensures that the local search space becomes wider as the whole MA progresses. As the algorithm iterates while $\rho > threshold$, the shrinking and expansion introduced through this modified algorithm also reduces the effective number of iterations as the MA progresses to convergence.

In the implementation for this project, some small changes were made to the LS algorithm. It was found that the LS would occasionally stall, thus resulting in the exit criteria ($\rho > threshold$) never being met. Thus, the maximum iteration limit was reintroduced and set to 100. Reintroducing the maximum iteration did not negatively impact the LS capability, and indicated potentially better performance (although this may be anecdotal as full verification was not performed).

Neither of the two articles that investigate the MA and PID control provided the rate of Local Search. The modified Solis & Wet's algorithm is typically performed on the best solution. There is evidence in research to suggest that this is more a limit of computing power. In this project, a rate of 0.05 was used so that 4 of the fittest individuals were subject to local search. This specific number was chosen, as the maximum number of cores available on the test machine is 4, therefore the local search can be split between 4 workers, with the 4 local searches taking approximately the same amount of time as performing 1 local search. This alludes to the last modification made to the LS algorithm; the scaling factors of 0.2 and 5 for *threshold* and ρ respectively, are normalised by the number of candidates selected for optimisation. This is done to compensate for the LS being called on more than one individual per generation. All other parameters are used as per the work by Zhang et al. (2012). The parameters are listed here for convenience:

- ρ - standard deviation initial value: 0.01
- *threshold* - termination condition initial value: 0.001
- *maxF* - maximum number of failures for decreasing ρ: 3
- *maxS* - maximum number of successes for increasing ρ: 3
- *conF* - exponential factor for increasing ρ: 2
- *maxF* - contraction factor for decreasing ρ: 0.5

For pseudocode of the algorithm, the reader should refer to the work by Zhang et al. (2012), whilst also noting the small changes made in this project.

3.4.7 Survivor Selection

The crossover, mutation and local search operations introduce additional individuals to the population, such that the population grows greater than the original size. Survivor selection is used to cull the population back to the original desired size. To maintain diversity, rather than select the population size of fittest individuals, tournament selection was used to force *popSize* number of the total individuals to fight for a place in the next generation. This method has been reported to provide good results when compared to other selection processes (Mahfoud et al. 2021).

Tournament selection is a simple method of sample and select, where k individuals are selected to complete from the extended population, and the value with the highest fitness value is selected. This process is repeated *popSize* number of times, so that the final population size is reduced back to *popSize*. Commonly selected values of k are 2, 4 and 7, and for this project a simple binary tournament with $k = 2$ is used. This method of selection is useful, in that it is simple and efficiently implemented in non-parallel and parallel architectures, and has a time complexity of $O(N)$, where $N = popSize$, as sorting of the population is not required (Fang & li 2010).

The survivor selection also permits the preservation of elite individuals - commonly referred to in literature as elitism. Prior to performing any of the operator methods, the elite individuals are preserved so that they are guaranteed a place in the next generation. Thus, N number of elite individuals are selected and propagated directly to the next generation during survivor selection. The number of tournaments held then becomes $popSize - N$. A *get_elites()* helper function is created to assist in retrieving *eliteCount* number of elite individuals. The function takes three inputs; *pop* - the population, *fitVec* - a vector of fitness values for each of the population individuals, and *eliteCount* - the number of elites to select. The function returns both a vector of elite individuals and a vector of corresponding fitness values. According to research by Mishra & Shukla (2017), a lower amount of elite individuals was better for avoiding premature convergence to a local optima, with 0 elites surprisingly giving the best results. However, for this research, the *eliteCount* is set to 1, to introduce the bare minimum amount of elitism, but to also guarantee that at least one elite is propagated throughout each generation.

Survivor selection is implemented through the *surivor_selection()* function, which takes five inputs:

- *extended_pop* - the extended population with the results from all the operators
- *extpop_fitness* - the fitness of each individual in the extended population
- *elites* - the elites to propagate directly to the next generation
- *elites_fitness* - the fitness of the elites
- *pop_size* - the desired population size for the next generation

The function returns a vector of the next generation and a vector of fitness values for

each individual of the next generation.

The code for survivor selection is listed in Appendix E.

3.5 Implementation Notes

3.5.1 Simulation Workstation Specifications

The specifications of the PC used for optimisation and simulation is as listed in Table 3.2

CPU	Intel (R) Core(TM) i7-7700HQ CPU @ 2.80GHz
GPU	NVIDIA GeForce GTX 1050
RAM	16.0 GB
OS	Windows 10 Education
System Type	64-bit operating system, x64-based processor

Table 3.2: Workstation PC Specifications

3.5.2 Parallelism

The code written for this project makes use of the Parallel Computing Toolbox by MATLAB. The toolbox utilises multicore CPUs/GPUs and/or clusters to perform computations in parallel (Mathworks 2021*b*). In this work, the *parfor* and *parfeval* constructions are used primarily to perform expensive computations in parallel, where possible. The reader may peruse the source code attached in the appendix and find where computations are performed in parallel by identifying the *parfor* loops, and the *parfeval* function calls. MATLAB utilises 'workers', which are essentially parallel computational engines in which a scope of code can be sent for execution.

By default, MATLAB uses a number of workers equal to the number of physical CPU cores running a single computational thread. Even though many CPU's make use of hyper-threading (i.e. multiple virtual cores per physical core), some resources which are important for computations, such as Floating Point Units, are shared between virtual cores, with only one physical unit per physical core. Therefore, it is usually optimal to run one worker per physical core. In this project, a parallel pool of 6 workers was

implemented, mainly due to experimental results. 6 workers in nearly all cases performed as well as 4 workers, with occasionally outperforming 4 workers only. Being a 4-core system with 2 threads per core (8 total threads), 8 workers was tested also. However, 8 workers resulted in the PC becoming unstable, therefore, a number of workers above 6 was avoided (Mathworks 2021*f*).

Much of the written code utilises *parfor* loops. These are similar in nature to normal for-loops, except that code inside *parfor* loops are sent to workers for execution. Executions are performed in a nondeterministic order, therefore loop iterations must be consecutively increasing integer values and the body of the loop must not be dependent on previous iterations (Mathworks 2021*d*). For some dependencies, Reduction Variables can be utilised (Mathworks 2021*e*), which was implemented for some cases in this project. Most cases that fit the requirements were implemented as *parfor* loops. In deciding when such loops can be used, the *Decide When to use parfor* guide was consulted (Mathworks 2021*a*). Another limitation of the *parfor* loop worth discussing, is that directly nested *parfor* loops are not supported, with the exception of functions containing *parfor* loops that are called within an outer *parfor* loop. This limitation meant that different runs of the algorithm were required to be conducted sequentially, rather than using a *parfor* loop, as the main code contained *parfor* loops.

Also included in the toolbox is *parfeval*, which can be used to run functions on parallel pool workers (Mathworks 2021*c*). This function was specifically used to perform the mutation, crossover and local search operators in parallel, as well as the *get_elites()* function. Each of these functions/operators are able to be run independent of each other, therefore parallel function evaluation was both sensible and beneficial in these cases. As with *parfor*, the *parfeval* is limited in that functions that are already executing on parallel workers cannot execute nested *parfeval* functions. This is another reason why, as in the example given previously, different runs of the MA cannot be executed in parallel. The client (workstation) is the only environment capable of calling *parfeval*.

The Parallel Computing Toolbox has been integral to ensuring good efficiency of the MA. To assist in identifying bottlenecks and candidates for parallelisation, MATLAB's Profiler tool was utilised. The Profiler tool provides detailed analysis, such as number of calls, total time, self time, and presents a flame graph for visualisation. For examples, the reader is recommended to visit the web page <https://au.mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html>. Through us-

ing this tool, areas of improvement were identified, and in most cases, parallelisation, or other basic fixes, drastically reduced execution time.

3.5.3 Testing and Verification

Due to the stochastic nature of EAs, a run may yield exceptional results, but a non-zero probability exists that reasonably poor results are produced. It is common in literature for many runs of the algorithm to be executed, and then analysed. Due to the computational expense of the GA, and particularly the MA, amassed with tight time constraints, only 5 runs for each case were executed. Therefore, the best individuals for each run are evaluated at the end of execution, and the run whose individual is the median of the set of runs is chosen for analysis.

First, verify the GA produces results that are as good, or better than existing GA research on TRMS. This will verify that the implemented GA is suitable as a testing benchmark for the MA. To do this, compare GA results for Pitch and Yaw 1DOF (SISO) systems to the existing research. If the GA performs equally or better, the GA is a good benchmark, and it can be used with confidence for comparison with the MA results. If not, analyse any shortcomings. MA results may still be compared to the GA testing results, but comparison directly to established research is the most appropriate procedure. To perform comparisons, use step, sine and square inputs to find the absolute error and absolute control force. Step response characteristics should also be compared between GA and MA results. Both Prasad et al. (2013) and Juang et al. (2008) clearly present results in their work on the TRMS. With similar parameters such as sampling time and sampling length, and by using similar evaluation methods, the results in this project can be directly compared. To fulfil all the requirements outlined in Section 3.2, the mean run times of GA and MA SISO tests will also be compared.

Second, test the MA MIMO parameters on the linearised 2DOF system. Most literature utilises 2 main path controllers and 2 controllers for the coupling paths for the TRMS, whereas this research utilises 2 controllers to perform both control and decoupling. Thus, it is appropriate to compare the MA MIMO test results directly to the GA MIMO test results, rather than comparing to results found in the literature. Juang et al. (2008) explores methods for testing the MIMO representation of the TRMS, therefore similar procedures should be implemented for testing the linearised 2DOF system. A step re-

sponse test should be done for each main input/output path, where one input is subject to a step, and the other is set to 0, as discussed in (Prasad et al. 2013) and (Alharbi & Gomm 2017). In a similar manner to the first step, ensure that all key requirements outlined in Section 3.2 are addressed.

Lastly, perform a test of the MA MIMO parameters on the nonlinear TRMS system, which is the closest representation of the real system. Similar methods from the second step may be employed here also. Also, a test of the GA MIMO parameters on the nonlinear TRMS system should be performed for comparison against the MA MIMO results, ensuring to address all key requirements outlined in Section 3.2.

To summarise the steps for testing of results:

1. For both Pitch and Yaw 1DOF Systems:
   (a) Test step, sine and square responses for the PID parameters obtained from GA optimisation. Use the same testing parameters as Prasad et al. (2013) and Juang et al. (2008) so that results can be compared.
   (b) Repeat the process with the PID parameters obtained from MA optimisation.
2. For Linear 2DOF System:
   (a) Use the same MIMO testing parameters and methodology as presented by Juang et al. (2008) to test step, sine and square responses for the parameters obtained from GA optimisation.
   (b) Perform single-path step response testing for both pitch and yaw, similar to the method outlined at the end of Section 6.2 in the work by (2017) Alharbi & Gomm.
   (c) Repeat steps 2.(a) and 2.(b) for the MA optimisation results
3. For Non-linear 1DOF and 2DOF system:
   (a) Repeat step 1 and 2 for the nonlinear system. Nonlinear simulation results are not presented in current literature for the TRMS, therefore the results in this step will be compared to those obtained from the previous steps. Again, in this step, GA-optimised results should be compared to the MA-optimised results.

3.5.4 Hardware Implementation

Due to project time constraints, hardware implementation was not investigated. Results are validated on the nonlinear simulation model as the closest representation of the physical system. Therefore it can be somewhat inferred that the results obtained on such a system will be reasonably representative of physical implementation. Hardware implementation is briefly discussed as part of potential future work in Chapter 5.

3.6 Project Planning

3.6.1 Resource Requirements

Certain resources were required to ensure that this project achieves its objectives. The largest investment required for the project is time. The project magnitude of effort called for approximately 350 hours over the period of February 2021 to November 2021. This is based on the project minimum requirement of 310 hours, and allowed additional time for iterative design and testing, and also factored the potential of further research of more advanced methods. Specific software and equipment are necessary in the successful completion of the project, however most of these are readily available and already acquired or are accessible through USQ. Time was allotted for obtaining pre-requisite knowledge, investigation, implementation, iterative testing, optimisation, dissertation review and writing, and communication with USQ supervisors for any required assistance and feedback/review at project milestones.

Most aspects of the project were able to be conducted outside of USQ, using personal equipment and resources. Due to time constraints, access to USQ laboratories and the 33-949S TRMS model to conduct real-world testing was not included as a requirement for fulfilling the aims of this project. The student has used a PC laptop, replacing the need for utilising USQ PCs.

The planned resources drafted at the start of the project is compiled in Table 3.3.

Item	Quantities	Source	Cost	Comment
PC with Microsoft Windows	1	Student & USQ	Nil	After hours access to students personal laptop. Potential requirement for usage of USQ PC
MATLAB/Simulink	1 full version copy	USQ	Nil	Use for simulating/modelling, and plotting/reporting of results
Microsoft Word	1	USQ	Nil	Word processing of dissertation (primarily note-taking and drafting)
Microsoft Excel	1	USQ	Nil	Data presentation/graphing, project planning
LaTeX software	1	Student	Nil	Word processing of dissertation and presentation
USQ laboratory	1	USQ	Nil	Access to laboratory to perform testing/verification of results
Twin Rotor MIMO System 33-949S	1	USQ	Nil	Used to perform testing/verification of controller

Table 3.3: Table of Required Project Resources

Risk Assessment

The risk assessment is listed in Appendix B.

Chapter 4

Results and Discussion

4.1 Results

4.1.1 SISO Optimisation Results

Pitch Optimisation Results

Genetic Algorithm

Table 4.1 presents the optimisation results of 5 independent runs of the GA pitch optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.1. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.2. Run 4 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 4 is shown in Figure 4.3. The average iteration run time for the GA optimisation on the pitch path was 614.117 seconds.

	Fitness	Rise time (s)	Settling time (s)	Overshoot (%)	System Stable?
Run 1	2.29	3.1198	13.4185	4.0609	Yes
Run 2	0.98586	1.3633	4.3317	9.5049	Yes
Run 3	2.3982	3.0331	21.2792	5.9724	Yes
Run 4	1.6928	2.9446	25.0201	9.5884	Yes
Run 5	1.5722	1.8413	5.1442	6.6710	Yes

Table 4.1: Results for 5 runs of the GA on the Pitch SISO system. The highlighted row is the median performer of the set.

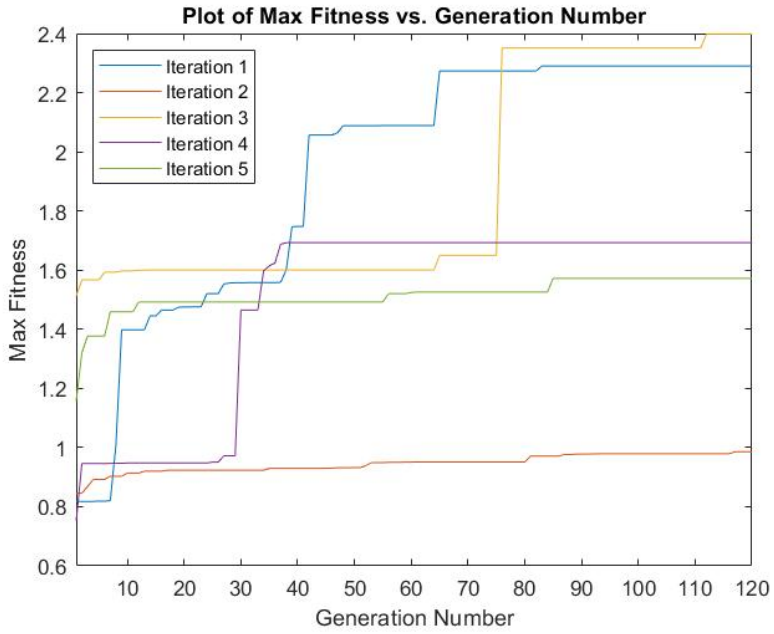

Figure 4.1: Maximum Fitness vs. Generation number for 5 runs of the GA on the Pitch SISO system.

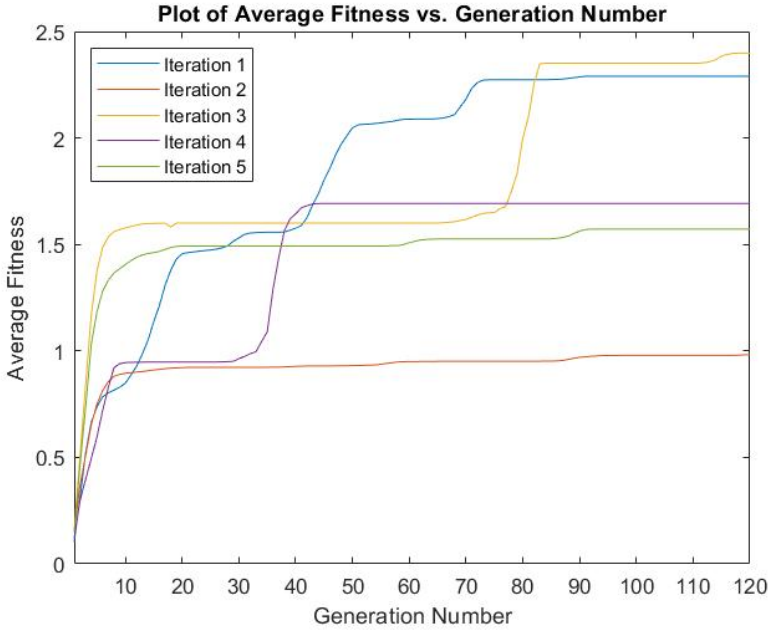

Figure 4.2: Average Fitness vs. Generation number for 5 runs of the GA on the Pitch SISO system.

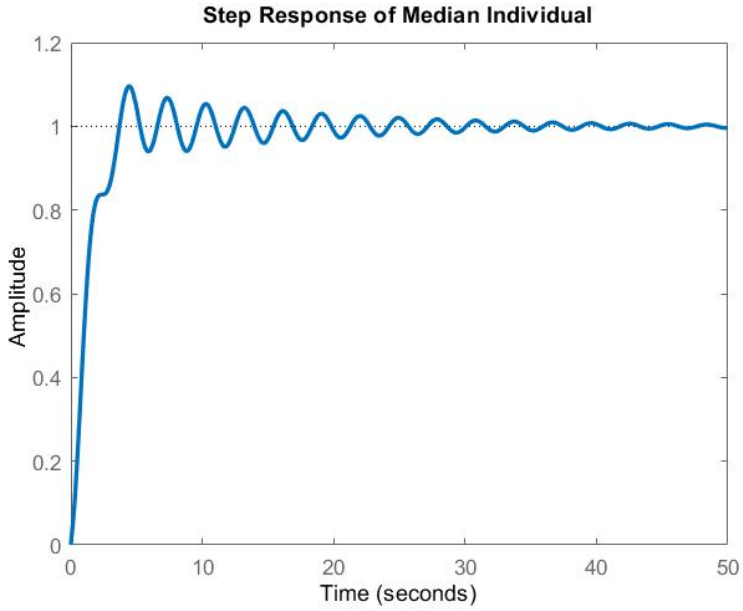

Figure 4.3: Step Response for the fittest individual of run 4 of the GA optimisation on the Pitch SISO system.

Memetic Algorithm

Table 4.2 presents the optimisation results of 5 independent runs of the MA pitch optimisation. The maximum fitness is plotted against generation number for each run of the

algorithm, and is shown in Figure 4.4. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.5. Run 4 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 4 is shown in Figure 4.6. The average iteration run time for the MA optimisation on the pitch path was 5832.9409 seconds.

	Fitness	**Rise time (s)**	**Settling time (s)**	**Overshoot (%)**	**System Stable?**
Run 1	2.853	2.0422	9.1698	4.4870	Yes
Run 2	3.7571	3.9469	12.2246	2.6286	Yes
Run 3	3.7568	3.9260	12.2485	2.6812	Yes
Run 4	3.7246	4.0526	12.0318	2.2590	Yes
Run 5	3.659	4.1152	11.8409	2.0858	Yes

Table 4.2: Results for 5 runs of the MA on the Pitch SISO system. The highlighted row is the median performer of the set.

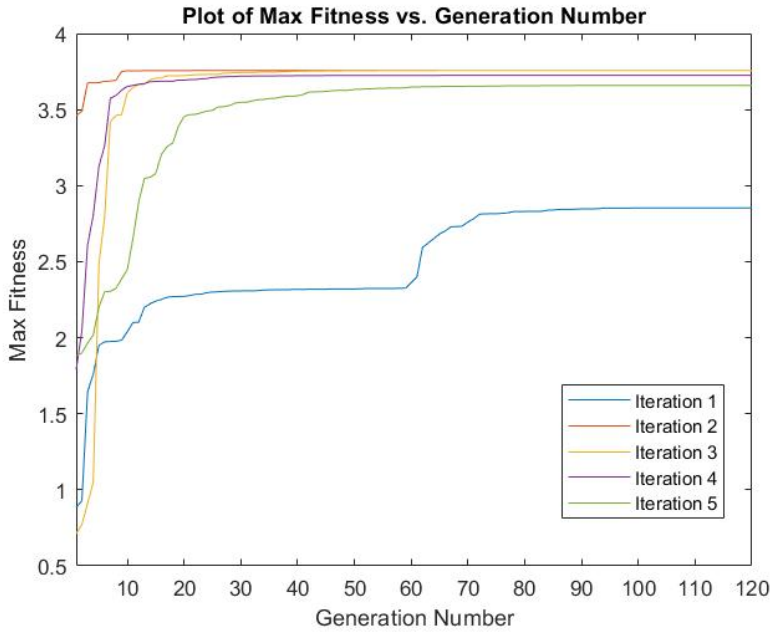

Figure 4.4: Maximum Fitness vs. Generation number for 5 runs of the MA on the Pitch SISO system.

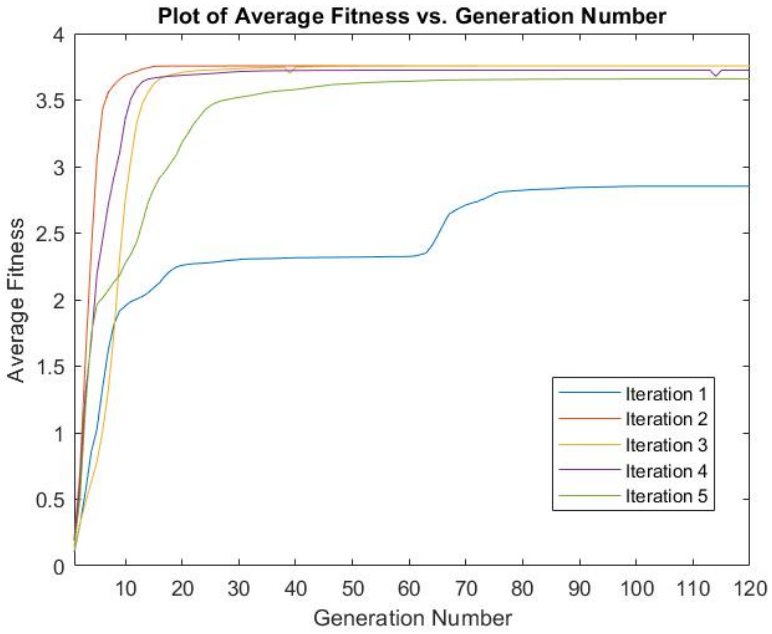

Figure 4.5: Average Fitness vs. Generation number for 5 runs of the MA on the Pitch SISO system.

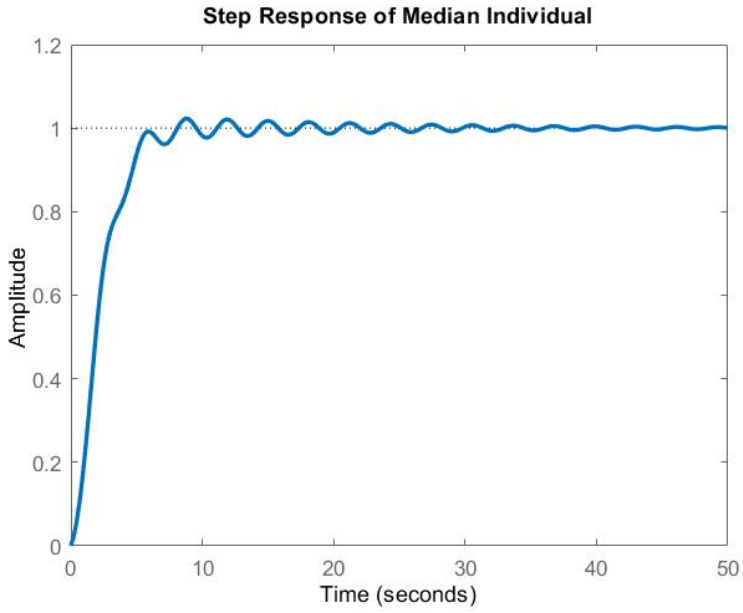

Figure 4.6: Step Response for the fittest individual of run 4 of the GA optimisation on the Pitch SISO system.

Yaw Optimisation Results

Genetic Algorithm

Table 4.3 presents the optimisation results of 5 independent runs of the GA yaw optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.7. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.8. Run 5 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 5 is shown in Figure 4.9. The average iteration run time for the GA optimisation on the yaw path was 620.8791 seconds.

	Fitness	**Rise time (s)**	**Settling time (s)**	**Overshoot (%)**	**System Stable?**
Run 1	0.47471	9.9354	25.3630	5.2779	Yes
Run 2	1.0046	4.3735	105.2105	11.9653	Yes
Run 3	1.1601	5.6345	82.1918	7.9832	Yes
Run 4	0.77853	12.3940	136.4734	4.0860	Yes
Run 5	0.79108	4.8663	40.6602	4.8957	Yes

Table 4.3: Results for 5 runs of the GA on the Yaw SISO system. The highlighted row is the median performer of the set.

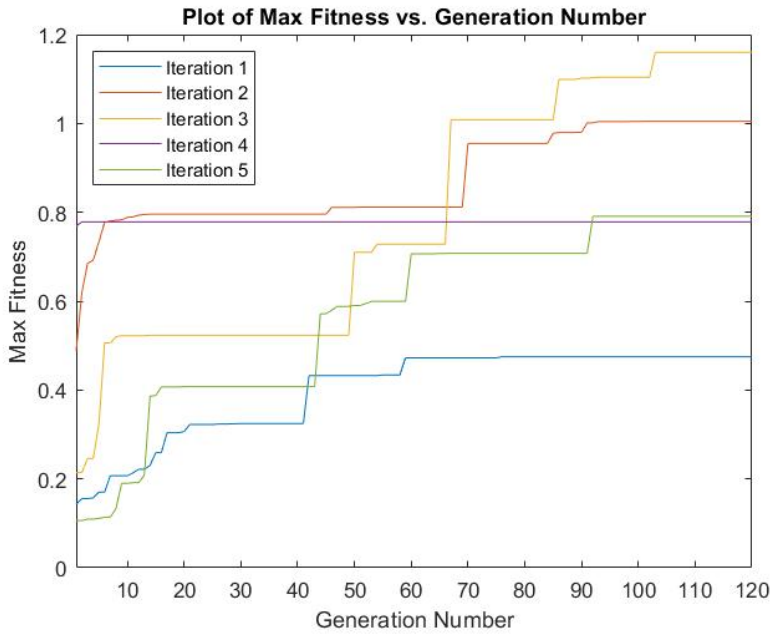

Figure 4.7: Maximum Fitness vs. Generation number for 5 runs of the GA on the Yaw SISO system.

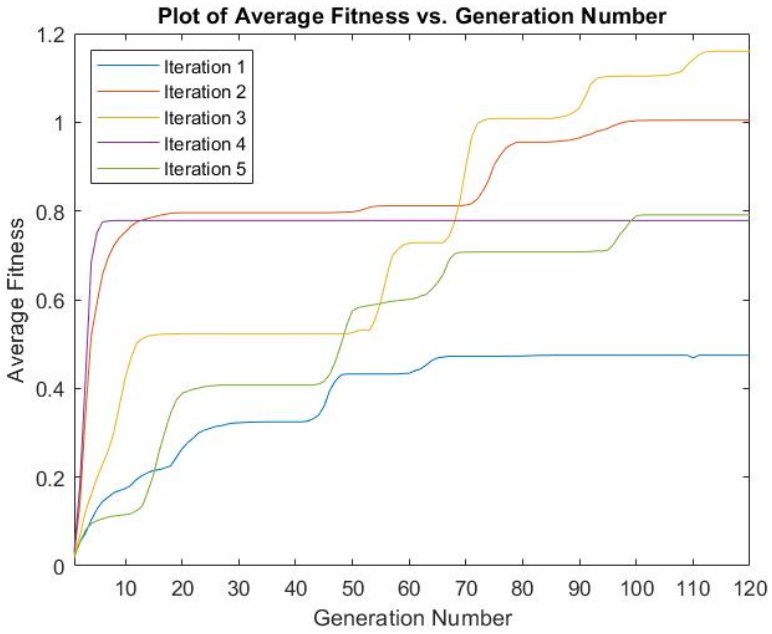

Figure 4.8: Average Fitness vs. Generation number for 5 runs of the GA on the Yaw SISO system.

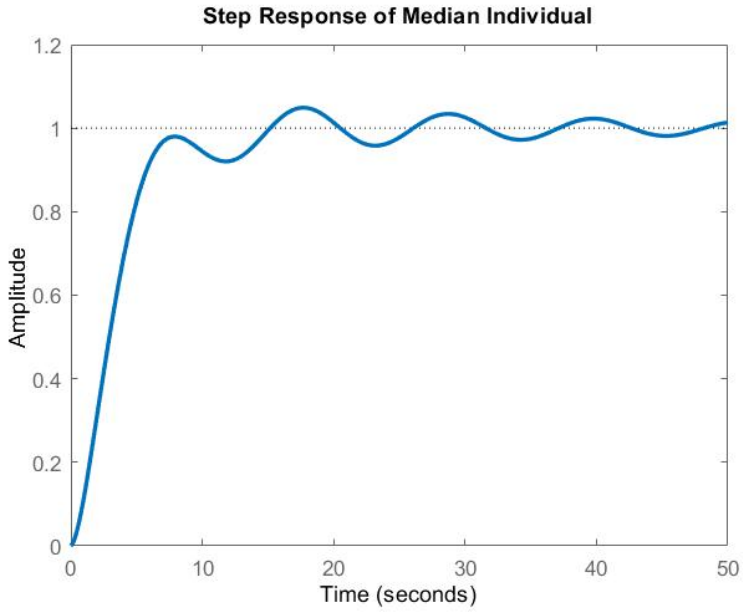

Figure 4.9: Step Response for the fittest individual of run 5 of the GA optimisation on the Yaw SISO system.

Memetic Algorithm

Table 4.4 presents the optimisation results of 5 independent runs of the MA yaw optimisation. The maximum fitness is plotted against generation number for each run of the

algorithm, and is shown in Figure 4.10. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.11. Run 2 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 2 is shown in Figure 4.12. The average iteration run time for the MA optimisation on the yaw path was 2524.0169 seconds.

	Fitness	**Rise time (s)**	**Settling time (s)**	**Overshoot (%)**	**System Stable?**
Run 1	0.61998	11.7298	38.1276	0.2123	Yes
Run 2	1.3593	14.2621	54.3952	4.0569	Yes
Run 3	1.2258	12.7717	35.3986	2.2310	Yes
Run 4	1.1031	12.1574	71.6190	7.9021	Yes
Run 5	0.55239	3.7508	41.4297	8.0107	Yes

Table 4.4: Results for 5 runs of the MA on the Yaw SISO system. The highlighted row is the median performer of the set.

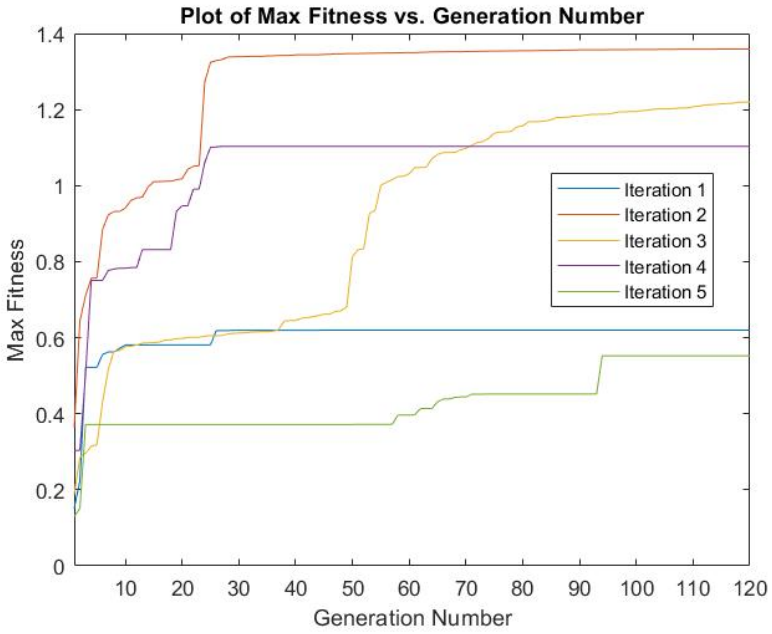

Figure 4.10: Maximum Fitness vs. Generation number for 5 runs of the MA on the Yaw SISO system.

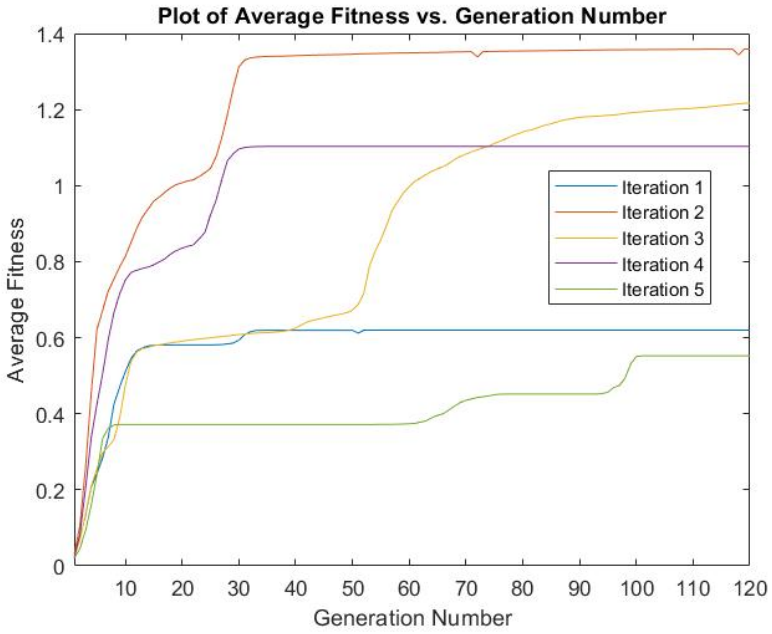

Figure 4.11: Average Fitness vs. Generation number for 5 runs of the MA on the Yaw SISO system.

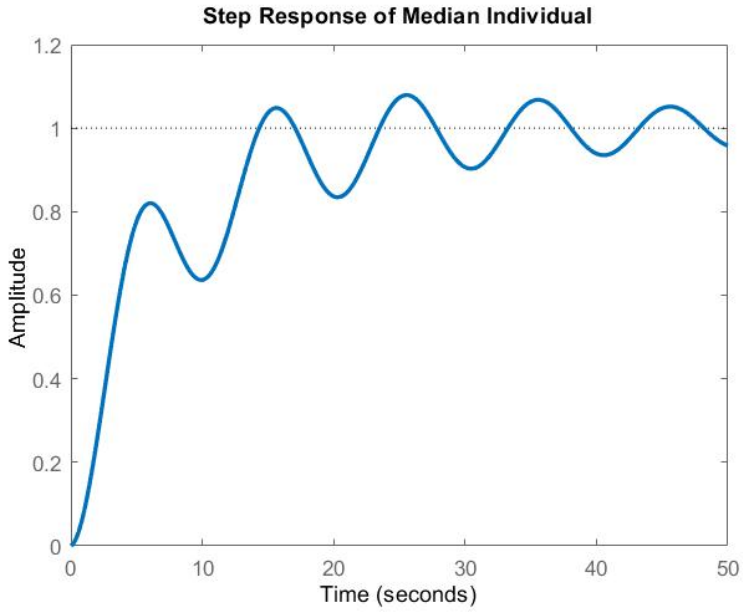

Figure 4.12: Step Response for the fittest individual of run 4 of the MA optimisation on the Yaw SISO system.

4.1.2 MIMO Optimisation Results

Pitch Optimisation Results

Genetic Algorithm

Table 4.5 presents the optimisation results of 5 independent runs of the GA MIMO pitch optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.13. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.14. Run 4 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 4 is shown in Figure 4.15. The average iteration run time for the GA optimisation on the pitch path was 1542.4625 seconds.

	Fitness	**Rise time (s)**	**Settling time (s)**	**Overshoot (%)**	**System Stable?**
Run 1	0.36603	3.5187	16.0637	1.7749	Yes
Run 2	0.4928	6.6353	32.9710	3.0691	Yes
Run 3	0.51776	6.3791	15.4771	1.0894	Yes
Run 4	0.43738	8.8356	19.7101	0.0958	Yes
Run 5	0.37217	3.2777	16.2695	5.3998	Yes

Table 4.5: Results for 5 runs of the GA on the Pitch for the MIMO system configuration. The highlighted row is the median performer of the set.

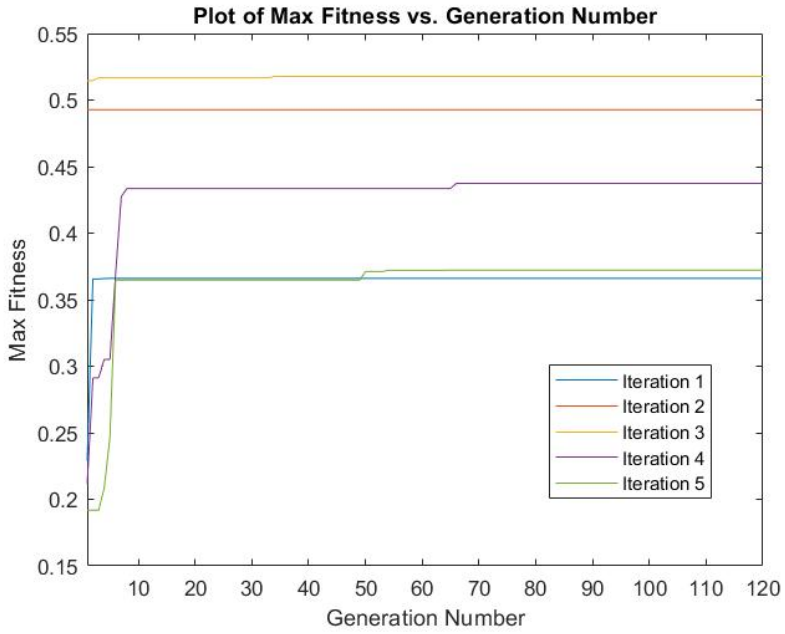

Figure 4.13: Maximum Fitness vs. Generation number for 5 runs of the GA on the Pitch path of the MIMO system.

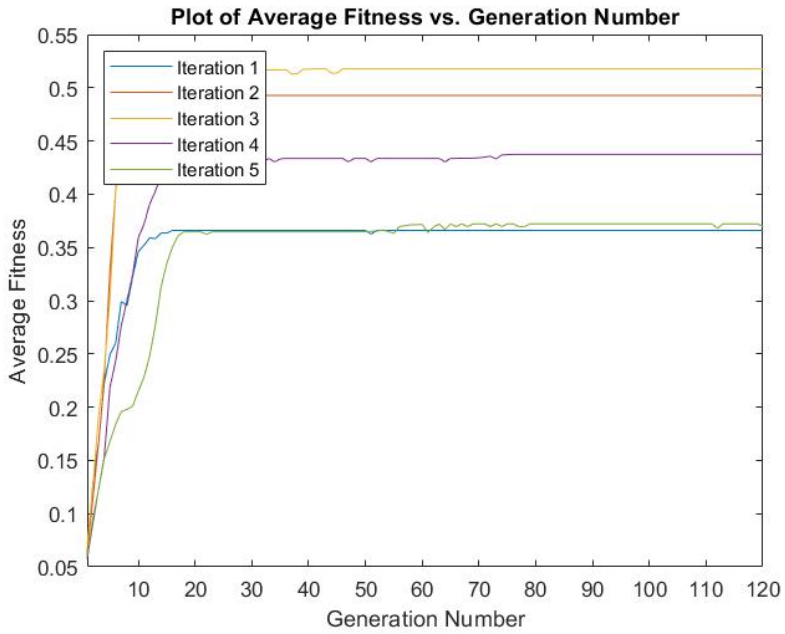

Figure 4.14: Average Fitness vs. Generation number for 5 runs of the GA on the Pitch path of the MIMO system.

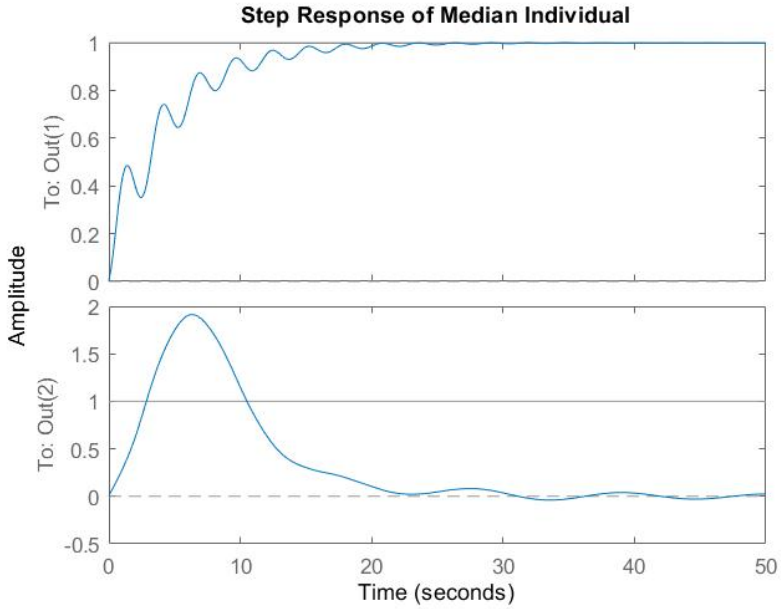

Figure 4.15: Step Response for the fittest individual of run 4 of the GA optimisation on the Pitch path of the MIMO system. The plot on top is the step response of the pitch path, and the bottom plot shows the pure cross-coupling from the pitch path into the yaw path (yaw input set to 0).

Memetic Algorithm

Table 4.6 presents the optimisation results of 5 independent runs of the MA MIMO pitch optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.16. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.17. Run 5 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 5 is shown in Figure 4.18. The average iteration run time for the MA optimisation on the pitch path was 1592.2072 seconds.

	Fitness	**Rise time (s)**	**Settling time (s)**	**Overshoot (%)**	**System Stable?**
Run 1	0.43252	3.5467	16.8845	3.1782	Yes
Run 2	0.46662	3.5673	24.2599	4.7530	Yes
Run 3	0.51945	6.6705	18.3968	1.1586	Yes
Run 4	0.48049	9.1887	68.3867	5.9137	Yes
Run 5	0.47118	6.0307	22.6607	3.1533	Yes

Table 4.6: Results for 5 runs of the MA on the Pitch for the MIMO system configuration. The highlighted row is the median performer of the set.

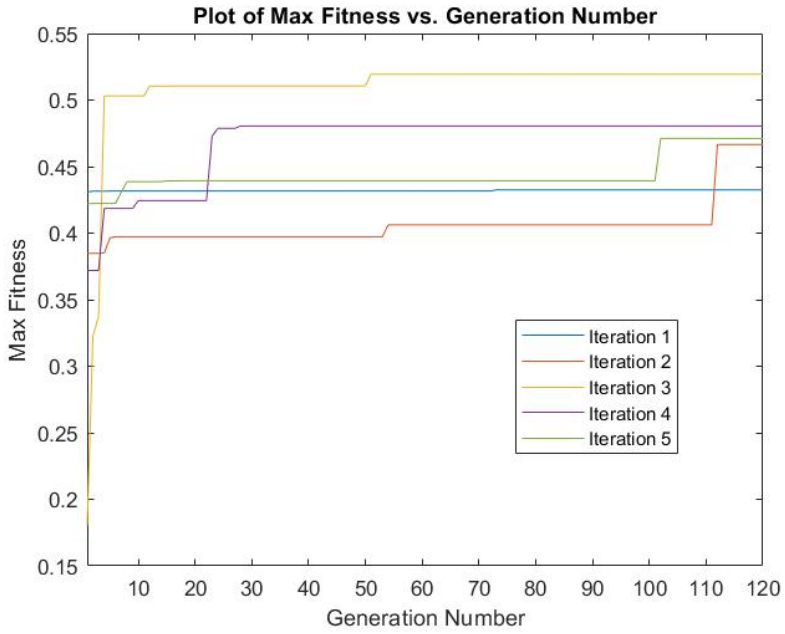

Figure 4.16: Maximum Fitness vs. Generation number for 5 runs of the MA on the Pitch path of the MIMO system.

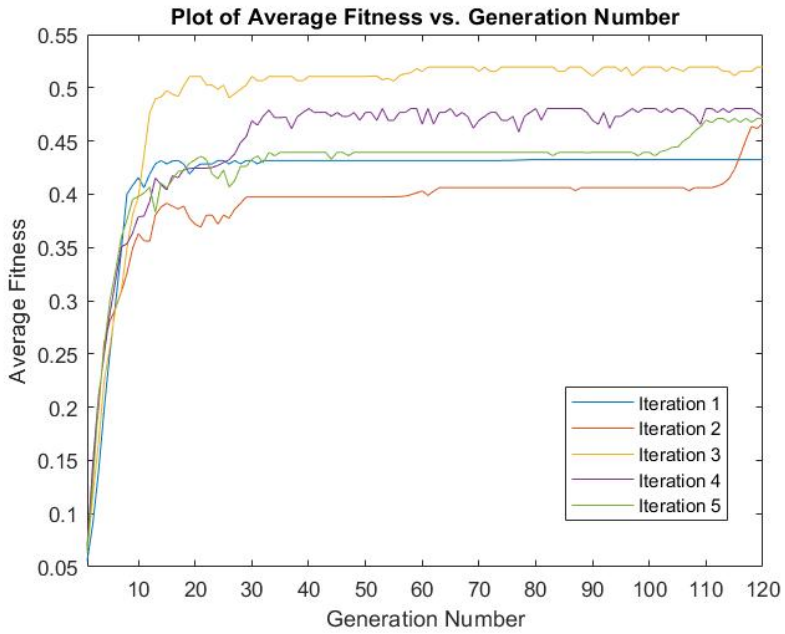

Figure 4.17: Average Fitness vs. Generation number for 5 runs of the MA on the Pitch path of the MIMO system.

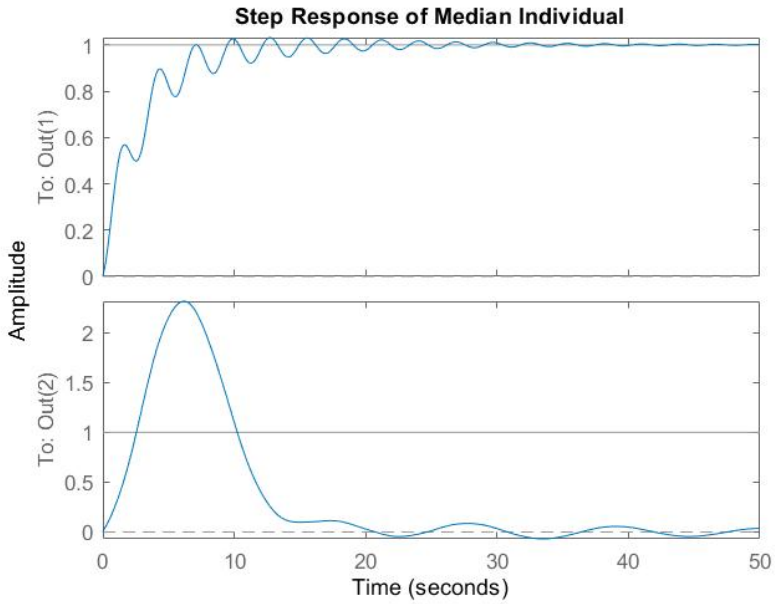

Figure 4.18: Step Response for the fittest individual of run 5 of the MA optimisation on the Pitch path of the MIMO system. The plot on top is the step response of the pitch path, and the bottom plot shows the pure cross-coupling from the pitch path into the yaw path (yaw input set to 0).

Yaw Optimisation Results

Genetic Algorithm

Table 4.7 presents the optimisation results of 5 independent runs of the GA MIMO yaw optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.19. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.20. Run 4 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 4 is shown in Figure 4.21. The average iteration run time for the GA optimisation on the yaw path was 1636.0385 seconds.

	Fitness	Rise time (s)	Settling time (s)	Overshoot (%)	System Stable?
Run 1	0.21904	1.5972	57.7620	26.5864	Yes
Run 2	0.49197	2.4466	50.4228	5.7289	Yes
Run 3	0.67592	2.9218	75.5303	12.5286	Yes
Run 4	0.48115	10.5970	28.9519	1.9383	Yes
Run 5	0.39742	2.3269	46.2773	12.1354	Yes

Table 4.7: Results for 5 runs of the GA on the Yaw for the MIMO system configuration. The highlighted row is the median performer of the set.

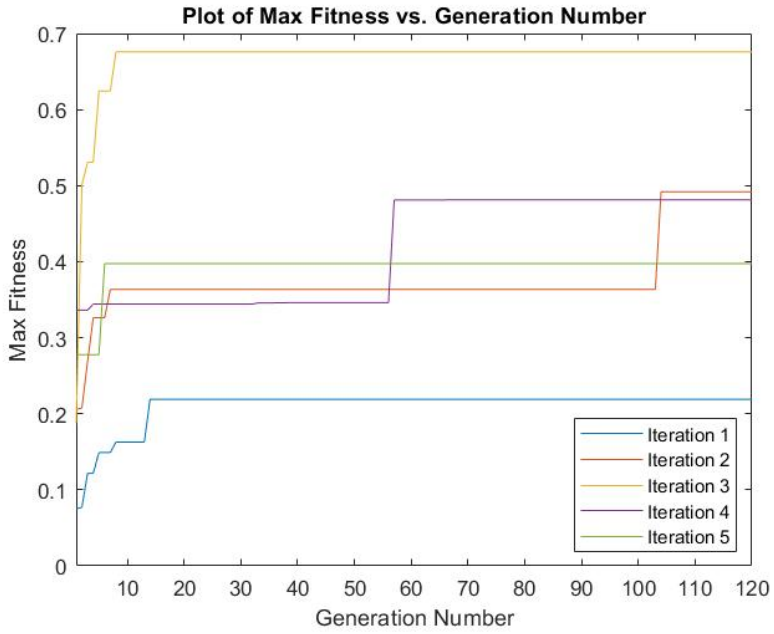

Figure 4.19: Maximum Fitness vs. Generation number for 5 runs of the GA on the Yaw path of the MIMO system.

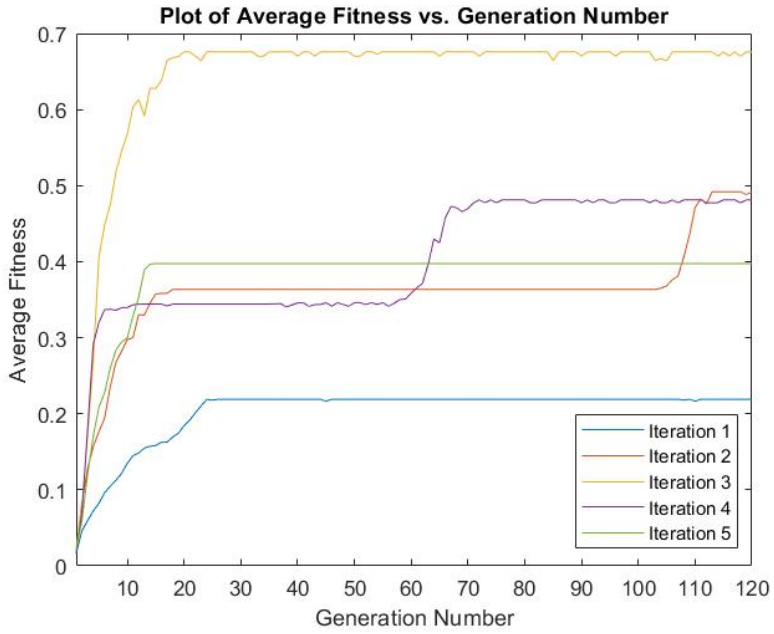

Figure 4.20: Average Fitness vs. Generation number for 5 runs of the GA on the Yaw path of the MIMO system.

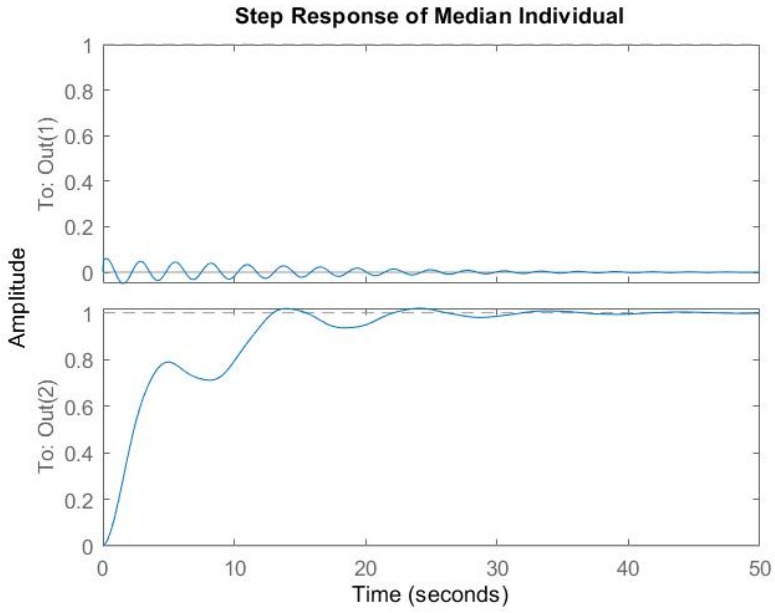

Figure 4.21: Step Response for the fittest individual of run 4 of the GA optimisation on the yaw path of the MIMO system. The plot on top is the step response of the yaw path, and the bottom plot shows the pure cross-coupling from the yaw path into the pitch path (pitch input set to 0).

Memetic Algorithm

Table 4.8 presents the optimisation results of 5 independent runs of the MA MIMO yaw optimisation. The maximum fitness is plotted against generation number for each run of the algorithm, and is shown in Figure 4.22. Similarly, the average fitness per generation for each run of the algorithm is plotted and shown in Figure 4.23. Run 4 was identified as the median performer in the set of runs, and was chosen for analysis. The step response for the fittest individual in run 4 is shown in Figure 4.24. The average iteration run time for the MA optimisation on the yaw path was 1835.9352s seconds.

	Fitness	Rise time (s)	Settling time (s)	Overshoot (%)	System Stable?
Run 1	0.5038	3.5840	114.8864	11.9515	Yes
Run 2	0.67504	3.3517	125.8870	17.3255	Yes
Run 3	1.0089	6.1961	54.5352	5.1239	Yes
Run 4	0.78143	12.5021	92.2993	2.2237	Yes
Run 5	0.25877	2.3092	40.4031	0.0017	Yes

Table 4.8: Results for 5 runs of the MA on the Yaw for the MIMO system configuration. The highlighted row is the median performer of the set.

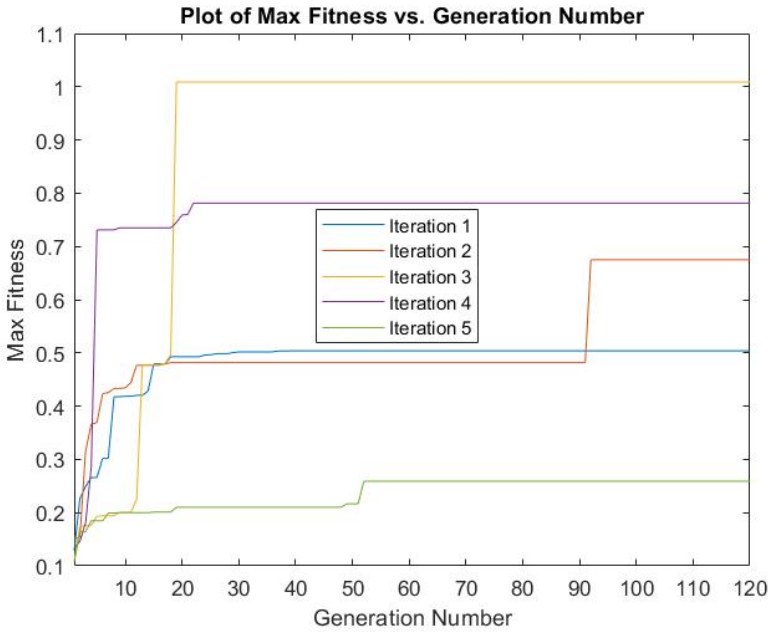

Figure 4.22: Maximum Fitness vs. Generation number for 5 runs of the MA on the Yaw path of the MIMO system.

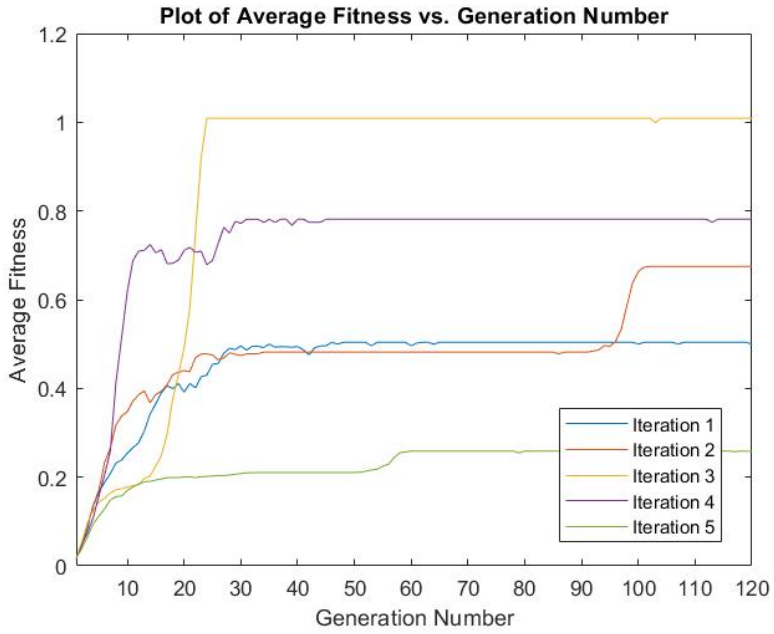

Figure 4.23: Average Fitness vs. Generation number for 5 runs of the MA on the Yaw path of the MIMO system.

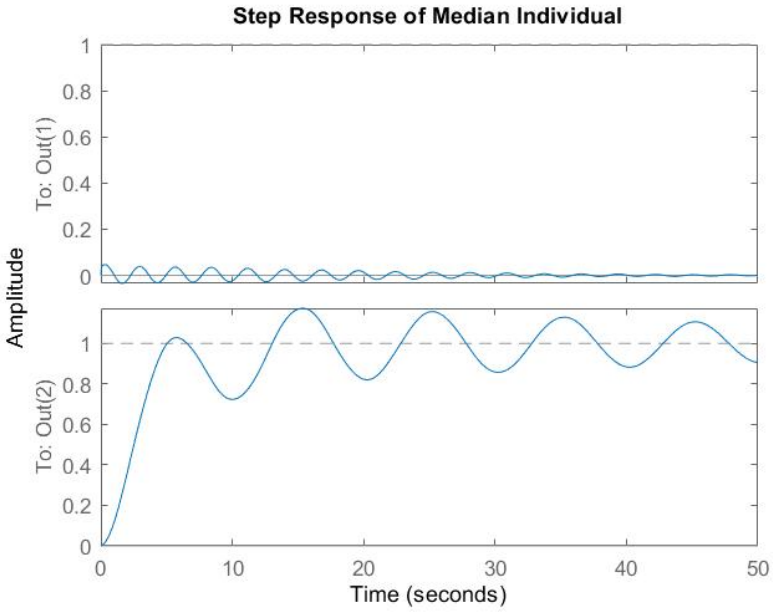

Figure 4.24: Step Response for the fittest individual of run 4 of the MA optimisation on the yaw path of the MIMO system. The plot on top is the step response of the yaw path, and the bottom plot shows the pure cross-coupling from the yaw path into the pitch path (pitch input set to 0).

4.1.3 Table Summary of Optimisation Results

Table 4.9 summarises the results analysed for GA and MA tests in both 1DOF and 2DOF system representations. This table is a useful reference for the following discussion section.

System	Optimisation Type	Path	Fitness	Rise time (s)	Settling time (s)	Overshoot (%)	Average Iteration Runtime (s)
1DOF	GA	Pitch	1.6928	2.944	25.02	9.588	614.117
		Yaw	0.79108	4.8663	40.6602	4.8957	620.8791
	MA	Pitch	3.7246	4.0526	12.0318	2.259	5832.9409
		Yaw	1.1031	12.1574	71.619	7.9021	2524.0169
2DOF	GA	Pitch	0.43738	8.8356	19.7101	0.0958	1542.4625
		Yaw	0.48115	10.5970	28.9519	1.9383	1636.0385
	MA	Pitch	0.47118	6.0307	22.6607	3.1533	1592.2072
		Yaw	0.67504	3.3517	125.8870	17.3255	1835.9352

Table 4.9: Table summary of optimisation results for both 1DOF and 2DOF, GA and MA median runs.

4.1.4 1DOF Testing and Verification

A MATLAB script called TestLinear1DOF, as attached in Appendix G, was developed to complete testing and verification of the system, according to the steps outlined in Section 3.5.3. It should be noted that testing with initial conditions was not implemented in the final test script, as issues were encountered whilst attempting to implement this testing (initial conditions were 4 times larger in magnitude than expected). Therefore, due to time constraints, 1DOF testing was limited to the 1DOF procedure outlined by Prasad et al. (2013), and the 2DOF testing followed the testing parameters by Juang et al. (2008) and were assumed to have initial conditions of 0.

Tables 4.10 and 4.11 present results of the performance criteria and absolute error and control effort values for both sets of parameters optimised by GA and MA. Figures 4.25,4.26,4.27 and 4.28 shows input responses on the left column, and control effort plots in the right column.

System	Optimisation Type	Path	Rise time (s)	Settling time (s)	Overshoot (%)
1DOF	GA	Pitch	2.9446	25.0199	9.5883
		Yaw	4.8664	40.6597	4.8954
	MA	Pitch	4.0527	12.0318	2.259
		Yaw	12.1573	71.6195	7.9027

Table 4.10: 1DOF rise time, settling time and overshoot testing results.

System	Path	Input Signal	Absolute Error (GA)	Absolute Error (MA)	Absolute Control Effort (GA)	Absolute Control Effort (MA)
1DOF	Pitch	Step	41.3033	51.6692	2683.7728	2309.9697
		Sine	118.6786	221.7039	1384.3321	1311.7825
		Square	169.4761	243.5065	2175.8904	1974.8772
	Yaw	Step	84.9281	146.0424	1621.4345	1263.7093
		Sine	308.9649	384.1104	496.3711	392.8848
		Square	349.6059	500.1544	797.4237	590.497

Table 4.11: Absolute Error and Absolute Control Effort Values for 1DOF Pitch and Yaw, of both GA and MA runs.

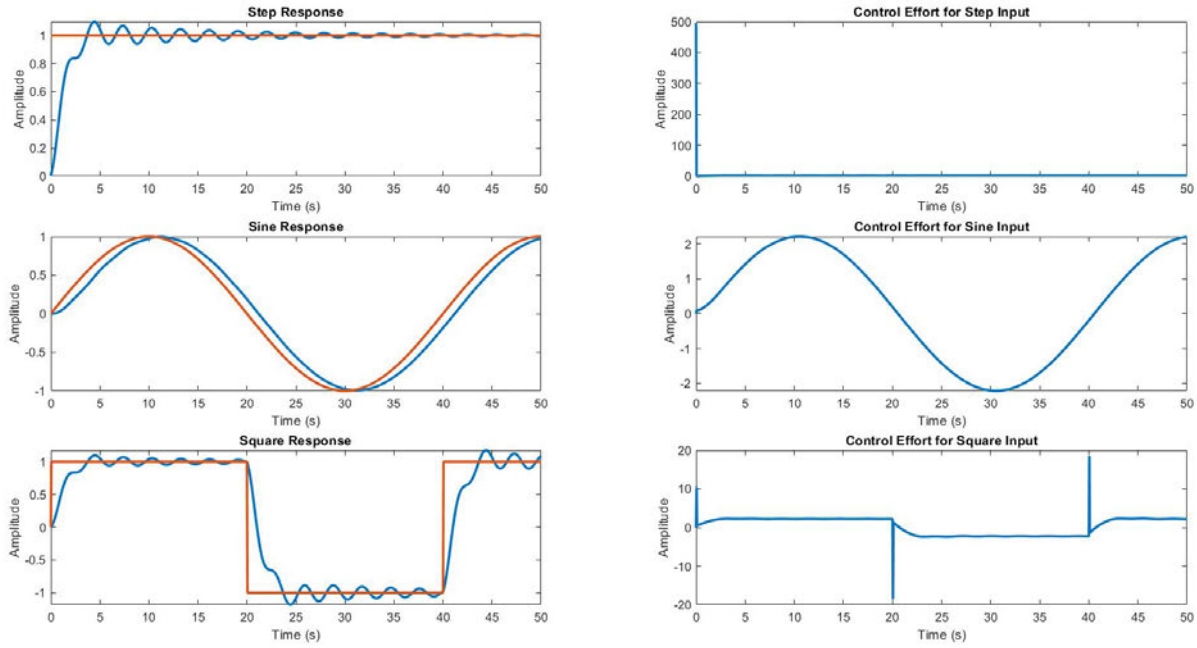

Figure 4.25: GA-optimised Pitch 1DOF test results

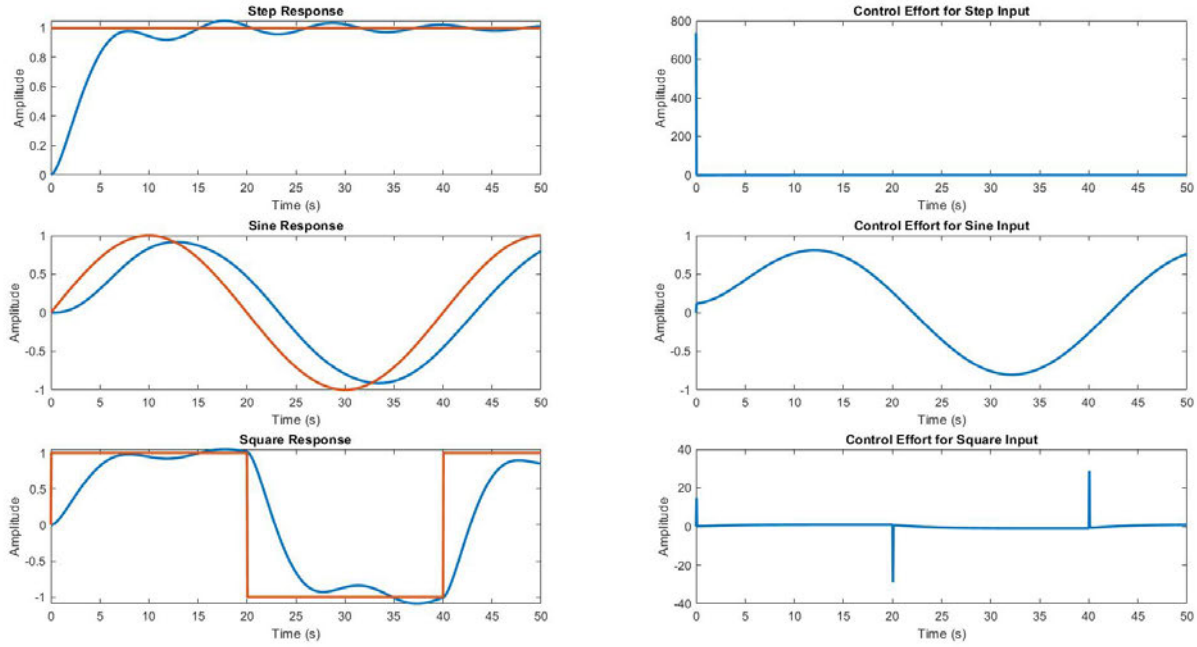

Figure 4.26: GA-optimised Yaw 1DOF test results

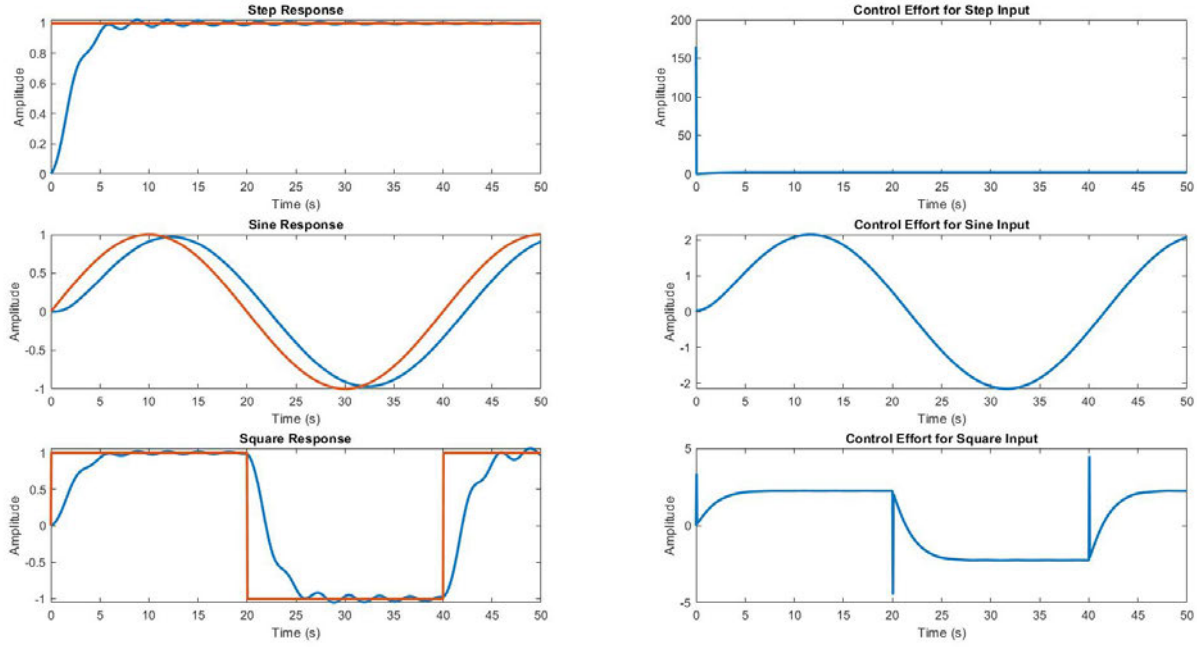

Figure 4.27: MA-optimised Pitch 1DOF test results

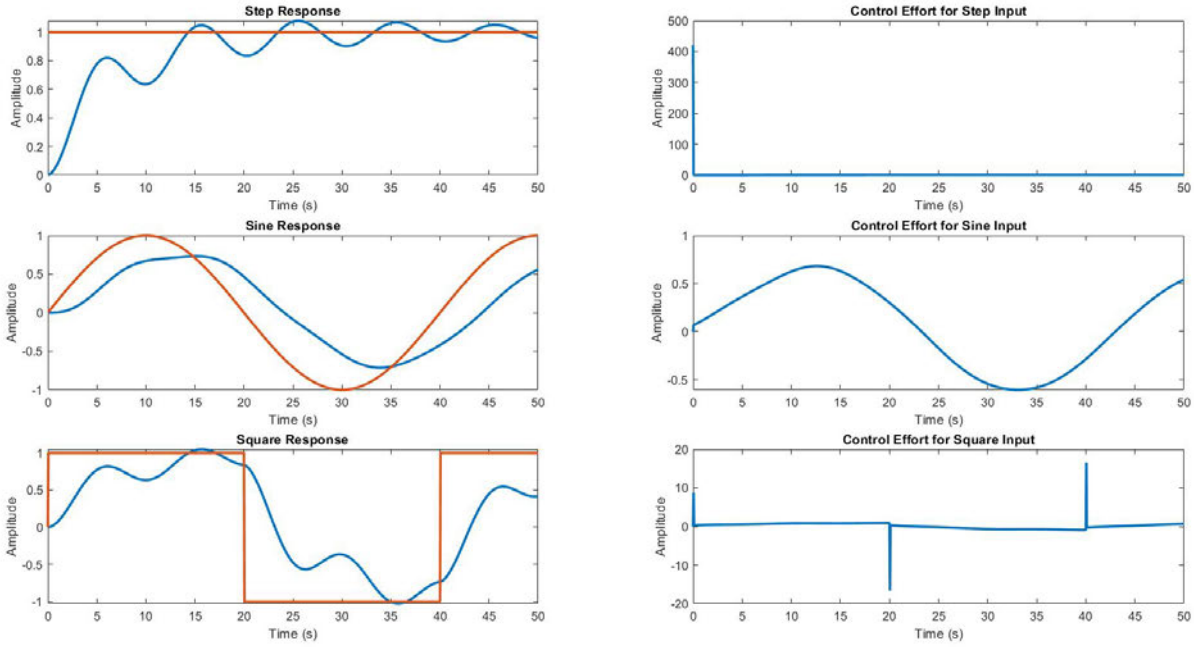

Figure 4.28: MA-optimised Yaw 1DOF test results

4.1.5 2DOF Testing and Verification

Similarly, Tables 4.12 and 4.13 present results of the performance criteria and absolute error and control effort values for both sets of parameters optimised by GA and MA for the 2DOF representation of the system. Figures 4.29,4.30,4.31 and 4.32 shows input responses on the left column, and control effort plots in the right column.

System	Optimisation Type	Path	Rise time (s)	Settling time (s)	Overshoot (%)
2DOF	GA	Pitch	8.8136	19.8276	0.077652
		Yaw	10.5953	28.8658	1.9551
	MA	Pitch	6.0312	22.8118	3.0803
		Yaw	3.3517	125.8924	17.3286

Table 4.12: 2DOF rise time, settling time and overshoot testing results.

System	Path	Input Signal	Absolute Error (GA)	Absolute Error (MA)	Absolute Control Effort (GA)	Absolute Control Effort (MA)
2DOF	Pitch	Step	15.9898	11.2533	540.8233	537.403
		Sine	62.1917	47.3359	232.4934	261.1698
		Square	74.3036	53.1625	343.5694	390.9454
	Yaw	Step	73.9294	88.7869	797.7371	783.4552
		Sine	202.874	192.4944	292.8222	347.2617
		Square	281.2459	261.4839	404.7526	481.6927

Table 4.13: Absolute Error and Absolute Control Effort Values for 2DOF Pitch and Yaw, of both GA and MA runs.

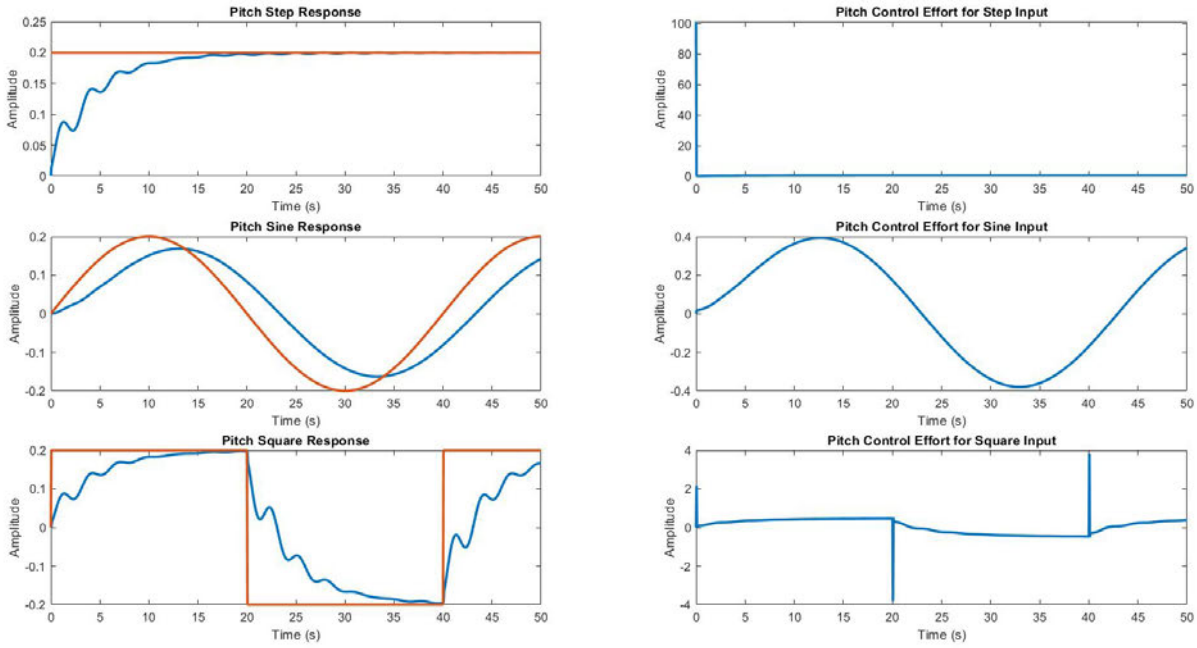

Figure 4.29: GA-optimised Pitch 1DOF test results

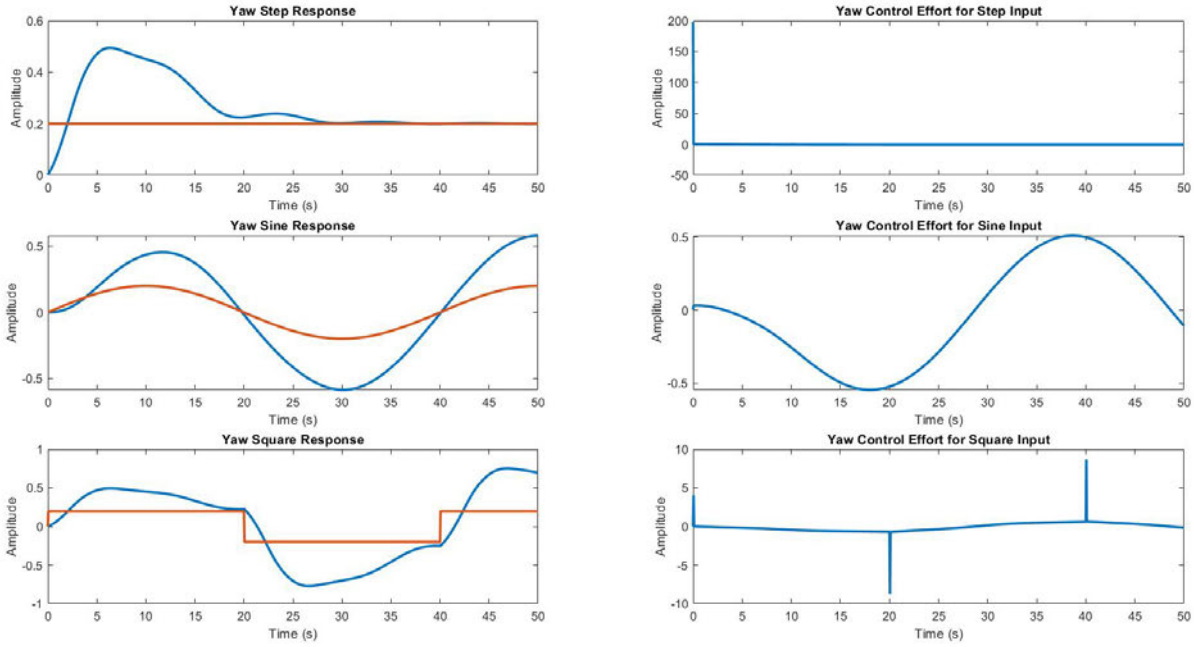

Figure 4.30: GA-optimised Yaw 1DOF test results

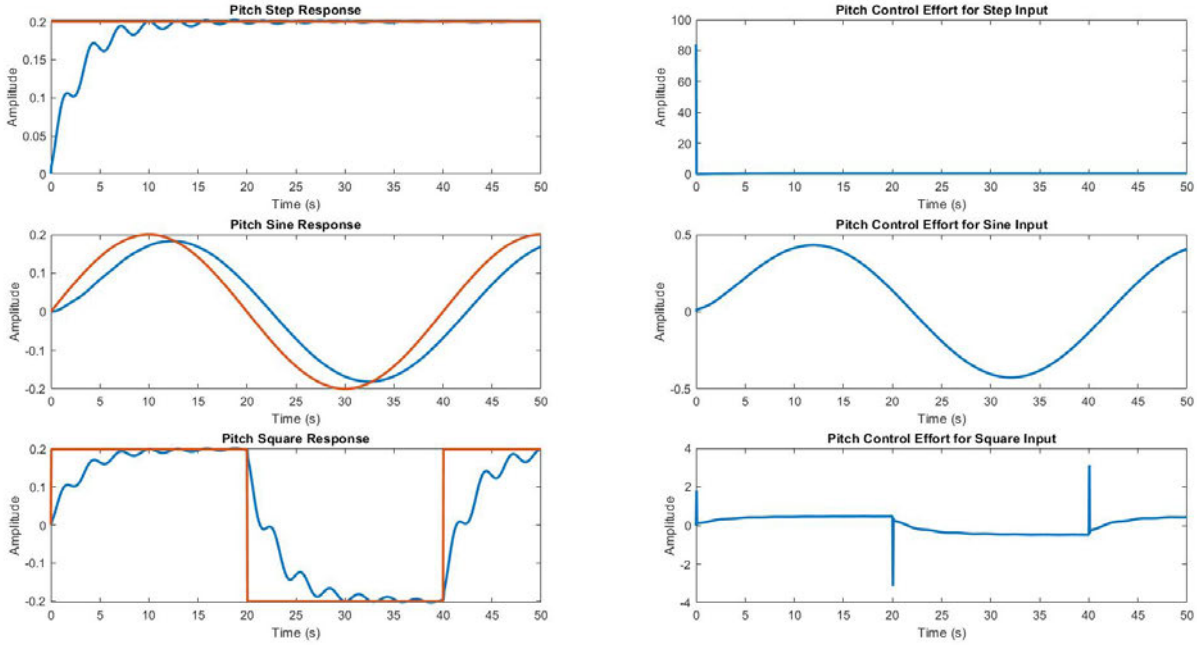

Figure 4.31: MA-optimised Pitch 1DOF test results

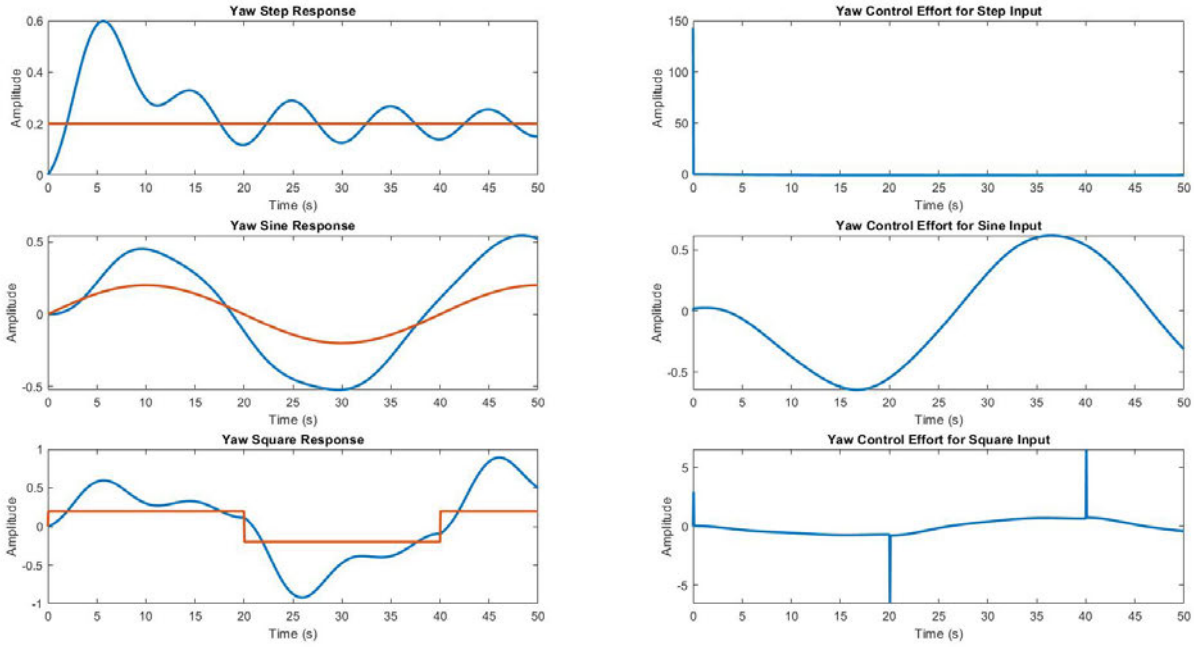

Figure 4.32: MA-optimised Yaw 1DOF test results

4.2 Discussion

As per the first step of Section 3.5.3, the results obtained should be compared to the literature. The main articles of reference are by Prasad et al. (2013) and Juang et al. (2008), where the 1DOF test parameters are derived from the former, and the 2DOF test parameters are derived from the latter.

The results for the 1DOF testing and verification, as listed in Table 4.11, are first analysed. Prasad et al. (2013) obtained the results as presented in Figure 4.14.

System	Reference	Absolute Error	Absolute Control Force
Horizontal 1-DOF	Step	1.075	1.972
	Sine	0.06328	1.513
	Square	4.014	11.38
Vertical 1-DOF	Step	11.22	45.76
	Sine	10.22	44.86
	Square	8.168	44.06
2-DOF System	Step	49.05	80.03
	Sine	43.01	66.92
	Square	45.25	65.23

Table 4.14: Results obtained by Prasad et al. for the TRMS, using Real-valued GA

As can be observed, the error and control force results obtained by Prasad et al. are orders of magnitude lower than the resutls obtained in this project (lower results are better). It is unclear what the cause of such a big difference in values may be, and calls into question whether the methods of testing in this project are different to those implemented in the work by Prasad et al..

However, comparison with Juang et al. shows more comparative results for both the 1DOF and 2DOF, even though initial conditions were not implemented in this project as they were for 1DOF studies in Juang et al.'s work. The 1DOF vertical (pitch) and horizontal (yaw), and 2DOF cross-coupled results are presented from the paper here, for the readers convenience, in Figures 4.15, 4.16 and 4.17

reference	*error*			*control*		
	PID	C-RGA	M-RGA	PID	C-RGA	M-RGA
Step	67.84	50.07	53.60	687.53	605.95	599.49
Sine	72.69	72.43	66.47	1185.74	1178.00	492.77
Square	137.27	106.91	75.40	1234.81	1213.9	457.34

Table 4.15: Vertical (pitch) results obtained for TRMS using different GA methods (Juang et al. 2008)

reference	*error*			*control*		
	PID	C-RGA	M-RGA	PID	C-RGA	M-RGA
Step	26.21	20.13	20.28	17.83	15.98	14.84
Sine	11.58	14.46	8.29	12.47	12.29	7.62
Square	86.36	63.11	38.07	78.66	58.83	37.71

Table 4.16: Horizontal (yaw) results obtained for TRMS using different GA methods (Juang et al. 2008)

Reference		*error*			*control*		
		PID	C-RGA	M-RGA	PID	C-RGA	M-RGA
Step	H	81.2	69.09	54.52	76.71	51.34	40.47
	V	40.11	34.92	27.46	812.36	701.23	617.10
Sine	H	23.21	19.33	20.92	27.41	20.12	18.93
	V	65.74	51.78	52.61	611.70	500.2	501.78
Square	H	150.22	141.52	134.03	202	171.28	165.32
	V	112.85	96.36	90.21	656.37	591.65	551.59

Table 4.17: 2DOF results obtained for TRMS using different GA methods (Juang et al. 2008)

Comparing the 1DOF results by (Juang et al. 2008), but also considering testing methods are slightly different, the project results listed in Table 4.11, perform reasonably better when compared directly with Tables 4.15 and 4.15. None of the project results are better, but they are comparable, unlike the results by Prasad et al. (2013).

Conversely, the 2DOF system in this research performs reasonably well in comparison to the results presented in Table 4.17 by Juang et al. (2008). Whilst the RGA results perform better in most areas, the results are comparable. The cases in which the GA in this research performs better is in pitch step absolute error, pitch square absolute error, pitch step control effort, pitch sine control effort and pitch square control effort. This comparison exercise need not be strenuous; it is enough to show that the MA results can be compared to the base GA with confidence. This does not necessarily infer that the results are optimal, or that the base GA is optimal; but for the simple purpose of demonstrating the efficacy of the MA, this will suffice.

To analyse the performance of the MA as compared to the GA, the key performance metrics, absolute error, absolute control effort and average iteration runtime will be directly compared.

4.2.1 SISO Analysis

Pitch SISO System

For the 1DOF pitch control system tuned by MA, as compared to the equivalent GA test:

- Rise time is increased by 37.63%
- Settling time is decreased by 51.91%
- Overshoot decreased by 76.44%
- Fitness increased by 120.026%
- Absolute net error across input types increased by 56.89%
- Absolute net control effort across input types decreased by 10.3678%
- Average iteration execution runtime increased by 849.809%

Yaw SISO System

For the 1DOF yaw control system tuned by MA, as compared to the equivalent GA test:

- Rise time is increased by 149.83%
- Settling time is increased by 76.14%
- Overshoot increased by 61.41%
- Fitness increased by 39.44%
- Absolute net error across input types increased by 38.58%
- Absolute net control effort across input types decreased by 22.92%
- Average iteration execution runtime increased by 306.52%

4.2.2 MIMO

Pitch MIMO System

For the 2DOF pitch control system tuned by MA, as compared to the equivalent GA test:

- Rise time is decreased by 31.75%
- Settling time is increased by 14.97%
- Overshoot increased by 3191.54%
- Fitness increased by 7.73%
- Absolute net error across input types decreased by 26.71%
- Absolute net control effort across input types increased by 6.50%
- Average iteration execution runtime increased by 3.23%

Yaw MIMO System

For the 2DOF yaw control system tuned by MA, as compared to the equivalent GA test:

- Rise time is decreased by 68.37%
- Settling time is increased by 334.81%

- Overshoot increased by 793.85%
- Fitness increased by 40.297%
- Absolute net error across input types decreased by 2.74%
- Absolute net control effort across input types increased by 7.83%
- Average iteration execution runtime increased by 12.22%

It can be observed that as rise time decreases, settling time and overshoot is often increased. This is not surprising, as minimising one will inevitably increase the other, as these are conflicting constraints. It can also be observed that if absolute error is decreased, then absolute control effort increases. This is also logical, as greater control effort is required to achieve minimum error, especially for higher-order, complex systems. It is important to note however, that MA optimisation produced the fittest individual in every tested case. This means that the overall net control effort and errors have been minimised further than the GA optimisation runs. It must be noted however, that in most cases, the average iteration execution runtime is greater for MA; partially from the local search method introduced, but also due to many executions of the objective function, which for this project, was found to be quite computationally expensive, due to the complex Control Systems Toolbox functions that are utilised to programmatically build and test the systems.

The MA does appear to be more effective at converging to a global optimum than a GA for the TRMS system, in seemingly every test case analysed. However, the computational cost may render the MA unfavourable when compared to smarter and faster approaches. In the results, as can be seen in Figure 4.18, even though somewhat minimised, significant cross-couplings still exist between the pitch to yaw cross-coupling path, and chattering oscillations are present in the converse coupling. Therefore, it is recommended that 4 controllers be used if possible, as other research has suggested. It is important to note that a solution might indeed NOT exist which has the capability to simultaneously control the main path and the cross-coupling with the given system dynamics. Another observation from the data is that the MA seems to rapidly converge within a very few number of generations, which may be due to a lack of diversity. This may need to be addressed by reviewing the LS method, extending the number of generations or increasing diversification (higher mutation, lower crossover, remove elitism) to allow the algorithm to better search global optima.

Although mentioned in Section 3.5.3, nonlinear testing was disregarded, due to the lack of explicit results in the literature for comparison. The analysis would not serve much more purpose than a quick visual inspection, thus this step is also considered as part of future work.

Chapter 5

Conclusions and Further Work

5.1 Conclusions

The aims of the project have developed during the different stages of the project life cycle. Initially the aims were kept broad to encourage a broad scan of the literature. As EA's were found to be a very suitable candidate for this area of research with many potential avenues of investigation, the aims of the project moved toward finding a way to further investigate some of the more recent developments in EA. Through further research, the GA was found, along with the more recent subset of MA. It was found that no current implementations have been made on the TRMS system using the MA, or on any MIMO at all. The aims then moved towards investigating the applicability of the MA with the TRMS, and whether it can be used to fulfil the primary aim of the project - a design of an optimal fuel/cost controller.

The MA was successfully implemented and tested on linear 1DOF and 2DOF representations of the TRMS. Results were positive for the MA, with every test case yielding higher fitness values with MA optimisation than with GA optimisation. The MA appeared to perform equally well in both representations of the system (with the exception of statistical outliers). However, the MA failed to find an optimal solution for significant decoupling of the system, which may either be a system limitation, or potentially even an implementation issue. The MA is also generally very inefficient, even with parallelism implemented. The bottleneck appeared to be the objective function, which utilises many Control Systems Toolbox functions for building and testing systems programmatically.

Limitations aside, the MA still assisted in finding an optimum that is not necessarily the absolute global optimum, but one that certainly shows better performance than the GA alone. Thus the primary aim of the project was achieved and a fuel/cost efficient controller design was able to be fulfilled.

5.2 Further Work

As with all research, there is always more that can be done. Items that are of interest for further work are:

- Review algorithm and diversify population to prevent premature convergence if possible/necessary
- Implement and test on the 33-949S physical system and/or on nonlinear test models. Verify and compare with simulation results.
- Investigate ways to optimise the objective function using parallelism or other advanced mechanisms. Try intercommunication processes (which MATLAB handles internally), but test other methods for efficacy.
- Complete an electrical energy expenditure study on the system and analyse the optimisation algorithm performance in respect to energy savings
- Investigate the feasibility of the algorithm for online parameter tuning, or implementation on a device such as a Field Programmable Gate Array (FPGA).
- Employ and test memetic algorithm LS methods that have not yet been applied to control systems
- Employ multiple local search methods and let the algorithm self-adaptively select local search methods depending on the problem-specific performance of each (Ong & Keane 2004).
- Implement adaptive parameters where possible to reduce the number of heuristic variables. These parameters should adapt to changes in the population (Molina et al. 2005).
- Extrapolate the algorithm and process to a higher DOF system, such as a UAV, or a novel system.

References

Adewuya, A. A. (1996), 'New Methods in Genetic Search with Real-Valued Chromosomes', Master's thesis, Massachusetts Institute of Technology.

Alam, T., Qamar, S., Dixit, A. & Benaida, M. (2020), 'Genetic algorithm reviews, implementations, and applications', *International Journal of Engineering Pedagogy (iJEP)* . DOI: 10.31219/osf.io/8tng5.

Albadr, M. A., Tiun, S., Ayob, M. & AL-Dhief, F. (2020), 'Genetic algorithm based on natural selection theory for optimization problems', *Symmetry* **12**(11). DOI: 10.3390/sym12111758.

Alharbi, W. & Gomm, B. (2017), ''Genetic Algorithm Optimisation of PID Controllers for a Multivariable Process", *International Journal of Recent Contributions from Engineering, Science & IT* **5**(1). DOI: 10.3991/ijes.v5i1.6692.

Assiroj, P., Warnars, H. L. H. S., Abdurachman, E., Kistijantoro, A. I. & Doucet, A. (2021), 'The implementation of memetic algorithm on image: a survey', *Journal of Mathematical and Computational Science* **11**(6), 6872–6896. DOI: 10.28919/jmcs/5961.

Ayala, F. J. (2009), 'Darwin and the scientific method', *Proceedings of the National Academy of Sciences* **106**(1), 10033–10039. DOI: 10.1073/pnas.0901404106.

Ben Hariz, M. & Bouani, F. (2015), Design of controllers for decoupled tito systems using different decoupling techniques, *in* '2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)', pp. 1116–1121. DOI: 10.1109/MMAR.2015.7284035.

Bereta, M. (2019), 'Baldwin effect and Lamarckian evolution in a memetic algo-

rithm for Euclidean Steiner tree problem', *Memetic Computing* **11**, 35–52. DOI: https://doi.org/10.1007/s12293-018-0256-7.

Chalupa, P., Přikryl, J. & Novák, J. (2015), 'Modelling of twin rotor mimo system', *Procedia Engineering* **100**, 249–258. DOI: 10.1016/j.proeng.2015.01.365.

Choi, S.-S. & Moon, B.-R. (2005), 'A graph-based lamarckian-baldwinian hybrid for the sorting network problem', *IEEE Transactions on Evolutionary Computation* **9**(1), 105–114. DOI: 10.1109/TEVC.2004.841682.

Devanshu, A. (2017), *in* ''Genetic Algorithm Tuned PID Controller for Process Control". DOI: 10.1109/ICISC.2017.8068639.

Didactic, L. (2021), '33-007i and 33-007-pci twin rotor mimo system', `http://www.ld-didactic.de/en/ld-didactic-download-center.html`.

Doğruer, T. & Tan, N. (2019), Decoupling control of a twin rotor mimo system using optimization method, *in* '2019 11th International Conference on Electrical and Electronics Engineering (ELECO)', pp. 780–784. DOI: 10.23919/ELECO47770.2019.8990435.

Fang, Y. & li, J. (2010), A review of tournament selection in genetic programming, *in* 'Genetic Programming', pp. 181–192. DOI: 10.1007/978-3-642-16493-4_19.

Feedback Instruments Ltd. (n.d.), *33-949S*, 1 edn, Feedback Instruments Ltd, Park Road, Crowborough, East Sussex, TN6 2QR, UK.

Ho, M.-T., Datta, A. & Bhattacharyya, S. (1998), 'An elementary derivation of the routh-hurwitz criterion', *IEEE Transactions on Automatic Control* **43**(3), 405–409. DOI: 10.1109/9.661607.

Jayachitra, A. & Vinodha, R. (n.d.), 'Genetic algorithm based pid controller tuning approach for continuous stirred tank reactor', *Advances in Artificial Intelligence* **2014**(791230), 1–8. DOI: 10.1155/2014/791230.

Juang, J.-G., Huang, M.-T. & Liu, W.-K. (2008), 'Pid control using presearched genetic algorithms for a mimo system', *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* **38**(5), 716–727. DOI: 10.1109/TSMCC.2008.923890.

Juang, J.-G. & tu, K.-T. (2013), 'Design and realization of a hybrid intelligent controller for a twin rotor MIMO system', *Journal of Marine Science and Technology* **21**(3), 333–341. DOI: 10.6119/JMST-012-1026-1.

Krasnogor, N. & Smith, J. (2005), 'A tutorial for competent memetic algorithms: model, taxonomy, and design issues', *IEEE Transactions on Evolutionary Computation* **9**(5), 474–488. DOI: 10.1109/TEVC.2005.850260.

Kumar, P. & Narayan, S. (2016), 'Optimal robust design for a twin rotor system using multi-objective genetic algorithm tuned model predictive controller', **9**(11), 5043–5056. https://www.researchgate.net/publication/311794292_Optimal_robust_design_for_a_twin_rotor_system_using_multi-objective_genetic_algorithm_tuned_model_predictive_controller.

Lin, G. & Liu, G. (2010), Tuning PID controller using adaptive genetic algorithms, *in* '2010 5th International Conference on Computer Science Education', pp. 519–523. DOI: 10.1109/ICCSE.2010.5593559.

Mahfoud, S., Derouich, A., EL Ouanjli, N., EL Mahfoud, M. & Taoussi, M. (2021), 'A new strategy-based pid controller optimized by genetic algorithm for dtc of the doubly fed induction motor', *Systems* **9**(2). DOI: 10.3390/systems9020037.

Maliński, u. & Figwer, J. (2015), Nonlinear system identification using memetic algorithms, *in* '2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)', pp. 1086–1091. DOI: 10.1109/MMAR.2015.7284030.

Mandal, A., Zafar, H., Ghosh, P., Das, S. & Abraham, A. (2011), An efficient memetic algorithm for parameter tuning of pid controller in avr system, *in* '2011 11th International Conference on Hybrid Intelligent Systems (HIS)', pp. 265–270. DOI: 10.1109/HIS.2011.6122116.

Mathworks (2021*a*), 'Decide when to use parfor'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/help/parallel-computing/reduction-variable.html*

Mathworks (2021*b*), 'Parallel computing toolbox'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/products/parallel-computing.html*

Mathworks (2021*c*), 'parfeval'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/help/parallel-computing/parallel.pool.parfeval.html*

Mathworks (2021*d*), 'parfor'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/help/parallel-computing/parfor.html*

Mathworks (2021*e*), 'Reduction variables'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/help/parallel-computing/reduction-variable.html*

Mathworks (2021*f*), 'What is parallel computing?'. Web page, viewed 19 October 2021.
URL: *https://au.mathworks.com/help/parallel-computing/what-is-parallel-computing.html*

Mirzal, A., Yoshii, S. & Furukawa, M. (2012), 'Pid parameters optimization by using genetic algorithm', *ISTECS Journal* **8**(2006), 34–43. https://arxiv.org/abs/1204.0885.

Mishra, A. & Shukla, A. (2017), Analysis of the effect of elite count on the behavior of genetic algorithms: A perspective, *in* '2017 IEEE 7th International Advance Computing Conference (IACC)', pp. 835–840. DOI: 10.1109/IACC.2017.0172.

Molina, D., Herrera, F. & Lozano, M. (2005), Adaptive local search parameters for real-coded memetic algorithms, *in* '2005 IEEE Congress on Evolutionary Computation', Vol. 1, pp. 888–895 Vol.1.

Moscato, P. & Cotta, C. (2003), *A Gentle Introduction to Memetic Algorithms*, Springer US, Boston, MA, pp. 105–144.

Naranjo, J. E., Serradilla, F. & Nashashibi, F. (2020), 'Speed control optimization for autonomous vehicles with metaheuristics', *Electronics* **9**(4). DOI: 10.3390/electronics9040551.

Neri, F. & Cotta, C. (2012), *A Primer on Memetic Algorithms*, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23247-3_4.

Norsahperi, N. & Danapalasingam, K. (2020), 'Particle swarm-based and neuro-based fopid controllers for a twin rotor system with improved tracking performance and energy reduction', *ISA Transactions* **102**, 230–244. DOI: 10.1016/j.isatra.2020.03.001.

Ong, Y. S. & Keane, A. (2004), 'Meta-lamarckian learning in memetic algorithms', *IEEE Transactions on Evolutionary Computation* **8**(2), 99–110. DOI: 10.1109/TEVC.2003.819944.

Prasad, G. D., Manoharan, P. S. & Ramalakshmi, A. P. S. (2013), PID control scheme for twin rotor MIMO system using a real valued genetic algorithm with a predeter-

mined search range, *in* '2013 International Conference on Power, Energy and Control (ICPEC)', pp. 443–448. DOI: 10.1109/ICPEC.2013.6527697.

Rivera-Santos, E. (2021), 'routh.m', https://www.mathworks.com/matlabcentral/fileexchange/58-routh-m. Retrieved from MATLAB Central File Exchange.

Sagharchi, F. (2021), 'Routh-hurwitz stability criterion', https://www.mathworks.com/matlabcentral/fileexchange/17483-routh-hurwitz-stability-criterion. Retrieved from MATLAB Central File Exchange.

Salazar Alvarez, T. (2010), 'Mathematical model and simulation for a helicopter with tail rotor'.

Sarvart, M. A. B. (2001), 'Modelling and Control of a Twin Rotor MIMO System', PhD thesis, University of Sheffield. unpublished thesis.

Serradilla, F., Cañas, N. & Naranjo, J. E. (2020), 'Optimization of the energy consumption of electric motors through metaheuristics and pid controllers', *Electronics* **9**(11). DOI: 10.3390/electronics9111842.

Shyr, W.-J., Wang, B.-W., Yeh, Y.-Y. & Su, T.-J. (2002), Design of optimal pid controllers using memetic algorithm, *in* 'Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)', Vol. 3, pp. 2130–2131 vol.3. DOI: 10.1109/ACC.2002.1023951.

Sivadasan, J. & Iruthayarajan, M. W. (2018), ''Tuning of Nonlinear PID Controller for TRMS using Evolutionary Computation Methods", *Tehnički vjesnik* **25**(1). DOI: 10.17559/TV-20170612090511.

Sloss, A. N. & Gustafson, S. (2020), *in* W. Banzhaf, E. Goodman, L. Sheneman, L. Trujillo & B. Worzel, eds, 'Genetic Programming Theory and Practice XVII', Springer, Cham. https://doi.org/10.1007/978-3-030-39958-0_16.

Subudhi, B. & Jena, D. (2009), Nonlinear system identification of a twin rotor mimo system, *in* 'TENCON 2009 - 2009 IEEE Region 10 Conference', pp. 1–6. DOI: 10.1109/TENCON.2009.5395966.

Thainiam, P. (2019), The effects of memes on memetic algorithms for solving quadratic assignment problem, *in* '2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)', pp. 1334–1338. DOI: 10.1109/IEEM44572.2019.8978780.

Toha, S. F. & Tokhi, M. O. (2009), Real-coded genetic algorithm for parametric modelling of a TRMS, *in* '2009 IEEE Congress on Evolutionary Computation', pp. 2022–2028.

Toha, S., Julai, S. & Tokhi, M. (2012), 'Ant colony based model prediction of a twin rotor system', *Procedia Engineering* **41**, 1135–1144. DOI: 10.1016/j.proeng.2012.07.293.

Wen, P. & Lu, T. (2008), 'Decoupling control of a twin rotor MIMO system using robust deadbeat control technique', *IET Control Theory & Applications* **2**(11), 999–1007.

Wilbur, M. L., Mistry, M. P., Lorber, P. F., Blackwell, R., Barbarino, S., Lawrence, T. H. & Arnold, U. T. (2018), Chapter 24 - rotary wings morphing technologies: State of the art and perspectives, *in* A. Concilio, I. Dimino, L. Lecce & R. Pecora, eds, 'Morphing Wing Technologies', Butterworth-Heinemann, pp. 759–797. DOI: 10.1016/B978-0-08-100964-2.00024-1.

Zhang, C., Gao, L., Li, X. & Wu, Q. (2012), 'A novel electromagnetism-like mechanism algorithm with modified solis and wets local search for global optimisation', *Int. J. of Services Operations and Informatics* **7**, 117 – 135.

Zhao, G., Luo, W., Nie, H. & Li, C. (2008), A genetic algorithm balancing exploration and exploitation for the travelling salesman problem, *in* '2008 Fourth International Conference on Natural Computation', Vol. 1, pp. 505–509. DOI: 10.1109/ICNC.2008.421.

Appendix A

Project Specification

ENG 4111/2 Research Project

Project Specification

For:	**Aaron Coutts**
Topic:	Optimal fuel cost controller design for a helicopter/twin rotor system
Supervisors:	Prof. Paul Wen Dr. Bob Song
Sponsorship:	Faculty of Health, Engineering & Sciences
Project Aim:	Optimise fuel/energy performance to reduce energy consumption and costs in helicopter twin rotor systems.

Program: Version 2, 16 September 2021

1. Research background and learn the Twin Rotor Multi-Input Multi-Output System (TRMS). Obtain simulation models for testing.
2. Research Artificial Intelligence (AI) Techniques and identify potential knowledge or implementation gaps in the literature.
3. Design a PID controller for the system using traditional techniques (i.e. Ziegler-Nichols Method - non AI techniques).
4. Investigate and identify an algorithm novel to optimising the TRMS system, with a suitable objective function to minimise the energy costs of the TRMS system whilst maintaining performance.
5. Implement algorithm and conduct testing on simulation models. Compare the results with other select algorithms and methods, both traditional and AI.
6. Refine algorithm and implement improvements where necessary.

As time and resources permit:

1. Implement the optimised PID controllers in the lab on the 33-949S physical system. Verify and compare with simulation results. Also compare against other physical implementation of select algorithms and methods, both traditional and AI.
2. Further improve algorithms through any discovered novel approaches.
3. Complete an electrical energy expenditure study on the system and analyse the optimisation algorithm performance in respect to energy savings
4. Investigate feasibility of algorithm for online parameter tuning.

Agreed:

Student Name:	Aaron Coutts
Date:	16 September 2021
Supervisor Name:	Prof. Paul Wen Dr. Bo Song
Date:	

Appendix B

Risk Assessment

University of Southern Queensland

USQ Safety Risk Management System

Print View

Version 2.0

Safety Risk Management Plan

Risk Management Plan ID:	Status:	Current User:	Author:	Supervisor:	Approver:
RMP_2021_5646	*Approval Requested*	[redacted]	[redacted]	[redacted]	[redacted]

Assessment Title:	Engineering Research Project 2021 - Aaron Coutts	Assessment Date:	7/06/2021
Workplace (Division/Faculty/Section):	204070 - School of Mechanical and Electrical Engineering	Review Date:	(5 years maximum)

Approver:	*Supervisor: (for notification of Risk Assessment only)*
Paul Wen	*Paul Wen*

Context

DESCRIPTION:

What is the task/event/purchase/project/procedure?	Optimise fuel performance to reduce operational costs of helicopter twin rotor systems		
Why is it being conducted?	To fulfill the research component of the Bachelor of Engineering (Honours) program		
Where is it being conducted?	USQ - Toowoomba & Ext		
Course code (if applicable)	ENG4111 & ENG4112	Chemical Name (if applicable)	

WHAT ARE THE NOMINAL CONDITIONS?

Personnel involved	Aaron Coutts, Paul Wen and other USQ staff
Equipment	Personal computer and electronics
Environment	Personal office space and USQ laboratory

Other	
Briefly explain the procedure/process	Lab verification of results on the TRMS 33-949S model and general development of project

Assessment Team - who is conducting the assessment?

Assessor(s):	Paul Wen
Others consulted: (eg elected health and safety representative, other personnel exposed to risks)	

Risk Matrix

Probability	Consequence				
	Insignificant No Injury 0-$5K	**Minor** First Aid $5K-$50K	**Moderate** Med Treatment $50K-$100K	**Major** Serious Injury $100K-$250K	**Catastrophic** Death More than $250K
Almost Certain 1 in 2	**M**	**H**	**E**	**E**	**E**
Likely 1 in 100	**M**	**H**	**H**	**E**	**E**
Possible 1 in 1,000	**L**	**M**	**H**	**H**	**H**
Unlikely 1 in 10,000	**L**	**L**	**M**	**M**	**M**
Rare 1 in 1,000,000	**L**	**L**	**L**	**L**	**L**

Recommended Action Guide	
Extreme:	**E**= Extreme Risk – Task ***MUST NOT*** proceed
High:	**H** = High Risk – Special Procedures Required (Contact USQSafe) Approval by VC only
Medium:	**M**= Medium Risk - A Risk Management Plan/Safe Work Method Statement is required
Low:	**L**= Low Risk - Manage by routine procedures.

Risk Register and Analysis

	Step 1	Step 2	Step 2a	Step 2b	Step 3			Step 4				
	Hazards: From step 1 or more if identified	**The Risk:** What can happen if exposed to the hazard without existing controls in place?	**Consequence:** What is the harm that can be caused by the hazard without existing controls in place?	**Existing Controls:** What are the existing controls that are already in place?	**Risk Assessment:** Consequence x Probability Risk Level			**Additional Controls:** Enter additional controls if required to reduce the risk level	**Risk assessment with additional controls:** Has the consequence or probability changed?			
					Probability	Risk Level	ALARP		Consequence	Probability	Risk Level	ALARP
	Example											
	Working in temperatures over 35^{0} C	*Heat stress/heat stroke/exhaustion leading to serious personal injury/death*	*catastrophic*	*Regular breaks, chilled water available, loose clothing, fatigue management policy.*	*possible*	*high*	*No*	*temporary shade shelters, essential tasks only, close supervision, buddy system*	*catastrophic*	*unlikely*	*mod*	*Yes*
1	Working with...	Electrocution and electronic equipment failure	Minor	Electrical equipment in USQ lab is protected by RCD. Electronic equipment is operated at low voltage. Ensure that electrical equipment is in good condition before use	Unlikely	Low	☑					☐
2	High speed r...	Cuts, bruises, or significant damage to body parts that come in contact with spinning rotors	Minor	Guards should be in place surrounding rotors to prevent exposure of spinning parts. Guards should be checked before operating device. When device is operating, maintain appropriate distances from spinning parts.	Unlikely	Low	☑					☐
3	Hard Drive/P...	Loss of Critical Files	Insignificant	Backup files often on multiple devices (for security). Ensure files are backed-up to the cloud.	Unlikely	Low	☑					☐
4	Travelling to/...	Car Crash	Moderate	Drive safely to conditions. Follow all road rules.	Rare	Low	☑					☐
5	Ergonomics	Musculoskeletal strain, eye strain and fatigue	Minor	Work with correct posture and take regular work breaks	Unlikely	Low	☑					☐

Step 5 - Action Plan (for controls not already in place)

	Additional Controls:	Exclude from Action Plan: (repeated control)	Resources:	Persons Responsible:	Proposed Implementation Date:

Supporting Attachments

No file attached

Step 6 – Request Approval

Drafters Name:	Aaron Coutts	Draft Date:	7/06/2021
Drafters Comments:			

Assessment Approval: **All risks are marked as ALARP** 0

Maximum Residual Risk Level: **Low - Manager/Supervisor Approval Required** 1

Document Status:	Approval Requested

Step 6 – Approval

Approvers Name:	Paul Wen	Approvers Position Title:	
Approvers Comments:			

I am satisfied that the risks are as low as reasonably practicable and that the resources required will be provided.

Approval Decision:	Approve / Reject Date:	Document Status: Approval Requested

Appendix C

Ethical Clearance

There are no Ethical Clearances applicable to this project.

Appendix D

Main Code and Objective Function

D.1 The `ma.m` Main Memetic Algorithm Script

The script `ma.m` is the main script to be called to start and run the Memetic Algorithm.

Listing D.1: Memetic Algorithm Main Script.

```
clear variables
close all
clc

addpath('helpers')
addpath('operators')

%initialise population parameters
numGenes = 3;
popSize = 80;
maxGenerations = 120;

%Operator rates
mutationRate = 0.01;
crossoverRate = 0.8;
localSearchRate = 0.05;  %0.05
eliteCount = 1;

%Gene boundaries (PID parameters)
pBounds = [0, 5];       %100    1.5    4
iBounds = [0, 5];       %10       1    4
dBounds = [0, 5];       %1        1    4

%Map gene index to bounds - this would be useful if a different
    ↪ number of
%genes were used (e.g. a FOPID)
geneBounds = {pBounds iBounds dBounds};

%------------ transfer function and related variables setup
    ↪ ------------%
channel = 2;          %1: Pitch, 2: Yaw
system_type = 'MIMO';
%fix the PID controller that is not being tuned with
    ↪ predetermined values
%K_fixed = [0.13136 0.27323 0.73833];     %GA Yaw SISO
%K_fixed = [0.59458 0.60207 0.50529];     %GA Pitch MIMO
K_fixed = [0.62141 0.89421 0.42039];     %MA Pitch MIMO

if channel == 1
    %pitch tf
    num = [0.01657 0.4194 2.454];
    den = [1 1.487 4.403 5.449];
elseif channel == 2
    %yaw tf
```

```
    num = [0.0009881 0.03361 0.4065];
    den = [1 1.345 0.4568 0.3826];
end

%global variables to do with simulating the system
global G
if strcmp(system_type, 'MIMO')
    G11 = tf([0.01657 0.4194 2.454],[1 1.487 4.403 5.449]);    %
        ↪ pitch main
    G12 = tf([0.04986 0.0962],[1 0.2377 4.902]);    %yaw
        ↪ coupling into pitch
    G21 = tf([0.02248 0.4527],[1 0.4099 0.2181]);    %pitch
        ↪ coupling into yaw
    G22 = tf([0.0009881 0.03361 0.4065],[1 1.345 0.4568 0.3826])
        ↪ ;    %yaw main

    G = [G11 G12;
        G21 G22];
else
    G = tf(num, den);
end

global dt
dt = 0.05;
global T
T = 50;

%save global variables to .mat file so they are accessible for
    ↪ workers
%(globals are not accessible to workers)
save('ws_vars', 'G', 'dt', 'T', 'channel', 'system_type', '
    ↪ K_fixed')

%
    ↪ ----------------------------------------------------------------------
    ↪

%TODO DELETE THE FOLLOWING - JUST FOR TESTING
global unstableCount stableCount;
unstableCount = 0;
stableCount = 0;
%END

%control for calculating percentage of stable systems for each
    ↪ geneation
calculateStableSystems = false;
percentageOfStableSystems = [];

%setup variables to control the number of runs and record data
    ↪ for each run
iterations = 5;
bestIndividualsOfEachRun = zeros(iterations,numGenes);
```

```
bestIndividualsOfEachRunFitness = zeros(iterations,1);

legendStrBuilder = strings(iterations,1); %for building the
    ↪ legend when plotting

%waitbar for visual representation of progress
f = waitbar(0, 'Starting Memetic Algorithm Optimisation...', '
    ↪ Name', 'Optimising...');

tic;
runTimeTracker = 0;

%start global iteration here
for iter = 1:iterations
disp(strcat(['------------------------------Run ' num2str(iter) '
    ↪ ------------------------------']))
if channel == 1
    channel_name = 'Pitch';
elseif channel == 2
    channel_name = 'Yaw';
end
disp(strcat(['Channel and System Type: ' channel_name ' '
    ↪ system_type]))
%solis-wet rho and treshold values that are updated and accessed
    ↪  through
%stored .mat file. These are here as they need to be reset each
    ↪ iteration
rho = 0.01;
threshold = 0.001;
save('solis_wets_ls_vars', 'rho', 'threshold')

%--------------------Generate initial population
    ↪ ----------------------------%
%formula for generating N random numbers in the interval (a,b):
%r = a + (b-a).*rand(N,1)

a = pBounds(1);
b = pBounds(2);
N = popSize;
pPop =  a + (b-a).*rand(N,1);    %Generate genes for P

a = iBounds(1);
b = iBounds(2);
iPop =  a + (b-a).*rand(N,1);    %Generate genes for I

a = dBounds(1);
b = dBounds(2);
dPop = a + (b-a).*rand(N,1);     %Generate genes for D

pop = [pPop iPop dPop];          %Combine to form population

%for tracking fitness over iterationsw
```

```
maxFitnessValues = zeros(1, maxGenerations + 1);
averageFitnessValues = zeros(1, maxGenerations + 1);

%------------------------- Start Operations
    ↪ -------------------------%
gen = 1;
while (gen <= maxGenerations)
    %EVALUATE FITNESS
    %create vector of fitness values that correspond to the
        ↪ fitness values of
    %each chromosome in the population, for efficiency.
    %i.e. fitnessVec(k) is fitness of pop(k,:)
    fitnessVec = zeros(popSize,1);
    parfor k = 1:length(pop)
        fitnessVec(k) = objective_function(pop(k,:));
    end

    %update waitbar
    waitbar((gen-1)/maxGenerations, f, ...
        {'Running Memetic Algorithm Optimisation...', strcat(['
            ↪ Current Generation: ' num2str(gen) ...
        ' Current Iteration: ' num2str(iter)])});

    %calculate percentage of stable results per generation
    if calculateStableSystems && (gen == 1)
        numberOfStableSystems = zeros(popSize,1);
        percentageOfStableSystems = zeros(maxGenerations,1);
    end
    if calculateStableSystems
        numberOfStableSystems = 0;
        parfor k = 1:popSize
            numberOfStableSystems = numberOfStableSystems +
                ↪ is_solution_feasible(pop(k,:), geneBounds,
                ↪ true);
        end
        percentageOfStableSystems(gen) = (sum(
            ↪ numberOfStableSystems)/popSize) * 100;
    end

    %!run operators in parallel!
    %PRESERVE ELITE
    OP1 = parfeval(@get_elites, 2, pop,fitnessVec,eliteCount);
    %CROSSOVER
    OP2 = parfeval(@crossover, 2, pop,fitnessVec,crossoverRate);
    %MUTATION
    OP3 = parfeval(@mutation, 2, pop,mutationRate,geneBounds);
    %LOCAL SEARCH
    OP4 = parfeval(@local_search, 2 , pop,fitnessVec,geneBounds,
        ↪ localSearchRate);


    %retrieve results from parallel operation
```

```
[R1, R2] = fetchOutputs([OP1, OP2, OP3, OP4], 'UniformOutput
    ↪ ', false);
elites = cell2mat(R1(1));
elitesFitVec = cell2mat(R2(1));
xover_offspring = cell2mat(R1(2));
xover_fitVec = cell2mat(R2(2));
mut_offspring = cell2mat(R1(3));
mut_fitVec = cell2mat(R2(3));
ls_offspring = cell2mat(R1(4));
ls_fitVec = cell2mat(R2(4));

%want to see what the elite was from before any operators
    ↪ take place
if gen == 1
    maxFitnessValues(gen) = max(fitnessVec);
    %also compute average
    averageFitnessValues(gen) = sum(fitnessVec)/popSize;
end

%get fittest individual from local search and add to elites
    ↪ vector
%(Lamarckian Model)
if ~isempty(ls_offspring)
    [maxLSFitness, idx] = max(ls_fitVec);
    maxLS = ls_offspring(idx,:);
    if eliteCount == 1
        %replace
        if (maxLSFitness > elitesFitVec)
            elites = maxLS;
            elitesFitVec = maxLSFitness;
        end
    elseif eliteCount == 0
        elites = maxLS;
        elitesFitVec = maxLSFitness;
    else
        %find max elite and replace
        [maxEliteFitness, idx] = max(elitesFitVec);
        maxElite = elites(idx,:);
        if maxLSFitness > maxEliteFitness
            elites(idx,:) = maxLS;
            elitesFitVec(idx) = maxLSFitness;
        end
    end
end

%SURVIVOR SELECTION - cull extended population back to
    ↪ popSize,
%applying elitism
extended_pop = [pop; xover_offspring; mut_offspring;
    ↪ ls_offspring];
extpop_fitness = [fitnessVec; xover_fitVec; mut_fitVec;
    ↪ ls_fitVec];
```

```
    [newPop, newPopFitness] = survivor_selection(extended_pop,
        ↪ extpop_fitness, elites, elitesFitVec, popSize);

    %assign new population to be used in the next generation
    pop = newPop;
    fitnessVec = newPopFitness;

    %store the max fitness value for plotting purposes
    maxFitnessValues(gen+1) = max(fitnessVec);
    averageFitnessValues(gen+1) = sum(fitnessVec)/popSize;

    gen = gen + 1;
end

%calculte percentage of stable systems across all generations
if calculateStableSystems
    percentageOfStableSystemsAllGenerations = sum(
        ↪ percentageOfStableSystems)/maxGenerations;
    disp(strcat(['Percentage of Stable Systems across all
        ↪ generations: ' ...
         num2str(percentageOfStableSystemsAllGenerations) '%']))
end

%provide update that algorithm is finalising
waitbar(1, f, 'Finalising...');

%build string for legend
legendStrBuilder(iter,:) = strcat(['Iteration ' num2str(iter)]);

figure(1)
plot(1:gen, maxFitnessValues)
title('Plot of Max Fitness vs. Generation Number')
xlabel('Generation Number')
ylabel('Max Fitness')
if maxGenerations > 1
    xlim([1 maxGenerations])
end
hold on

figure(2)
plot(1:gen, averageFitnessValues)
title('Plot of Average Fitness vs. Generation Number')
xlabel('Generation Number')
ylabel('Average Fitness')
if maxGenerations > 1
    xlim([1 maxGenerations])
end
hold on

%get the fittest individual at the end - this will be the
    ↪ optimal value
[optimisedParametersFitness,idx] = max(fitnessVec);
```

```
optimisedParameters = pop(idx,:);
%display
disp(strcat(['Optimal PID Parameters: ' num2str(
    ↪ optimisedParameters) '']));
disp(strcat(['Fitness Value: ' num2str(
    ↪ optimisedParametersFitness)]));
newline;

if pidtest(optimisedParameters, false, true)
    stableStr = 'Yes';
else
    stableStr = 'No';
end
disp(strcat(['System Stable: ' stableStr]));

bestIndividualsOfEachRun(iter,:) = optimisedParameters;
bestIndividualsOfEachRunFitness(iter) =
    ↪ optimisedParametersFitness;

%log elapsed time
t = toc;
runTimeTracker = runTimeTracker + t;
end

figure(1)
legend(legendStrBuilder, 'Location', 'northwest')
figure(2)
legend(legendStrBuilder, 'Location', 'northwest')

disp('------------------------------------------------------------')

%Find the median iteration, display and plot
[~, med_idx] = min(abs(bestIndividualsOfEachRunFitness-median(
    ↪ bestIndividualsOfEachRunFitness)));
disp(strcat(['The run representative of the median of all
    ↪ iterations is Run ' num2str(med_idx)]));
disp(strcat(['Optimal PID Parameters: ' num2str(
    ↪ bestIndividualsOfEachRun(med_idx,:)) '']));
disp(strcat(['Fitness Value: ' num2str(
    ↪ bestIndividualsOfEachRunFitness(med_idx,:))]));
figure(3)
pidtest(bestIndividualsOfEachRun(med_idx,:), true, true);
title('Step Response of Median Individual')

%print average time taken for each run
newline;
disp(strcat(['Average Iteration Runtime: ' num2str(
    ↪ runTimeTracker/iterations) 's']))

%delete waitbar
delete(f)
```

D.2 The objective_function.m

The function objective_function.m is the main script to be called to start and run the Memetic Algorithm.

Listing D.2: Function to evaluate objective function and calculate fitness.

```
function [fitness, stable] = objective_function(K_arr, testRH)
%UNTITLED Calculates the fitness of the given gene (PID
    ↪ parameters)
%   Detailed explanation goes here
    global unstableCount
    global stableCount

    %load variables from the saved .mat file
    load('ws_vars', 'dt', 'T', 'system_type', 'channel')

    if nargin < 2
        testRH = true;
    end

    t = 0:dt:T;

    if strcmp(system_type, 'SISO')
        [ClosedLoopSys, K] = build_control_system(K_arr);
        [y,t] = step(ClosedLoopSys,t);  %step response

        %control effort
        u = lsim(K,1-y,t);

        %Combute Objective Function -
        %ITE: integral of absolute error and squared control
            ↪ energy
        J = T^2.*(dt.*sum(abs(1-y(:)))) + (dt.*sum(u(:).^2));
    elseif strcmp(system_type, 'MIMO')
        if channel == 1
            r1 = ones(length(t), 1);
            r2 = zeros(length(t), 1);
        elseif channel == 2
            r1 = zeros(length(t), 1);
            r2 = ones(length(t), 1);
        end

        %build and get system
        [ClosedLoopSys, K] = build_control_system(K_arr);

        K1 = K(1);
        K2 = K(2);
```

```
        %simulate system
        y = lsim(ClosedLoopSys,[r1 r2], t);

        y1 = y(:,1);      %channel 1 output
        y2 = y(:,2);      %channel 2 output

        %errors
        e1 = r1-y1;
        e2 = r2-y2;

        %control efforts
        u1 = lsim(K1,e1,t);    %channel 1 PID control effort
        u2 = lsim(K2,e2,t);    %channel 2 PID control effort

        J = T^2.*(dt.*(sum(abs(e1))+sum(abs(e2)))) + (dt.*(sum(
            ↪ u1.^2)+sum(u2.^2)));
    end

    %Test if unstable using Routh-Hurwitz Criterion
    if testRH
        if strcmp(system_type, 'SISO')
            stable = is_system_stable(ClosedLoopSys);
        elseif strcmp(system_type, 'MIMO')
            stable = is_system_stable(ClosedLoopSys(1,channel))
                ↪ && is_system_stable(ClosedLoopSys(2,channel));
        end
    else
        stable = true;
    end

    %penalise if unstable, calculate and return fitness
    if testRH
        if ~stable
            J = J + 100000;      %large penalty for unstable
                ↪ system
            unstableCount = unstableCount + 1;
        else
            stableCount = stableCount + 1;
        end
    end
    fitness = 30000/J;
end
```

Appendix E

Operator Functions

E.1 The `crossover.m` Operator

The function `crossover.m` is an operator function that performs crossover on given parents.

Listing E.1: Function to perform crossover.

```
%modifies offspring using a modified whole arithmetic
    ↪ recombination
%approach
function [offspring, offspring_fitVec] = crossover(pop, fitVec,
    ↪ crossover_rate)
    %setup

    number_of_offspring = floor((crossover_rate * length(pop))
        ↪ /2) * 2; %always round to even number
    if (number_of_offspring < 2)
        number_of_offspring = 2;
    end
    numGenes = size(pop, 2);

    offspring = [];
    offspring_fitVec = zeros(number_of_offspring, 1);

    parfor k = 1:number_of_offspring/2
        %start acrossover here
        %randomly select two parents x_pm and x_pn
        x_pm = parent_selection(pop, fitVec);
        x_pn = parent_selection(pop, fitVec);
        %initialise offspring
        x_o1 = zeros(1, numGenes);
        x_o2 = zeros(1, numGenes);

        %Split chromosome at the randomly chosen index, a, where
            ↪ as is in
        %the inverval [1,u], where u is numGenes
        a = randi(numGenes, 1);

        %store the same genes from respective parents for each
            ↪ child up to, but
        %not indluding the specified index
        if (a~=1)
            x_o1(1:a-1) = x_pm(1:a-1);
            x_o2(1:a-1) = x_pn(1:a-1);
        end

        %calculate the x_o1 and x_o2 for each child and store at
            ↪ the index 'a'
        B = rand(1,1);
```

```
            x_o1(a) = (1 - B)*x_pm(a) + B*x_pn(a);
            x_o2(a) = (1 - B)*x_pn(a) + B*x_pm(a);

            %swap genes of parents after index 'a' for both
               ↪ offspring
            if (a < numGenes)
                x_o1(a+1:end) = x_pm(a+1:end);
                x_o2(a+1:end) = x_pn(a+1:end);
            end

            offspring = [offspring; x_o1];
            offspring = [offspring; x_o2];
        end

        parfor k = 1:number_of_offspring
            offspring_fitVec(k) = objective_function(offspring(k,:))
               ↪ ;
        end
    end
```

E.2 The `mutation.m` Operator

The function `mutation.m` is an operator function that performs mutation on randomly selected genes from a population.

Listing E.2: Function to perform mutation.

```
function [offspring, offspring_fitVec] = mutation(pop,
    ↪ mutation_rate, gene_bounds)
%MUTATION Perform mutation on specified amount of population
    %Change allele values randomly within its domain
    %i.e.
    %<x1, x2, ..., xn)> -> <x'1, x'2, ..., x'n>
    %where xi, x'i is an element of [Li, Ui]
    %where Ui and Li are upper and lower boundaries respectively

    number_of_offspring = ceil(mutation_rate * size(pop,1)); %
        ↪ use ceiling as num of mutations should never be 0 if
        ↪ rate is non-zero
    numGenes = size(pop, 2);
    offspring = zeros(number_of_offspring, numGenes);
    offspring_fitVec = zeros(number_of_offspring, 1);

    disp(number_of_offspring);

    %select random genes to mutate according to given
        ↪ mutation_rate
    for k = 1:number_of_offspring
        %select random chromosome
        offspring(k,:) = pop(randi(size(pop,1)),:);
        %mutate chromosome by selecting random gene:
        x = randi(numGenes, 1);
        geneBounds = cell2mat(gene_bounds(x));
        L_k = geneBounds(1);     %L_k - lower bound
        U_k = geneBounds(2);     %U_k - upper bound
        offspring(k,randi(numGenes)) = L_k + rand*(U_k - L_k);
            ↪ %mutate gene
    end

    %calculate fitness in parallel
    parfor k = 1:number_of_offspring
        offspring_fitVec(k) = objective_function(offspring(k,:))
            ↪ ;
    end


end
```

E.3 The local_search.m Operator

The function local_search.m is an operator function that handles all the calls to the chosen local search method.

Listing E.3: Function to call LS algorithm.

```
function [offspring, offspring_fitVec] = local_search(pop,
    ↪ fitVec, gene_bounds, rate)
%LOCAL_SEARCH Method for handling local search calls

    %define parameters for the LS algorithm
    maxf = 3;
    maxs = 3;
    expf = 2;
    conf = 0.5;
    max_iter = 100;

    ls_count = ceil(rate * length(pop));

    %get pool of candidates for local search
    [candidatePool, poolFitness] = get_elites(pop, fitVec,
        ↪ ls_count);

    offspring = [];
    offspring_fitVec = [];

    parfor k = 1:ls_count
        [result, resultFitness] =  ...
            solis_wets_LS(candidatePool(k,:), poolFitness(k),...
            maxf, maxs, expf, conf, gene_bounds, max_iter,
                ↪ ls_count);
        if ~isempty(result)
            offspring = [offspring; result];
            offspring_fitVec = [offspring_fitVec; resultFitness
                ↪ ];
        end
    end
end
```

E.4 The `solis_wets_LS.m` Operator

The function `solis_wets_LS.m` is an implementation of the modified Solis & Wets algorithm.

Listing E.4: LS algorithm.

```
function [offspring, offspring_fitVec] = solis_wets_LS(
    ↪ candidateToImprove,...
     candidateToImproveFitness, maxF, maxS, expf, conf,
        ↪ gene_bounds, max_iter, ls_count)
%SOLIS_WETS_LS implements a modified version of Solis & Wet's LS
    ↪  algorithm

     load('solis_wets_ls_vars', 'rho', 'threshold')

     offspring = [];
     offspring_fitVec = [];

     %set bias to 0
     b = 0;

     %initialise failure and success trackers
     failures = 0;
     successes = 0;

     %set the new candidate to be the candidate to improve
        ↪ initially
     newCandidate = candidateToImprove;
     newBestIndividual = [];
     newBestIndividualFitness = [];

     iter = 0;
     while ((rho > threshold) && (iter < max_iter))
         %try to improve candidate
         D = normrnd(b, rho);
         newCandidate = newCandidate + D;
         [newCandidateFitness, stable] = objective_function(
            ↪ newCandidate);
         if (newCandidateFitness > candidateToImproveFitness) &&
            ↪ stable && is_solution_feasible(newCandidate,
            ↪ gene_bounds, false)
             failures = 0;
             successes = successes + 1;
             b = 0.5*D + 0.2*b;
             newBestIndividual = newCandidate;
             newBestIndividualFitness = newCandidateFitness;
         else
             newCandidate = candidateToImprove + D;
```

```
            [newCandidateFitness, stable] = objective_function(
               ↪ newCandidate);
            if (newCandidateFitness > candidateToImproveFitness)
               ↪  && stable && is_solution_feasible(
               ↪ newCandidate, gene_bounds, false)
                failures = 0;
                successes = successes + 1;
                b = b - 0.4*D;
                newBestIndividual = newCandidate;
                newBestIndividualFitness = newCandidateFitness;
            else
                failures = failures + 1;
                successes = 0;
                b = 0.5*b;
            end
        end

        if successes >= maxS
            failures = 0;
            successes = 0;
            rho = expf * rho;
        end
        if failures >= maxF
            failures = 0;
            successes = 0;
            rho = conf * rho;
        end
        iter = iter + 1;
    end

    %if the new individual is better than before the local
       ↪ search, shrink
    %threshold and return the new individual
    if ~isempty(newBestIndividual)
        if (newBestIndividualFitness > candidateToImproveFitness
           ↪ ) && is_solution_feasible(newBestIndividual,
           ↪ gene_bounds, false)
            threshold = (0.2/ls_count)*threshold;
        end
        offspring = newBestIndividual;
        offspring_fitVec = newBestIndividualFitness;
    end
    rho = (5/ls_count)*rho;

    save('solis_wets_ls_vars', 'rho', 'threshold')

end
```

Appendix F

Helper Functions

F.1 The `build_control_system.m` Helper Function

Listing F.1: Helper function to help build control system programmatically.

```
function [sys, K] = build_control_system(K_arr)
%BUILD_CONTROL_SYSTEM Builds a control system for the given PID
    ↪ parameters
%   Returns the closed loop system and K, a 1x1 representing the
    ↪  PID
%   controller if the system type is SISO, and a 1x2
    ↪ representing the PID
%   controller for both channels if the system type is MIMO

    %load variables from the saved .mat file
    load('ws_vars', 'G', 'system_type', 'channel', 'K_fixed')

    if strcmp(system_type, 'SISO')
        K = pid(K_arr(1), K_arr(2), K_arr(3), 0.001);
        Loop = series(K,G);
        sys = feedback(Loop, 1);
    elseif strcmp(system_type, 'MIMO')
        if channel == 1
            K1 = pid(K_arr(1), K_arr(2), K_arr(3), 0.001);
            K2 = pid(K_fixed(1), K_fixed(2), K_fixed(3), 0.001);
        elseif channel == 2
            K1 = pid(K_fixed(1), K_fixed(2), K_fixed(3), 0.001);
            K2 = pid(K_arr(1), K_arr(2), K_arr(3), 0.001);
        end

        K = [K1 K2];

        %cascade PIDs BEFORE cross-coupling paths
        y1_pid_loop = series(K1, G, 1, 1);
        y2_pid_loop = series(K2, G, 1, 2);

        sys_with_pids = [y1_pid_loop y2_pid_loop];

        fb = tf(ss(eye(2)));    %generate identity state space
            ↪ for unity feedback lines
        sys = feedback(sys_with_pids, fb, -1);  %negative
            ↪ feedback for both i/o lines
    end
end
```

F.2 The get_elites.m Helper Function

Listing F.2: Helper function to get elites from the given population.

```
function [eliteVec,eliteFitVec] = get_elites(pop,fitVec,
    ↪ eliteCount)
%GET_ELITES gets eliteCount number of elites from the given
    ↪ population
    if (eliteCount > 0)
        tmpPop = pop;
        tmpFitVec = fitVec;
        eliteVec = zeros(eliteCount, size(pop,2));
        eliteFitVec = zeros(eliteCount, 1);
        for k = 1:eliteCount
            [~, I] = max(tmpFitVec);
            eliteVec(k,:) = tmpPop(I,:);
            eliteFitVec(k) = tmpFitVec(I);
            tmpFitVec(I) = [];
            tmpPop(I,:) = [];
        end
        %no need to hang onto these temp vectors when done, so
            ↪ free them
        %from memory
        clear tmpPop tmpFitVec
    else
        eliteVec = [];
        eliteFitVec = [];
    end
end
```

F.3 The `is_solution_feasible.m` Helper Function

Listing F.3: Helper function to test if a potential solution is within bounds and is stable. Useful in LS.

```
function feasible = is_solution_feasible(K_arr, gene_bounds,
    ↪ check_stability)
%IS_SOLUTION_FEASIBLE: Checks if the solution meets all criteria
    ↪  to be
%classed as a feasible solution
%   Check if the given solution is within bounds and the
%system produced is stable

    load('ws_vars', 'channel', 'system_type')

    %check if the solution is within bounds
    for k = 1:length(gene_bounds)
        geneBounds = cell2mat(gene_bounds(k));
        L = geneBounds(1);      %lower bound
        U = geneBounds(2);      %upper bound
        if (K_arr(k) < L) || (K_arr(k) > U)
            feasible = false;
            return
        end
    end

    %check if stable
    if check_stability
        ClosedLoop = build_control_system(K_arr);
        if strcmp(system_type, 'SISO')
            feasible = is_system_stable(ClosedLoop);
        elseif strcmp(system_type, 'MIMO')
            feasible = is_system_stable(ClosedLoop(1,channel))
               ↪ && is_system_stable(ClosedLoop(2,channel));
        end
    else
        feasible = true;
    end
end
```

F.4 The is_system_stable.m Helper Function

Listing F.4: Helper function to test if a potential solution is stable.

```
function stable = is_system_stable(sys)
%GENERATE_RH_TABLE Checks if solution is stable using Routh-
    ↪ Hurwitz criterion.
    [~,den] = tfdata(sys);
    den = cell2mat(den);
    rhTableAsDbl = rhStabilityCriterion(den);    %generate RH
       ↪ table using script by Farzad Sagharchi
    %Check change in signs in first column - if sign changes,
       ↪ system
    %unstable
    stable = true;
    for i = 1:length(den) - 1
        if sign(rhTableAsDbl(i,1)) * sign(rhTableAsDbl(i+1,1))
           ↪ == -1
            stable = false;
            break
        end
    end
end
```

F.5 The `parent_selection.m` Helper Function

Implements RWS to select parents.

Listing F.5: Helper function to implement RWS to select parents.

```
%select parent from pop using roulette wheel selection.
function parent = parent_selection(pop, fitVec)
    %calculate the sum of all fitness values in the population
    totalFitness = sum(fitVec);

    %generate random number in the interval of [0, totalFitness]
    r = totalFitness*rand(1,1);

    %loop through population and sum the fitness for each
       ↪ individual. Stop
    %and return the individual when the sum of fitness values
       ↪ becomes greater
    %than, or equal to, the random number r
    partialFitness = 0;
    for k=length(pop):-1:1
        partialFitness = partialFitness + fitVec(k);
        if (partialFitness >= r)
            parent = pop(k,:);
            break
        end
    end
end
```

F.6 The pidtest.m Helper Function

Listing F.6: Helper function used for building and testing a PID controller programmatically.

```
function [stable] = pidtest(K_arr, plot_response, print_stepinfo
    ↪ )
    global dt

    load('ws_vars', 'channel', 'system_type')

    ClosedLoop = build_control_system(K_arr);
    t = 0:dt:50;

    %stability check
    if strcmp(system_type, 'SISO')
        stable = is_system_stable(ClosedLoop);
    elseif strcmp(system_type, 'MIMO')
        stable = is_system_stable(ClosedLoop(1,channel)) &&
            ↪ is_system_stable(ClosedLoop(2,channel));
    end


    if plot_response
        if strcmp(system_type, 'SISO')
            step(ClosedLoop,t)
            h = findobj(gcf,'type','line');
            set(h,'linewidth',2);
            drawnow
        elseif strcmp(system_type, 'MIMO')
            if channel == 1
                r1 = ones(length(t), 1);
                r2 = zeros(length(t), 1);
            elseif channel == 2
                r1 = zeros(length(t), 1);
                r2 = ones(length(t), 1);
            end
            lsim(ClosedLoop,[r1 r2], t);
        end
    end

    %print system step info
    if print_stepinfo
        disp('Step_Analysis');
        if strcmp(system_type, 'SISO')
            disp(stepinfo(ClosedLoop));
        elseif strcmp(system_type, 'MIMO')
            disp(stepinfo(ClosedLoop(channel,channel)))
        end
    end
end
```

F.7 The `rhStabilityCriterion.m` Helper Function

Created by Sagharchi Sagharchi, and modified to return Routh table only.

Listing F.7: Helper function to return routh table.

```
%% Routh-Hurwitz stability criterion
%
%  The Routh-Hurwitz stability criterion is a necessary (and
   ↪ frequently
%  sufficient) method to establish the stability of a single-
   ↪ input,
%  single-output(SISO), linear time invariant (LTI) control
   ↪ system.
%  More generally, given a polynomial, some calculations using
   ↪ only the
%  coefficients of that polynomial can lead us to the conclusion
   ↪  that it
%  is not stable.

%  Instructions
%  ------------
%
%  in this program you must give your system coefficients and
   ↪ the
%  Routh-Hurwitz table would be shown
%
%   Farzad Sagharchi ,Iran
%   2007/11/12
%   Modified by Aaron Coutts
%   9/10/2021



function rhTable = rhStabilityCriterion(coeffVector)
    %% Initialization

    % Taking coefficients vector and organizing the first two
       ↪ rows
    ceoffLength = length(coeffVector);
    rhTableColumn = round(ceoffLength/2);

    %  Initialize Routh-Hurwitz table with empty zero array
    rhTable = zeros(ceoffLength,rhTableColumn);

    %  Compute first row of the table
    rhTable(1,:) = coeffVector(1,1:2:ceoffLength);

    %  Check if length of coefficients vector is even or odd
    if (rem(ceoffLength,2) ~= 0)
```

```
        % if odd, second row of table will be
        rhTable(2,1:rhTableColumn - 1) = coeffVector(1,2:2:
            ↪ ceoffLength);
    else
        % if even, second row of table will be
        rhTable(2,:) = coeffVector(1,2:2:ceoffLength);
    end

    %% Calculate Routh-Hurwitz table's rows

    %  Set epss as a small value
    epss = 0.01;

    %  Calculate other elements of the table
    for i = 3:ceoffLength

        %  special case: row of all zeros
        if rhTable(i-1,:) == 0
            order = (ceoffLength - i);
            cnt1 = 0;
            cnt2 = 1;
            for j = 1:rhTableColumn - 1
                rhTable(i-1,j) = (order - cnt1) * rhTable(i-2,
                    ↪ cnt2);
                cnt2 = cnt2 + 1;
                cnt1 = cnt1 + 2;
            end
        end

        for j = 1:rhTableColumn - 1
            %  first element of upper row
            firstElemUpperRow = rhTable(i-1,1);

            %  compute each element of the table
            rhTable(i,j) = ((rhTable(i-1,1) * rhTable(i-2,j+1))
                ↪ - ....
                (rhTable(i-2,1) * rhTable(i-1,j+1))) /
                    ↪ firstElemUpperRow;
        end


        %  special case: zero in the first column
        if rhTable(i,1) == 0
            rhTable(i,1) = epss;
        end
    end
end

% %%  Compute number of right hand side poles(unstable poles)
% %   Initialize unstable poles with zero
% unstablePoles = 0;
%
```

```
%%   Check change in signs
% for i = 1:ceoffLength - 1
%     if sign(rhTable(i,1)) * sign(rhTable(i+1,1)) == -1
%         unstablePoles = unstablePoles + 1;
%     end
% end
%
%%   Print calculated data on screen
% fprintf('\n Routh-Hurwitz Table:\n')
% rhTable
%
%%   Print the stability result on screen
% if unstablePoles == 0
%     fprintf('~~~~~> it is a stable system! <~~~~~\n')
% else
%     fprintf('~~~~~> it is an unstable system! <~~~~~\n')
% end
%
% fprintf('\n Number of right hand side poles =%2.0f\n',
    ↪ unstablePoles)
%
% reply = input('Do you want roots of system be shown? Y/N ', 's
    ↪ ');
% if reply == 'y' || reply == 'Y'
%     sysRoots = roots(coeffVector);
%     fprintf('\n Given polynomial coefficients roots :\n')
%     sysRoots
% end
```

F.8 The survivor_selection.m Helper Function

Selects next generation.

Listing F.8: Helper function to select next generation.

```
function [next_generation, nextgen_fitness] = survivor_selection
    ↪ (extended_pop, ...
    extpop_fitness, elites, elites_fitness, pop_size)
    %SURVIVOR_SELECTION Perform selection to reduce
    %population size back to pop_size

    %propagate elites to the next generation
    next_generation = elites;
    nextgen_fitness = elites_fitness;

    %perform binary tournament selection
    parfor k = 1:pop_size-size(elites,1)
        [winner, winnerFitness] = tournament_selection(
            ↪ extended_pop, extpop_fitness, 2);
        next_generation = [next_generation; winner];
        nextgen_fitness = [nextgen_fitness; winnerFitness];
    end
end
```

F.9 The `tournament_selection.m` Helper Function

Implements tournament selection. To be used for selecting the next generation.

Listing F.9: Helper function to implement tournament selection.

```
function [next_generation, nextgen_fitness] = survivor_selection
    ↪ (extended_pop, ...
    extpop_fitness, elites, elites_fitness, pop_size)
    %SURVIVOR_SELECTION Perform selection to reduce
    %population size back to pop_size

    %propagate elites to the next generation
    next_generation = elites;
    nextgen_fitness = elites_fitness;

    %perform binary tournament selection
    parfor k = 1:pop_size-size(elites,1)
        [winner, winnerFitness] = tournament_selection(
            ↪ extended_pop, extpop_fitness, 2);
        next_generation = [next_generation; winner];
        nextgen_fitness = [nextgen_fitness; winnerFitness];
    end
end
```

Appendix G

Test Scripts & Functions

G.1 The `TestLinear1DOF.m` Script

The test script written to test and verify 1DOF systems for pitch and yaw control of the TRMS.

Listing G.1: Test 1DOF system and print results.

```
clear variables
close all
clc

%transfer functions
pitch_tf = tf([0.01657 0.4194 2.454],[1 1.487 4.403 5.449]);
yaw_tf = tf([0.0009881 0.03361 0.4065],[1 1.345 0.4568 0.3826]);

%
    ↪ ------------------------------------------------------------------------
    ↪
optimisation_type = 'MA';   %change this depending on whether
    ↪ testing GA or MA

if strcmp(optimisation_type, 'MA')
    Pitch_Kp = 0.039411;
    Pitch_Ki = 0.96991;
    Pitch_Kd = 0.16499;
    Yaw_Kp = 0.31597;
    Yaw_Ki = 0.15789;
    Yaw_Kd = 0.42171;
elseif strcmp(optimisation_type, 'GA')
    Pitch_Kp = 0.451;
```

```
    Pitch_Ki = 1.8701;
    Pitch_Kd = 0.4959;
    Yaw_Kp = 0.13136;
    Yaw_Ki = 0.27323;
    Yaw_Kd = 0.73833;
end

dt = 0.05;
T = 50;
t = 0:dt:T;

%define test parameters for Prasad et. al.'s testing parameters
testParameters = struct;
testParameters.step = 1;
testParameters.sineAmp = 1;
testParameters.sineFreq = 0.025;
testParameters.sqrAmp = 1;
testParameters.sqrFreq = 0.025;
testParameters.initialCondition = 0;

%test pitch
K_arr = [Pitch_Kp Pitch_Ki Pitch_Kd];
[sys,controller] = build_SISO_control_system(pitch_tf, K_arr);

[riseTime, settlingTime, overshoot, error, controlEffort] =
    ↪ systest_SISO(sys, controller, t, testParameters, true);
    disp(strcat(['---------- ' optimisation_type ' test results
        ↪ for Pitch Control ----------']))
    disp(strcat(['Rise time: ' num2str(riseTime)]))
    disp(strcat(['Settling time: ' num2str(settlingTime)]))
    disp(strcat(['Percentage overshoot: ' num2str(overshoot)]))
    disp(strcat(['Sum of Absolute Error for Step Input: '
        ↪ num2str(error.step)]))
    disp(strcat(['Sum of Absolute Error for Sine Input: '
        ↪ num2str(error.sine)]))
    disp(strcat(['Sum of Absolute Error for Square Input: '
        ↪ num2str(error.sqr)]))
    disp(strcat(['Sum of Absolute Control Effort for Step Input:
        ↪ ' num2str(controlEffort.step)]))
    disp(strcat(['Sum of Absolute Control Effort for Sine Input:
        ↪ ' num2str(controlEffort.sine)]))
    disp(strcat(['Sum of Absolute Control Effort for Square
        ↪ Input: ' num2str(controlEffort.sqr)]))

%test yaw
K_arr = [Yaw_Kp Yaw_Ki Yaw_Kd];
[sys, controller] = build_SISO_control_system(yaw_tf, K_arr);
[riseTime, settlingTime, overshoot, error, controlEffort] =
    ↪ systest_SISO(sys, controller, t, testParameters, true);
    disp(strcat(['---------- ' optimisation_type ' test results
        ↪ for Yaw Control ----------']))
    disp(strcat(['Rise time: ' num2str(riseTime)]))
```

```
disp(strcat(['Settling time: ' num2str(settlingTime)]))
disp(strcat(['Percentage overshoot: ' num2str(overshoot)]))
disp(strcat(['Sum of Absolute Error for Step Input: '
    ↪ num2str(error.step)]))
disp(strcat(['Sum of Absolute Error for Sine Input: '
    ↪ num2str(error.sine)]))
disp(strcat(['Sum of Absolute Error for Square Input: '
    ↪ num2str(error.sqr)]))
disp(strcat(['Sum of Absolute Control Effort for Step Input:
    ↪ ' num2str(controlEffort.step)]))
disp(strcat(['Sum of Absolute Control Effort for Sine Input:
    ↪ ' num2str(controlEffort.sine)]))
disp(strcat(['Sum of Absolute Control Effort for Square
    ↪ Input: ' num2str(controlEffort.sqr)]))
```

G.2 The `TestLinear2DOF.m` Script

The test script written to test and verify 2DOF systems for pitch and yaw control of the TRMS.

Listing G.2: Test 2DOF system and print results.

```
clear variables
close all
clc

%transfer functions
G11 = tf([0.01657 0.4194 2.454],[1 1.487 4.403 5.449]);     %
    ↪ pitch main
G12 = tf([0.04986 0.0962],[1 0.2377 4.902]);     %yaw coupling
    ↪ into pitch
G21 = tf([0.02248 0.4527],[1 0.4099 0.2181]);     %pitch coupling
    ↪  into yaw
G22 = tf([0.0009881 0.03361 0.4065],[1 1.345 0.4568 0.3826]);
    ↪     %yaw main

G = [G11 G12;
     G21 G22];

%
    ↪ ----------------------------------------------------------------------------
    ↪
optimisation_type = 'MA';    %change this depending on whether
    ↪ testing GA or MA

if strcmp(optimisation_type, 'GA')
    Pitch_Kp = 0.59458;
    Pitch_Ki = 0.60207;
    Pitch_Kd = 0.50529;
    Yaw_Kp = 0.31266;
    Yaw_Ki = 0.22875;
    Yaw_Kd = 0.99066;
elseif strcmp(optimisation_type, 'MA')
    Pitch_Kp = 0.62141;
    Pitch_Ki = 0.89421;
    Pitch_Kd = 0.42039;
    Yaw_Kp = 0.36102;
    Yaw_Ki = 0.30189;
    Yaw_Kd = 0.71477;
end

dt = 0.05;
T = 50;
t = 0:dt:T;
```

```
%define test parameters for Juang et. al.'s testing parameters
testParametersChannel1 = struct;
testParametersChannel1.step = 1;
testParametersChannel1.sineAmp = 0.5;
testParametersChannel1.sineFreq = 0.025;
testParametersChannel1.sqrAmp = 0.5;
testParametersChannel1.sqrFreq = 0.025;
testParametersChannel1.initialCondition = 0;

testParametersChannel2 = struct;
testParametersChannel1.step = 0.2;
testParametersChannel1.sineAmp = 0.2;
testParametersChannel1.sineFreq = 0.025;
testParametersChannel1.sqrAmp = 0.2;
testParametersChannel1.sqrFreq = 0.025;
testParametersChannel1.initialCondition = 0;

%test
K1_arr = [Pitch_Kp Pitch_Ki Pitch_Kd];
K2_arr = [Yaw_Kp Yaw_Ki Yaw_Kd];
[sys, controllers] = build_MIMO_control_system(G, K1_arr, K2_arr
    ↪ );

[riseTime, settlingTime, overshoot, error, controlEffort] = ...
    systest_MIMO(sys, controllers, t, testParametersChannel1,
        ↪ testParametersChannel1, true);
disp(strcat(['---------- ' optimisation_type  ' test results for
    ↪ Pitch Control ----------']))
disp(strcat(['Rise time: ' num2str(riseTime.channel1)]))
disp(strcat(['Settling time: ' num2str(settlingTime.channel1)]))
disp(strcat(['Percentage overshoot: ' num2str(overshoot.channel1
    ↪ )]))
disp(strcat(['Sum of Absolute Error for Step Input: ' num2str(
    ↪ error.channel1.step)]))
disp(strcat(['Sum of Absolute Error for Sine Input: ' num2str(
    ↪ error.channel1.sine)]))
disp(strcat(['Sum of Absolute Error for Square Input: ' num2str(
    ↪ error.channel1.sqr)]))
disp(strcat(['Sum of Absolute Control Effort for Step Input: '
    ↪ num2str(controlEffort.channel1.step)]))
disp(strcat(['Sum of Absolute Control Effort for Sine Input: '
    ↪ num2str(controlEffort.channel1.sine)]))
disp(strcat(['Sum of Absolute Control Effort for Square Input: '
    ↪  num2str(controlEffort.channel1.sqr)]))

disp(strcat(['---------- ' optimisation_type  ' test results for
    ↪ Yaw Control ----------']))
disp(strcat(['Rise time: ' num2str(riseTime.channel2)]))
disp(strcat(['Settling time: ' num2str(settlingTime.channel2)]))
disp(strcat(['Percentage overshoot: ' num2str(overshoot.channel2
    ↪ )]))
```

```
disp(strcat(['Sum of Absolute Error for Step Input: ' num2str(
    ↪ error.channel2.step)]))
disp(strcat(['Sum of Absolute Error for Sine Input: ' num2str(
    ↪ error.channel2.sine)]))
disp(strcat(['Sum of Absolute Error for Square Input: ' num2str(
    ↪ error.channel2.sqr)]))
disp(strcat(['Sum of Absolute Control Effort for Step Input: '
    ↪ num2str(controlEffort.channel2.step)]))
disp(strcat(['Sum of Absolute Control Effort for Sine Input: '
    ↪ num2str(controlEffort.channel2.sine)]))
disp(strcat(['Sum of Absolute Control Effort for Square Input: '
    ↪  num2str(controlEffort.channel2.sqr)]))
```

G.3 The build_SISO_control_system.m Script

Helper function for building a SISO control system programmatically for given PID parameters.

Listing G.3: Builds SISO Control System.

```
function [sys, K] = build_SISO_control_system(G, K_arr)
%BUILD_CONTROL_SYSTEM
%   Returns the closed loop system and K, a 1x1 representing the
    ↪ PID
%   controller for the given system
    K = pid(K_arr(1), K_arr(2), K_arr(3), 0.001);
    Loop = series(K,G);
    sys = feedback(Loop, 1);
end
```

G.4 The `build_MIMO_control_system.m` Script

Helper function for building a MIMO control system programmatically for given PID parameters.

Listing G.4: Builds MIMO Control System.

```
function [sys, K] = build_MIMO_control_system(G, K1_arr, K2_arr)
%BUILD_CONTROL_SYSTEM
%   Returns the closed loop system and K, a 1x2 representing the
    ↪ PID
%   controller for both channels
        K1 = pid(K1_arr(1), K1_arr(2), K1_arr(3), 0.001);
        K2 = pid(K2_arr(1), K2_arr(2), K2_arr(3), 0.001);

        K = [K1 K2];

        %cascade PIDs BEFORE cross-coupling paths
        y1_pid_loop = series(K1, G, 1, 1);
        y2_pid_loop = series(K2, G, 1, 2);

        sys_with_pids = [y1_pid_loop y2_pid_loop];

        fb = tf(ss(eye(2)));    %generate identity state space
           ↪ for unity feedback lines
        sys = feedback(sys_with_pids, fb, -1);  %negative
           ↪ feedback for both i/o lines
end
```

G.5 The systest_MIMO.m Script

Helper function for testing, returning and plotting response characteristics for MIMO control systems.

Listing G.5: For testing MIMO systems.

```
function [riseTime, settlingTime, overshoot, error,
    ↪ controlEffort] = ...
    systest_MIMO(sys, controllers, tvec,
        ↪ test_parameters_channel1, test_parameters_channel2,
        ↪ plot_response)
    %
    t = tvec;

    %return system info
    riseTime = struct;
    settlingTime = struct;
    overshoot = struct;

    si = stepinfo(sys);
    riseTime.channel1 = si(1,1).RiseTime;
    settlingTime.channel1 = si(1,1).SettlingTime;
    overshoot.channel1 = si(1,1).Overshoot;
    riseTime.channel2 = si(2,2).RiseTime;
    settlingTime.channel2 = si(2,2).SettlingTime;
    overshoot.channel2 = si(2,2).Overshoot;

    %step response using initial
    error = struct;
    error.channel1 = struct;
    error.channel2 = struct;
    controlEffort = struct;
    controlEffort.channel1 = struct;
    controlEffort.channel2 = struct;

%     %make initial condition vectors
%     x_0 = (test_parameters.initialCondition\ss(sys).C)';
%
%     lsim(ss(sys),test_parameters.step*ones(1,length(t)), t,
    ↪ x_0)

    if test_parameters_channel1.initialCondition == 0 &&
        ↪ test_parameters_channel2.initialCondition == 0
        step_input_1 = (test_parameters_channel1.step*ones(1,
            ↪ length(t)))';
        step_input_2 = (test_parameters_channel1.step*ones(1,
            ↪ length(t)))';
        y_step = lsim(sys, [step_input_1 step_input_2] , t);
```

```
    sine_input_1 = (test_parameters_channel1.sineAmp*sin(2*
        ↪ pi*t*test_parameters_channel1.sineFreq))';
    sine_input_2 = (test_parameters_channel2.sineAmp*sin(2*
        ↪ pi*t*test_parameters_channel2.sineFreq))';
    y_sine = lsim(sys, [sine_input_1 sine_input_2], t);

    sqr_input_1 = (test_parameters_channel1.sqrAmp*square(t,
        ↪ test_parameters_channel1.sqrFreq))';
    sqr_input_2 = (test_parameters_channel2.sqrAmp*square(t,
        ↪ test_parameters_channel2.sqrFreq))';
    y_sqr = lsim(sys, [sqr_input_1 sqr_input_2], t);

    %return sum of absolute error
    error.channel1.step = sum(abs(step_input_1-y_step(:,1)))
        ↪ ;
    error.channel1.sine = sum(abs(sine_input_1-y_sine(:,1)))
        ↪ ;
    error.channel1.sqr = sum(abs(sqr_input_1-y_sqr(:,1)));
    error.channel2.step = sum(abs(step_input_2-y_step(:,2)))
        ↪ ;
    error.channel2.sine = sum(abs(sine_input_2-y_sine(:,2)))
        ↪ ;
    error.channel2.sqr = sum(abs(sqr_input_2-y_sqr(:,2)));

    %return sum of control effort
    effort_step_channel1 = lsim(controllers(1), step_input_1
        ↪ -y_step(:,1), t);
    effort_sine_channel1 = lsim(controllers(1), sine_input_1
        ↪ -y_sine(:,1), t);
    effort_sqr_channel1 = lsim(controllers(1), sqr_input_1-
        ↪ y_sqr(:,1), t);
    effort_step_channel2 = lsim(controllers(2), step_input_2
        ↪ -y_step(:,2), t);
    effort_sine_channel2 = lsim(controllers(2), sine_input_2
        ↪ -y_sine(:,2), t);
    effort_sqr_channel2 = lsim(controllers(2), sqr_input_2-
        ↪ y_sqr(:,2), t);

    controlEffort.channel1.step = sum(abs(
        ↪ effort_step_channel1));
    controlEffort.channel1.sine = sum(abs(
        ↪ effort_sine_channel1));
    controlEffort.channel1.sqr = sum(abs(effort_sqr_channel1
        ↪ ));
    controlEffort.channel2.step = sum(abs(
        ↪ effort_step_channel2));
    controlEffort.channel2.sine = sum(abs(
        ↪ effort_sine_channel2));
    controlEffort.channel2.sqr = sum(abs(effort_sqr_channel2
        ↪ ));
end
```

```
if plot_response
    %plot Pitch channel results
    figure

    subplot(3,2,1)
    plot(t, y_step(:,1), t, step_input_1)
    xlabel('Time (s)')
    ylabel('Amplitude')
    title('Pitch Step Response')
    h = findobj(gcf,'type','line');
    set(h,'linewidth',2);
    drawnow

    subplot(3,2,3)
    plot(t, y_sine(:,1),t, sine_input_1)
    xlabel('Time (s)')
    ylabel('Amplitude')
    title('Pitch Sine Response')
    h = findobj(gcf,'type','line');
    set(h,'linewidth',2);
    drawnow

    subplot(3,2,5)
    plot(t, y_sqr(:,1), t, sqr_input_1)
    xlabel('Time (s)')
    ylabel('Amplitude')
    title('Pitch Square Response')
    h = findobj(gcf,'type','line');
    set(h,'linewidth',2);
    drawnow

    subplot(3,2,2)
    plot(t, effort_step_channel1)
    xlabel('Time (s)')
    ylabel('Amplitude')
    title('Pitch Control Effort for Step Input')
    h = findobj(gcf,'type','line');
    set(h,'linewidth',2);
    drawnow

    subplot(3,2,4)
    plot(t, effort_sine_channel1)
    xlabel('Time (s)')
    ylabel('Amplitude')
    title('Pitch Control Effort for Sine Input')
    h = findobj(gcf,'type','line');
    set(h,'linewidth',2);
    drawnow

    subplot(3,2,6)
    plot(t, effort_sqr_channel1)
```

```
xlabel('Time (s)')
ylabel('Amplitude')
title('Pitch Control Effort for Square Input')
h = findobj(gcf,'type','line');
set(h,'linewidth',2);
drawnow

%plot Yaw channel results
figure

subplot(3,2,1)
plot(t, y_step(:,2), t, step_input_2)
xlabel('Time (s)')
ylabel('Amplitude')
title('Yaw Step Response')
h = findobj(gcf,'type','line');
set(h,'linewidth',2);
drawnow

subplot(3,2,3)
plot(t, y_sine(:,2),t, sine_input_2)
xlabel('Time (s)')
ylabel('Amplitude')
title('Yaw Sine Response')
h = findobj(gcf,'type','line');
set(h,'linewidth',2);
drawnow

subplot(3,2,5)
plot(t, y_sqr(:,2), t, sqr_input_2)
xlabel('Time (s)')
ylabel('Amplitude')
title('Yaw Square Response')
h = findobj(gcf,'type','line');
set(h,'linewidth',2);
drawnow

subplot(3,2,2)
plot(t, effort_step_channel2)
xlabel('Time (s)')
ylabel('Amplitude')
title('Yaw Control Effort for Step Input')
h = findobj(gcf,'type','line');
set(h,'linewidth',2);
drawnow

subplot(3,2,4)
plot(t, effort_sine_channel2)
xlabel('Time (s)')
ylabel('Amplitude')
title('Yaw Control Effort for Sine Input')
h = findobj(gcf,'type','line');
```

```
        set(h,'linewidth',2);
        drawnow

        subplot(3,2,6)
        plot(t, effort_sqr_channel2)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Yaw Control Effort for Square Input')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow
    end
end
```

G.6 The systest_SISO.m Script

Helper function for testing, returning and plotting response characteristics for SISO control systems.

Listing G.6: For testing SISO systems.

```
function [riseTime, settlingTime, overshoot, error,
    ↪ controlEffort] = ...
    systest_SISO(sys, controller, tvec, test_parameters,
        ↪ plot_response)

    t = tvec;

    %return system info
    si = stepinfo(sys);
    riseTime = si.RiseTime;
    settlingTime = si.SettlingTime;
    overshoot = si.Overshoot;

    %step response using initial
    error = struct;
    controlEffort = struct;

%     %make initial condition vectors
%     x_0 = (test_parameters.initialCondition\ss(sys).C)';
%
%     lsim(ss(sys),test_parameters.step*ones(1,length(t)), t,
    ↪ x_0)

    if test_parameters.initialCondition == 0
        step_input = (test_parameters.step*ones(1,length(t)))';
        y_step = lsim(sys, step_input , t);
        sine_input = (test_parameters.sineAmp*sin(2*pi*t*
            ↪ test_parameters.sineFreq))';
        y_sine = lsim(sys, sine_input, t);
        sqr_input = (test_parameters.sqrAmp*square(t,
            ↪ test_parameters.sqrFreq))';
        y_sqr = lsim(sys, sqr_input, t);

        %return sum of absolute error
        error.step = sum(abs(step_input-y_step));
        error.sine = sum(abs(sine_input-y_sine));
        error.sqr = sum(abs(sqr_input-y_sqr));

        %return sum of control effort
        effort_step = lsim(controller, step_input-y_step, t);
        effort_sine = lsim(controller, sine_input-y_sine, t);
        effort_sqr = lsim(controller, sqr_input-y_sqr, t);
```

```
        controlEffort.step = sum(abs(effort_step));
        controlEffort.sine = sum(abs(effort_sine));
        controlEffort.sqr = sum(abs(effort_sqr));
    end

    if plot_response
        figure

        subplot(3,2,1)
        plot(t, y_step, t, step_input)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Step Response')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow

        subplot(3,2,3)
        plot(t, y_sine,t, sine_input)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Sine Response')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow

        subplot(3,2,5)
        plot(t, y_sqr, t, sqr_input)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Square Response')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow

        subplot(3,2,2)
        plot(t, effort_step)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Control Effort for Step Input')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow

        subplot(3,2,4)
        plot(t, effort_sine)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Control Effort for Sine Input')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
```

```
        drawnow

        subplot(3,2,6)
        plot(t, effort_sqr)
        xlabel('Time (s)')
        ylabel('Amplitude')
        title('Control Effort for Square Input')
        h = findobj(gcf,'type','line');
        set(h,'linewidth',2);
        drawnow
    end

end
```

G.7 The square.m Script

Helper function for generating a square wave.

Listing G.7: For generating square waves without Signal Processing Toolbox.

```
function sqr = square(t,f)
%SQUARE Generates a square wave using time vector t, and real-
    ↪ frequency f
    sqr = sign(sin(2*pi*t*f));
end
```