Distributed learning for multi-agent control of a dynamic system

Mungo Louis Pay

Ph.D.

The University of York
Department of Electronics

November 2011

Abstract

This thesis describes an investigation of self-organising, distributed control of dynamic, non-linear systems. The distribution is achieved through a multi-agent based approach. The self-organisation is addressed through reinforcement learning.

The feasibility is tested using a well-established agent framework: JADE. The target system for the study is a simulation of a well-known laboratory demonstrator, the twin-rotor MIMO system, but configured to introduce strong cross-couplings in its non-linear dynamics.

A multi-agent PID controller is developed as an interim solution to test the feasibility of the use of JADE for control purposes.

An overarching constraint on the development of any solutions was that the plant knowledge was minimal, which placed great importance on the need for a self-organising scheme.

Results of a developed system are presented against the context of more conventional control methodologies.

Contents

List of Tables

List of Figures

Acknowledgments

The completion of this thesis would not have been possible without a number of people keeping me going throughout the whole process. Some helped directly with insight and guidance into the research itself, many helped indirectly with understanding and distraction. I would firstly like to thank my supervisor Tim Clarke who has always been enthusiastic, supportive and creative with suggestions on how to approach certain difficulties. It was always extremely useful to discuss and often debate the merits of different approaches with him, and without his guidance I never would have got so far. Thanks must also be given to Jonathan Aitken and Oliver Bunting for helping with a variety of queries related to LaTeXwhen writing this thesis.

For keeping me sane, I would like to thank my girlfriend Sarah for being so supportive and understanding during periods of frustration and irritation towards the work. She has been especially supportive over the last few months as the research has come to a close and stress levels have mounted. I would also like to thank Patrick Clarke who has been a friend and ally throughout our 7 years at the University of York.

For giving me a hobby and a weekly night out, I would like to thank the members of Cueball Z: James Lee, Jon Croxford, Michael Walsh, George Hogg and Matthew Robson, arguably the best pool team in York.

Finally, I would like to thank my parents and brother for always being supportive and knowing when not to ask me how the research was going.

M. L. Pay, November 2011

Declaration

This work is wholly my own and has been carried out with funding from EPSRC, the Engineering and Physical Sciences Research Council. To the best of my knowledge, this particular work has not been done anywhere else.

M. L. Pay, November 2011

Chapter 1

Introduction

Contents

This work is concerned with the development of a self-organising controller for dynamic systems. A multi-agent approach was taken, drawing upon the previous experience gained within the control laboratory at the University of York (Mendham & Clarke (2003*a*), Mendham & Clarke (2003*b*), Mendham et al. (2004), Mendham & Clarke (2005*a*) and Mendham & Clarke (2005*b*)), and harnessing some powerful and appropriate properties. The aim was to effect successful control action following a period of learning by a group of distributed agents. The agents were afforded the least amount of target plant information possible. The critical question asked, as the scheme emerged was, 'is this sufficiently generic to be applicable to any arbitrary plant without the need for any modification?'

The focus of the experimentation revolves around a Twin Rotor MIMO System (TRMS) as depicted in Figure 1.1. The TRMS is a fan-controlled dynamic system with two rotational degrees of freedom. It is moved to desired positions by balancing the speeds of the two rotors so that the rotational forces generated elicit a desired response. A more detailed look at the TRMS and the motivations for the selection of this system can be found in Chapter 3.

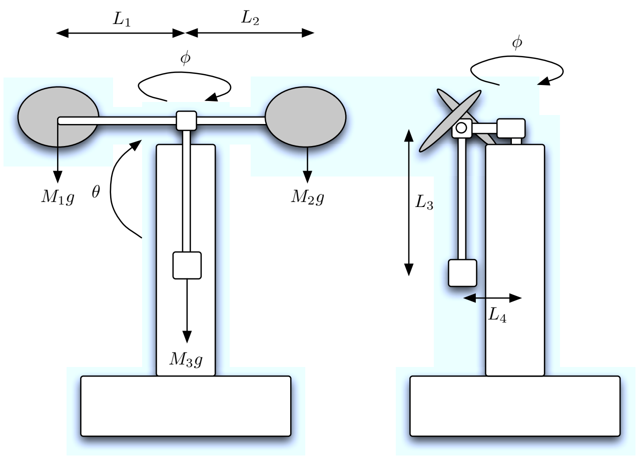

Figure 1.1: Diagram of the Twin Rotor MIMO system.

The use of multi-agent systems to act as a framework for control engineering has been an area with a great deal of promise but limited success. Many of the emergent properties that appear in real-life multi-agent systems, such as a freemarket economy (Smith, 1776) are attractive for control engineering. One of the more enticing properties is the potential to drive products towards an equilibrium price which has parallels with control engineering, as one primary concern there is to drive plant demand errors to zero. However, emergent behaviour, whilst being relatively simple to produce, is notoriously difficult to design and engineer for specific and complex goals. Stepney et al. (2006) note, when discussing *complicated* large engineering systems that:

> . . . they are difficult to understand, to analyse, and to design, because they cannot easily be separated into simpler parts. They are irreducible, not expressible in terms of the properties of their parts alone.

Some of the reasons that engineering emergent behaviour can be problematic will be discussed further in Section 1.2. Firstly, Section 1.1 demonstrates how seemingly complex behaviours can be produced from simple interaction rules.

1.1 Complex Dynamics from Simple Rules

It might seem logical for someone delving for the first time into the subject of complexity that, for something to display complex behaviour, there must be either complex inputs to a system or complex rules governing it. This is not the case, as is demonstrated in some examples now presented. In his book, *A New Kind of Science*, Wolfram (2002) demonstrates binary cellular automata organising into complex structures when their transition rules are very simple. Figure 1.2 shows a triangular fractal image that has been generated when the rule is that a cell should be black if one, and only one, of its neighbours to the left or right was black on the level above it, starting from a single black cell.

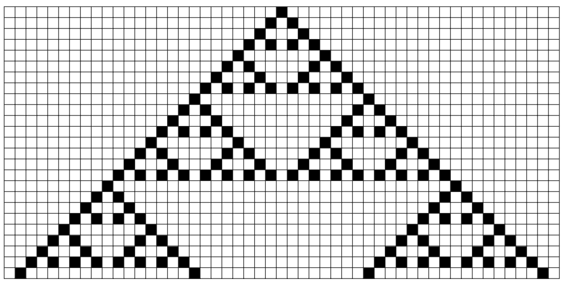

Figure 1.2: Cellular automata self organising into a fractal image.

Another commonly-cited example is that of swarm behaviour. As described in his seminal work, Reynolds (1987) investigates the potential for modelling the flocking behaviour of birds as particles, or *boids*, using three simple rules:

1. Collision Avoidance: avoid collisions with nearby flockmates
2. Velocity Matching: attempt to match velocity with nearby flockmates
3. Flock Centering: attempt to stay close to nearby flockmates

The result is a collection of individual entities which appear to move as a unit. He describes this:

> The animations showing simulated flocks built from this model seem to correspond to the observer's intuitive notion of what constitutes 'flock-like motion'.

Such rules could hardly be described as complicated and yet they produce complex structures and interactions that might otherwise have seemed difficult at best and impossible at worst to represent by any other means.

1.2 Engineering Emergence

This section discusses the inherent difficulties in attempting to design and engineer emergent systems. However, it is key that the term emergence is formally defined for the purposes of this thesis. When referring to 'emergent' in the context of science, the Oxford English Dictionary defines it as:

Definition 1. An effect produced by a combination of several causes, but not capable of being regarded as the sum of their individual effects.

This captures the essence of an emergent property being the by-product of both the constituent parts and also the interactions between said parts. In effect, the system is greater than the sum of its parts.

In his paper on epiphenomena, Abbott (2006) describes emergence as follows:

Definition 2. A phenomenon is emergent if it may be characterized independently of its implementation.

This is the definition that will be used throughout this thesis as it draws a clear divide between the emergent phenomenon and the implementation of the system as a whole.

As was noted in Section 1.1, it is possible and, in many cases, simple to produce seemingly complex emergent properties from simple rules. When it comes to *designing* emergent properties for complex systems, the task becomes altogether more difficult. An interesting analogue that describes this notion of reductionism not applying to emergent systems is that of the 'special sciences'. Whilst it can be thought that physics underpins all other sciences and therefore could, if it were understood fully, explain everything, why, in that case, are there the special sciences? Fodor (1997) writes on this subject:

> The very existence of the special sciences testifies to reliable macrolevel regularities that are realized by mechanisms whose physical substance is quite typically heterogeneous. Does anybody really doubt that mountains are made of all sorts of stuff? Does anybody really think that, since they are, generalizations about mountains-as-such won't continue to serve geology in good stead? Damn near everything we know about the world suggests that unimaginably complicated to-ings and fro-ings of bits and pieces at the extreme microlevel manage somehow to converge on stable macrolevel properties.

This is not to say that there are some things that are described by the special sciences that cannot be described by physics. It merely solidifies the notion that the added complexity that is applied when analysis is done from this viewpoint is impractical for most situations and potentially impossible for others.

Other issues with emergent engineering arise when the system is not fully understood. Wooldridge (2009) notes that:

> ...it is not always the case that *all* the characteristics of a system are known at design time.

When this is the case, the emergent properties would need to be adaptable to both known *and* unknown system properties. This is a difficult task for classical engineering, let alone emergent engineering.

1.3 Thesis Hypothesis

This thesis aims to defend and demonstrate that:

> A multi-agent approach is *viable* for the development of a distributed, self-organising, self-learning control scheme for a dynamic, non-linear plant.

The purpose of this work is not to produce a control strategy that rivals current control techniques in terms of performance, rather to lay the groundwork for future development of a plant independent, self-organising framework to control dynamic, non-linear systems.

1.4 Structure of this Thesis

Chapter 2 introduces some notions particular to the field of Control Engineering and of close relevance to this work. It discusses a variety of control techniques from Classical Control, through state variable control, to more unconventional methodologies. It also highlights some analysis techniques key to the design of modern control systems.

Chapter 3 introduces the TRMS system which will act as the plant for all simulation and experimentation. It provides details of the motivations for its use, analysis of the system properties and the modelling process, and gives some insights into control methodologies that have been used for control. The analysis of the TRMS dynamics show it to be a non-trivial system to control – especially if a *consistent* behaviour is required. A simple controller is applied as a base-line for future developments. The link between the analysis of the open loop system and subsequent closed loop behaviours is established.

Any self-organising system will respond to its environment. Here the system learns about the target plant. Chapter 4 investigates the motivations for and the various approaches to reinforcement learning. This includes a discussion of how the performance of different machine learning techniques differ when attempting to learn different behaviours. It considers the problem of identifying an appropriate state-action policy to achieve a learning goal based upon a reward and punishment strategy. Starting from a very simple scheme, the one actually applied in later chapters, it progresses to discussing some of the very popular and more elaborate approaches which have been applied on simple control problems using centralised implementations.

Chapter 5 discusses the attractive qualities of distributed artificial intelligence whilst detailing some of the potential pitfalls in their implementation. It also provides information on the development and propagation of multi-agent frameworks for various systems. A major part of the work of the author was the engineering of a substantial software system, used in Chapters 6/7, based on agent-oriented programming (AOP). An analysis of how this agent-oriented programming differs from object-oriented programming (OOP) is provided, identifying under what circumstances these differences are beneficial or detrimental. The JADE agent development framework, employed here, is introduced, along with details of its core functionality and key features. Finally, examples of multi-agent systems of note are discussed along with an introduction to the emergent field of multi-agent reinforcement learning.

Chapter 6 provides details of a distributed PID controller that was developed to control the TRMS. It details the motivations for developing such a system along with the system's limitations. Results of the control scheme are presented and analysed.

Chapter 7 describes the multi-agent reinforcement learning framework employed as part of the self-learning, self-organising control scheme for the TRMS. Implementation strategies are presented with justification for why some are used and others discarded. It also presents the results that the framework produce and discusses the extent to which they were successful.

Chapter 8 provides conclusions about the work along with descriptions of how the work could be furthered.

Each chapter includes its own bibliography.

1.5 Chapter Bibliography

Abbott, R. (2006), 'Emergence explained: Abstractions: Getting epiphenomena to do real work', *Complexity* **12**(1), 13–26.

Fodor, J. (1997), 'Special Sciences: Still Autonomous after All these Years', *Noûs* **31**(s11), 149–163.

Mendham, P. D. & Clarke, T. (2003*a*), Dependable intelligent control through the use of multiple intelligent agents, *in* 'Proceedings of the 16th International Conference on Systems Engineering, ICSE2003', IEEE, pp. 478–483.

Mendham, P. D. & Clarke, T. (2003*b*), Growing dependability using a multi-agent approach to fault tolerance, *in* 'Proceedings of the 54th Congress of the International Astronautical Federation', IEEE.

Mendham, P. D. & Clarke, T. (2005*a*), MACSim: A simulink-enabled environment for multiagent simulation, *in* 'Proceedings of the 16th IFAC World Congress', IEEE.

Mendham, P. D. & Clarke, T. (2005*b*), On acheiving true autonomy: Using multi-agent control for assto spaceplane, *in* 'Proceedings of Data Systems in Aerospace', IEEE.

Mendham, P. D., Pomfret, A. & Clarke, T. (2004), Dependable dynamic control using distributed intelligent agents, *in* 'Proceedings of the 55th Congress of the International Astronautical Federation', IEEE.

Reynolds, C. W. (1987), 'Flocks, Herds and Schools: A distributed behavioral model', *SIGGRAPH Computer Graphics* **21**(4), 25–34.

Smith, A. (1776), *An Inquiry into the Nature and Causes of the Wealth of Nations*, Liberty Fund.

Stepney, S., Polack, F. & Turner, H. (2006), Engineering emergence, *in* '11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06)', pp. 89–97.

Wolfram, S. (2002), *A New Kind of Science*, Wolfram Media Inc.

Wooldridge, M. (2009), *An Introduction to MultiAgent Systems*, 2nd edn, John Wiley & Sons.

Chapter 2

Control Engineering

Contents

Control systems vary significantly in complexity and design, ranging from a thermostat that turns heating or cooling off when a set temperature is reached, to the multiplicity of systems required to control a modern, sophisticated, military helicopter. They are not limited to mechanical systems either: organic life has a mélange of feedback and feedforward control pathways, influencing everything from enzyme production for metabolisation to hormone regulation.

Control Engineering is the discipline of developing control systems for engineering applications so that desired dynamic characteristics may be achieved. Nise (2010) defines a control system in the following way:

> A control system consists of *subsystems* and *processes* (or *plants*) assembled for the purpose of obtaining a desired *output* with desired *performance*, given a specified *input*.

To meet these goals a control engineer can use a variety of tools and design methodologies to analyse and implement an appropriate control system.

The aim of the work described in this thesis is the development of autonomous, self-organising, self-learning, distributed control of a non-linear dynamic system. Many of the analysis techniques and terminology are grounded in the field of Control Engineering. This chapter aims to focus the attention of the reader on a selection of aspects of Control Engineering. These will later be applied in the analysis of a target dynamic system and some intermediate controller schemes. This chapter is designed as a simple introduction to Control Engineering, for readers with limited or no former experience of the field, so that analysis of the target system can be understood. Readers with a good understanding of the field may wish to move on to Chapter 3.

2.1 Defining the System

For the purposes of this thesis, a dynamic system will be defined as a collection of interacting components that, together, are capable of carrying out some objective and whose behaviour can be encapsulated using a set of differential or difference equations. The system will normally respond to an excitation (input) and produce a response (output). Its behaviour can be ascribed to the **system variables**. These can be split into four categories as depicted in Figure 2.1:

- **Input variables:** The inputs to the system, frequently human-operated, are how the system is stimulated.
- **Disturbance variables:** These are inputs to the system that are out of the control

of the operator. Normally they are due to the system's environment or its own internal components. They usually lead to a deviation away from the system's expected response.

- **Output variables:** The measured system responses to the combination of the input and disturbance variables.
- **State variables:** These are internal variables which characterise the system behaviour and define the output response to a combination of input and disturbance variables.

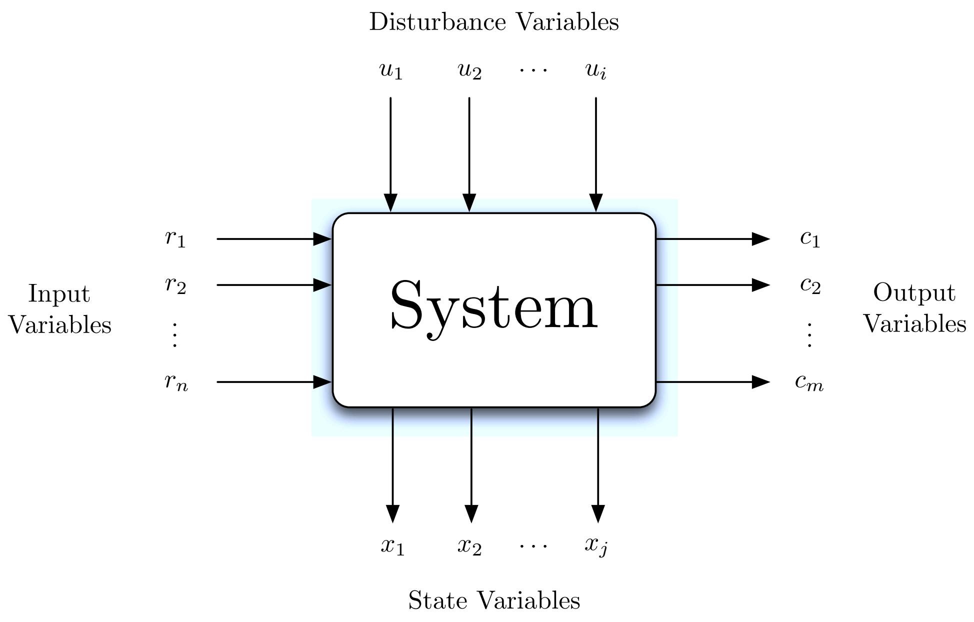
Disturbance Variables
u_1 u_2 $\cdots$ u_i
Input Variables
r_1 r_2 $\vdots$ r_n
System
c_1 c_2 $\vdots$ c_m
Output Variables
x_1 x_2 $\cdots$ x_j
State Variables

Figure 2.1: The system variables

2.2 Control System Classification

Producing a desired output response relies on the existence of inputs of an appropriate nature. However, appropriate inputs are not necessarily congruent with the desired input mechanism, controlled by the operator. For this reason a controller is added to adapt (or shape) the operator's control input to produce appropriate system inputs for a desired response.

2.2.1 An Open-Loop Control System

Figure 2.2 depicts the simplest form of a control system: a combination of controller and the system under control, without feedback.

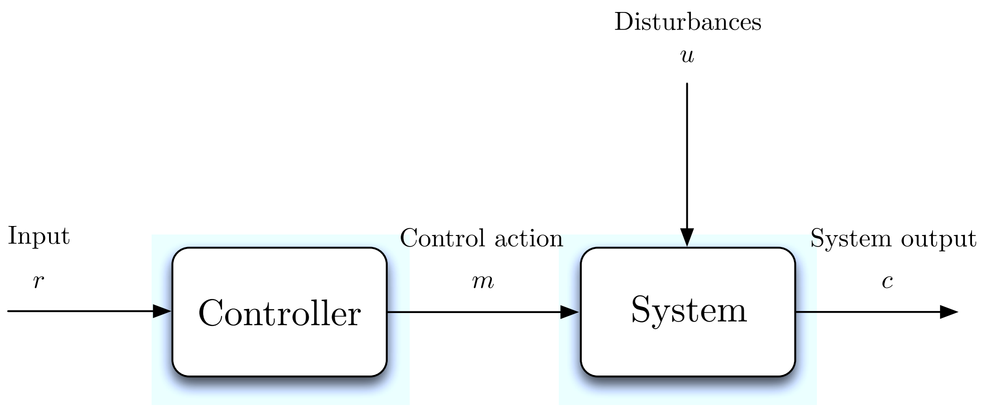

Figure 2.2: A standard open-loop control system

This is known as open-loop control. Such a scheme will reshape operator inputs but relies upon time-invariant properties of the system. It also ignores the effects of external influences. Consequently, for anything other than the simplest of systems, such a scheme is often impractical.

2.2.2 Closing the Loop

By applying negative feedback, the closed loop system performance is made less sensitive to plant changes. Any good text on systems theory will afford the reader with a full treatment of the effect of negative feedback on system sensitivity. Additionally, negative feedback has, generally, excellent disturbance rejection properties, assuming a well designed closed loop system. Most fundamentally, an appropriate controller will shape the behaviour of the system – discussed in some detail later. Figure 2.3 depicts a controller with negative feedback.

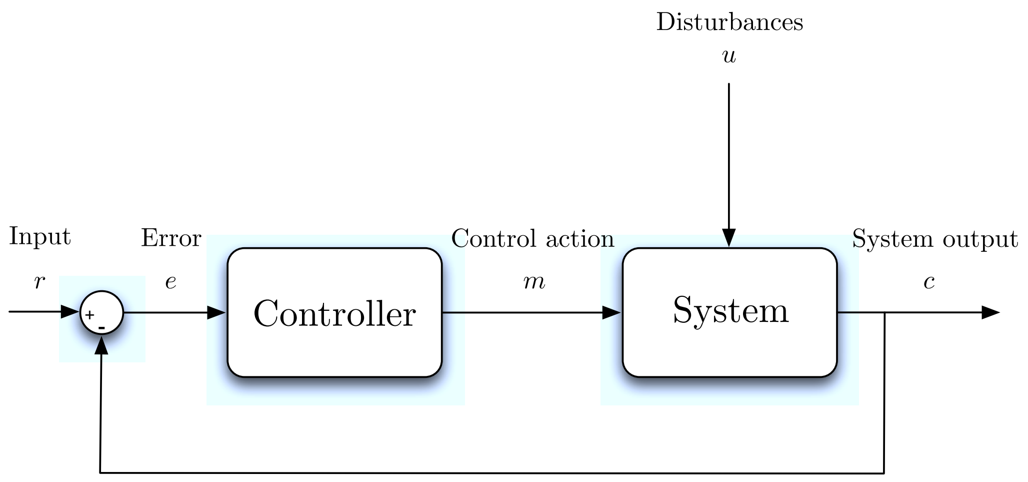

Figure 2.3: A standard closed-loop control system

2.2.3 SISO and MIMO Control Systems

It is common to classify control systems based on the number of inputs and outputs: Single Input Single Output (SISO) and Multi Input Multi Output (MIMO). See Figure 2.4.

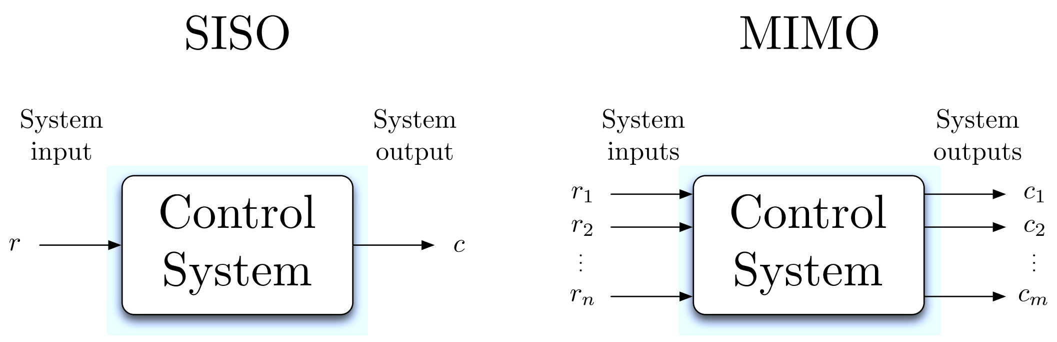

Figure 2.4: A general description of SISO and MIMO systems

2.2.4 Model Description Based System Classification

Control systems can also be classified by how they can be represented mathematically. In such a discussion of dynamic systems, three classes of controlled systems can be considered: The plant and controller having continuous dynamics (i.e. both can be described using differential equations); the plant having continuous dynamics but the controller being digital and relying upon discrete inputs to and sampled outputs from the plant; and the plant being discrete in nature (i.e. described by difference equations) and the controller being the same.

This chapter focusses on continuous plant and controllers. Salient issues can be mapped across to the second category. Discrete plants are not discussed further as they are outside the context of this work. As such, the discussion will be restricted to linear controllers operating on linear and non-linear systems. A system is said to be linear if it conforms to the principles of superposition and homogeneity. Superposition is satisfied if the response of a system to the sum of a collection of inputs is equivalent to the sum of the responses of the system to the individual inputs. So, for two input components

$$f\left(u_1\right) + f\left(u_2\right) = f\left(u_1 + u_2\right) \tag{2.1}$$

Homogeneity describes the system response when the input is multiplied by a scalar, i.e. for some function

$$y = f\left(u\right) \tag{2.2}$$

for homogeneity to be satisfied, the following must be true

$$\alpha y = f\left(\alpha u\right) \tag{2.3}$$

Systems which break either or both properties are classed as non-linear. Most real systems do this, but may be classed as approximately linear, if the range of operation is restricted.

For some systems, the effects of non-linearity are simply too great to ignore and must be taken into account. In Chapter 3, the degree of non-linearity of the TRMS, which is the main focus of this work, will be considered.

2.3 The Laplace Transform

The mathematical description of a linear system can be represented by differential equations. The complexity of the system generally dictates the complexity of the mathematical representation. However, a generic linear differential equation representation using constant coefficients can be written in the form:

$$A_n \frac{d^n c\left(t\right)}{dt^n} + \ldots + A_1 \frac{dc\left(t\right)}{dt} + A_0 c\left(t\right) = B_m \frac{d^m r\left(t\right)}{dt^m} + \ldots + B_1 \frac{dr\left(t\right)}{dt} + B_0 r\left(t\right) \tag{2.4}$$

where $m < n$ for causality and where $r(t)$ and $c(t)$ are inputs and outputs respectively.

Whilst there are analytical methods for directly solving such equations, they tend to be tedious and error prone for all but very simple systems.

The Laplace transform provides an alternative representation. It relies on transforming the equations from the time domain into a different domain, which makes analysis more straightforward. It is given by

$$\mathcal{L}[f(t)] = F(s) = \int_{-\infty}^{\infty} f(t)\, e^{-st} dt \tag{2.5}$$

where

$$s = \sigma + j\omega \tag{2.6}$$

This particular form is the bilateral Laplace transform, integrating over all time, both positive and negative. However, since dynamic systems are generally considered quiescent prior to stimulus, it is conventional to change the limits of integration to give the single-sided Laplace transform, which will be employed hereafter:

$$\mathcal{L}[f(t)] = F(s) = \int_{0}^{\infty} f(t)\, e^{-st} dt \tag{2.7}$$

The change in notation to using an uppercase F to represent a function in the Laplace domain is purely by convention rather than denoting any additional system property.

Transformation to the Laplace domain facilitates handling of differential equations as algebraic expressions. The fundamental two properties that enable this are:

$$\mathcal{L}\left[\frac{df(t)}{dt}\right] \equiv sF(s) - f(0) \tag{2.8}$$

and

$$\mathcal{L}\left[\int_0^t f(t)\, dt\right] \equiv \frac{1}{s} F(s) \tag{2.9}$$

The equation for general differentiation is given by

$$\mathcal{L}\left[\frac{d^n f(t)}{dt^n}\right] \equiv s^n F(s) - s^{n-1} f(0) - s^{n-2} \frac{df(0)}{dt} - \dots \frac{d^{n-1} f(0)}{dt} \tag{2.10}$$

Using this, the Laplace domain representation of the differential equation given at Equation 2.4 can be represented as

$$\begin{aligned}
&\left(A_n s^n + A_{n-1} s^{n-1} + \ldots + A_0\right) C(s) - \\
&\left(A_{n-1} s^{n-1} + A_{n-2} s^{n-2} + \ldots + A_1\right) c(0) - \\
&\left(A_{n-2} s^{n-2} + A_{n-3} s^{n-3} + \ldots + A_2\right) \frac{dc(0)}{dt} - \ldots - A_n \frac{d^{n-1} c(0)}{dt^{n-1}} = \\
&\left(B_m s^m + B_{m-1} s^{m-1} + \ldots + B_0\right) R(s) - \\
&\left(B_{m-1} s^{m-1} + B_{m-2} s^{m-2} + \ldots + B_1\right) r(0) - \\
&\left(B_{m-2} s^{m-2} + B_{m-3} s^{m-3} + \ldots + B_2\right) \frac{dr(0)}{dt} - \ldots - B_m \frac{d^{n-1} r(0)}{dt^{n-1}}
\end{aligned} \tag{2.11}$$

where $c(0)$, $\frac{dc(0)}{dt}$, …, $\frac{d^{n-1}c(0)}{dt^{n-1}}$ and $r(0)$, $\frac{dr(0)}{dt}$, …, $\frac{d^{m-1}r(0)}{dt^{m-1}}$ are the functions $c(t)$, $r(t)$ and their derivatives at $t = 0$, otherwise called their initial conditions.

By convention, using this representation, zero initial conditions are assumed and Equation 2.11 can be rewritten as

$$\left(A_n s^n + A_{n-1} s^{n-1} + \ldots + A_0\right) C(s) = \left(B_m s^m + B_{m-1} s^{m-1} + \ldots + B_0\right) R(s) \tag{2.12}$$

Again, by convention, this is normally rearranged to give the transfer function $G(s)$ of the form

$$G(s) = \frac{C(s)}{R(s)} = \frac{\left(B_m s^m + B_{m-1} s^{m-1} + \ldots + B_0\right)}{\left(A_n s^n + A_{n-1} s^{n-1} + \ldots + A_0\right)} \tag{2.13}$$

Similarly, a Laplace domain representation of a controller can be defined as $H(s)$. The structure of a series controller is shown in Figure 2.5.

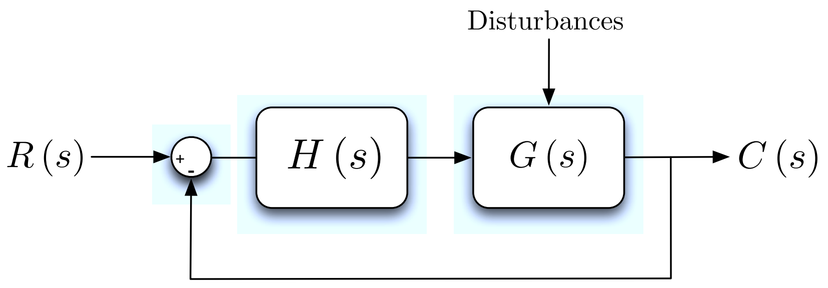

Figure 2.5: Negative feedback with an added controller in the Laplace domain

This closed loop transfer function is written as

$$\frac{C(s)}{R(s)} = \frac{H(s)G(s)}{1 + H(s)G(s)} \tag{2.14}$$

Clearly, the properties of the controlled system can be altered by changing the transfer function $H(s)$ to suit the requirements.

$G(s)$ represents the Laplace transform of the impulse of a linear system. To give the output for any arbitrary input, in the time domain, the convolution operator must be used, where

$$c(t) = \int_0^\infty r(\tau)\, g(t-\tau)\, d\tau \tag{2.15}$$

denoted $r(t) * g(t)$, is this operation.

A useful property of a Laplace domain representation is that convolution in the time domain is equivalent to multiplication in the Laplace domain and vice versa, so

$$\mathcal{L}[r(t) * g(t)] \equiv R(s)G(s) = C(s) \tag{2.16}$$

and

$$\mathcal{L}[r(t)\, g(t)] \equiv R(s) * G(s) \tag{2.17}$$

This greatly simplifies the process of determining the output for a given input.

Table 2.1 provides a list of standard inputs and their Laplace domain representation.

$f(t)$	$F(s)$
Unit impulse, $\delta(t)$	1
Unit step, $u(t)$	$\frac{1}{s}$
$t \cdot u(t)$	$\frac{1}{s^2}$
$t^n \cdot u(t)$	$\frac{n!}{s^n+1}$
$e^{-at} \cdot u(t)$	$\frac{1}{s+a}$
$\sin(\omega t) \cdot u(t)$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t) \cdot u(t)$	$\frac{s}{s^2+\omega^2}$

Table 2.1: Laplace transform table

2.4 System Poles and Zeros

The system poles and zeros provide a particularly valuable insight into the analysis of system behaviour. Formally, the poles are defined as those values of the Laplace variable, s, such that

$$G(s) = \infty \tag{2.18}$$

Similarly, the zeros of a system are the values of s such that

$$G(s) = 0 \tag{2.19}$$

The conventional representation of these is via an Argand diagram (as depicted in Figure 2.6), representing the complex variable s-plane in which the poles and zeros lie.

2.5 Important System Properties

When designing a controller for a system there are a three fundamental system properties a control engineer will have to consider:

- Stability.
- The transient behaviour.
- Steady state error performance.

The pole-zero map provides insights into all three system properties. The reader is referred to any good control engineering text for detail.

2.5.1 Stability

Dorf & Bishop (2008) define a stable system as:

> A dynamic system with a bounded response to a bounded input.

This is often referred to as absolute stability. For such a system, all poles must lie in the left half of the complex s-plane. There are, however, varying degrees of stability, which can be defined as relative stability. If a stable system is perturbed by small, brief transient input, then the output will return to its original value. How it does this (i.e. the transient behaviour) is described as the system relative stability.

2.5.2 Transient Behaviour

Consideration of the transient response of a system is important when designing a control system. Nise (2010) describes an example of a lift where the tradeoff is between patience and comfort. If the lift moves too quickly then the passengers will experience large and uncomfortable forces. On the other hand, if the lift moves too slowly then it will take a long time to reach the floor and the passengers will be frustrated. By applying specific input test signals such as steps, ramps and impulses, the resultant output can give information to the control engineer about transient behaviour generally based upon:

- Percentage overshoot and settling time.
- Oscillatory behaviour.
- Speed of response.

Consider the following standard second order system, as given in Equation 2.20, where ζ is the damping ratio and ω_n is the natural frequency of the system.

$$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.20}$$

This equates to a Pole-Zero map as shown in Figure 2.6.

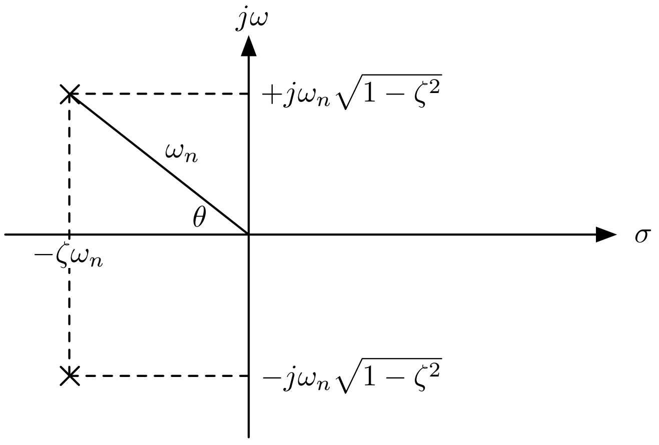

Figure 2.6: Pole plot for a second order underdamped system. Adapted from Nise (2010)

The damping angle, θ, is given by

$$\zeta = cos\theta \tag{2.21}$$

Overshoots and Settling Time

The settling time, or T_s, is conventionally defined as the time required for the system to settle to within 98% of the final settled output value, so

$$T_s \approx \frac{4}{\zeta\omega_n} \tag{2.22}$$

The percentage overshoot, or %OS, is defined as the maximum peak value of the response curve measured with respect to the final steady state value. It can be calculated by using Equation 2.23.

$$\%OS = e^{-\left(\frac{\zeta\pi}{\sqrt{1-\zeta^2}}\right)} \times 100 \tag{2.23}$$

Settling is a function of both the natural frequency of the poles ω_n and the damping ratio ζ, whereas the $\%OS$ is simply a function of ζ.

Figure 2.7 shows the relationship between pole positions and transient behaviour, where a) gives constant settling times, b) gives constant natural frequency and c) gives constant damping and, hence overshoot.

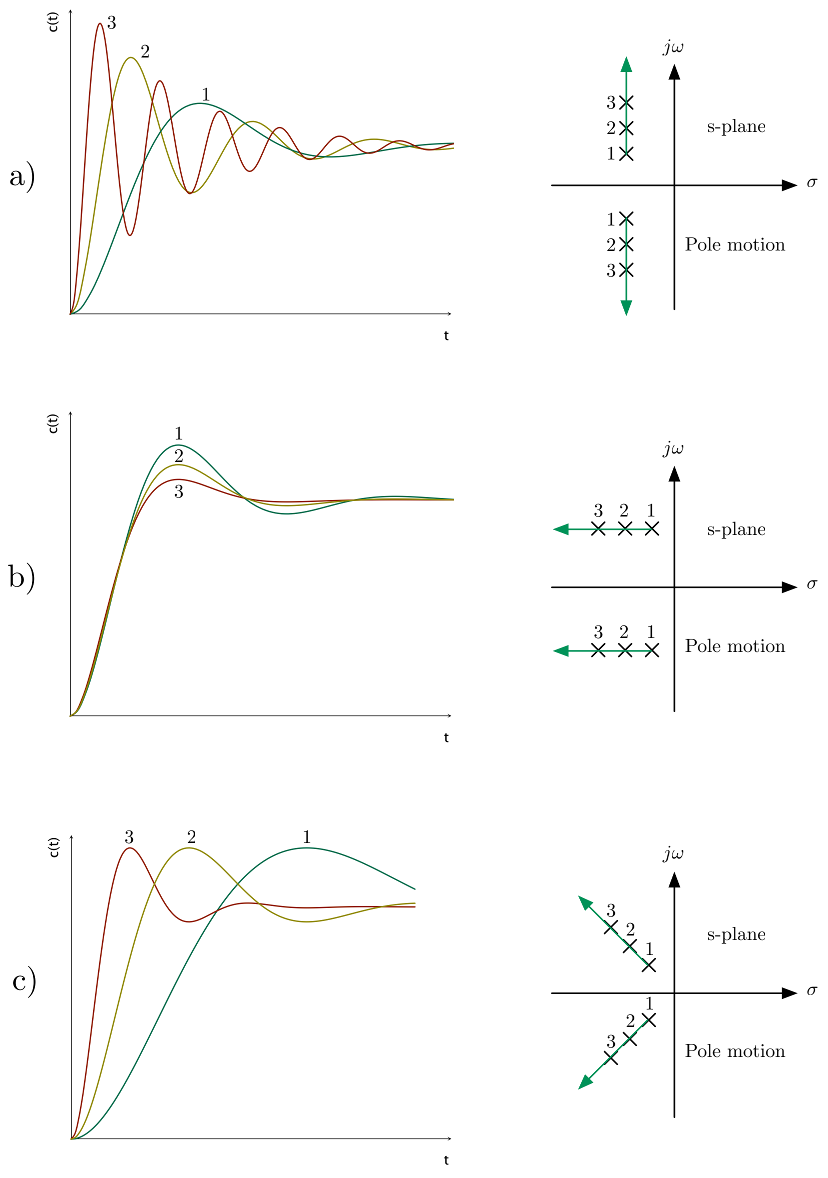

Figure 2.7: The result of migrating poles on a step response. Adapted from Nise (2010)

Oscillatory Behaviour

Clearly there is a strong relationship between complex pole pair positions and oscillatory behaviour in terms of frequency and damping.

2.5.3 Steady State Error

Steady state error is defined as the residual difference between the desired and the achieved output magnitudes, once all transients have died away. Using the previous lift analogy, if there is a step up or down between the lift floor and the building floor after the doors have opened, this would be the steady state error of the lift controller.

In analytic terms, the open loop transfer function defines the closed loop steady state error performance in the absence of a dynamic controller. It is the system type that does this, which is defined as the number of pure integrators in the open loop plant. The errors will depend upon the nature of the input signal: step, ramp, or higher order. They can be summarised in tabular form as in Table 2.2, where the unity negative feedback control configuration in Figure 2.8 has been employed. A constant gain value, K, is applied.

	Step input $r(t)=1$	Ramp input $r(t)=t$	Parabola input $r(t)=\frac{1}{2}t^2$
Type 0 system	$\frac{1}{1+K}$	∞	∞
Type 1 system	0	$\frac{1}{K}$	∞
Type 2 system	0	0	$\frac{1}{K}$

Table 2.2: Steady state error in terms of the gain K. Adapted from Ogata (2001)

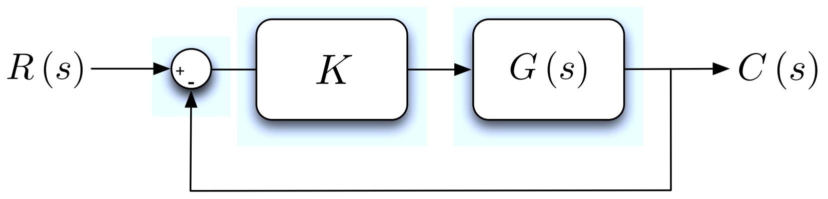

Figure 2.8: The unity negative feedback control configuration

By applying an appropriate control policy, these errors can be eliminated, as well as implementing improvements in transient behaviour. A very popular control policy uses the PID approach.

2.6 PID Control

Proportional-Integral-Differential (PID) controllers are perhaps the most widely used control structure in Control Engineering. The reason for this is that they are relatively simple to implement, understand and tune. The structure of a PID controller is shown in Figure 2.9.

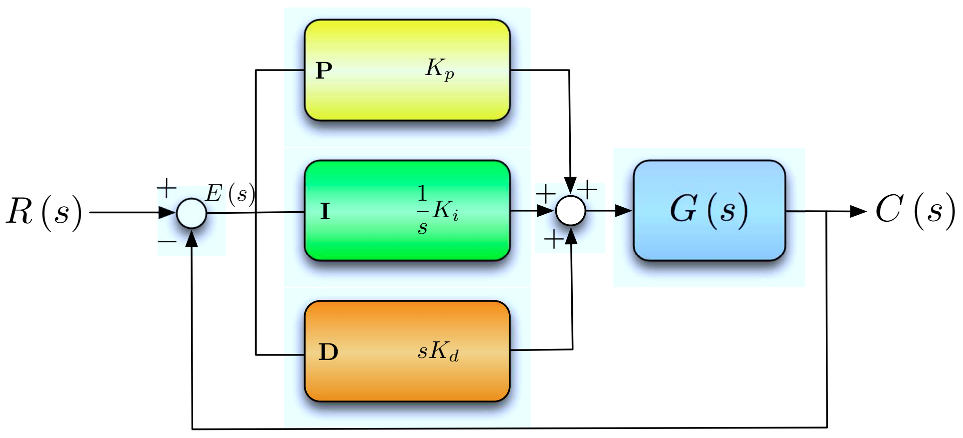

Figure 2.9: The structure of a PID controller

An example of a system that might benefit from such a control scheme would be a motorized tracking system for a telescope. The specifications would require the control system not only to locate celestial objects based on error terms, but also track them across the night sky as the Earth rotates. The input signal that the system must track will be a composite of angular position, velocity, albeit minute, acceleration components. Having the proportional and integral terms provides functionality to reduce the steady state error for both position and velocity to zero. It does this because the integral term effectively increases the system type. The differential term comes into force to damp the effects of the proportional and integral terms, providing good transient behaviour.

2.7 Fuzzy Control Systems

Fuzzy control is the application of fuzzy logic to the field of Control Engineering. The concept of fuzzy logic derives from the idea that there are instances where the absolute nature of Boolean logic is too restrictive, so the notion of variable truth is applied. It was first proposed by Zadeh (1965) in his paper on fuzzy sets. One example he uses for a set that cannot be split into Boolean states is that of 'tall men'. If one was to define whether a man was tall or not, there would have to be a rigid boundary where men above a certain height were considered tall and those below it would not. Fuzzy logic allows for this variable truth, by giving every member in the set a value of truth between 0 and 1. Figure 2.10 depicts an example of how a membership function for 'tall men' might be implemented.

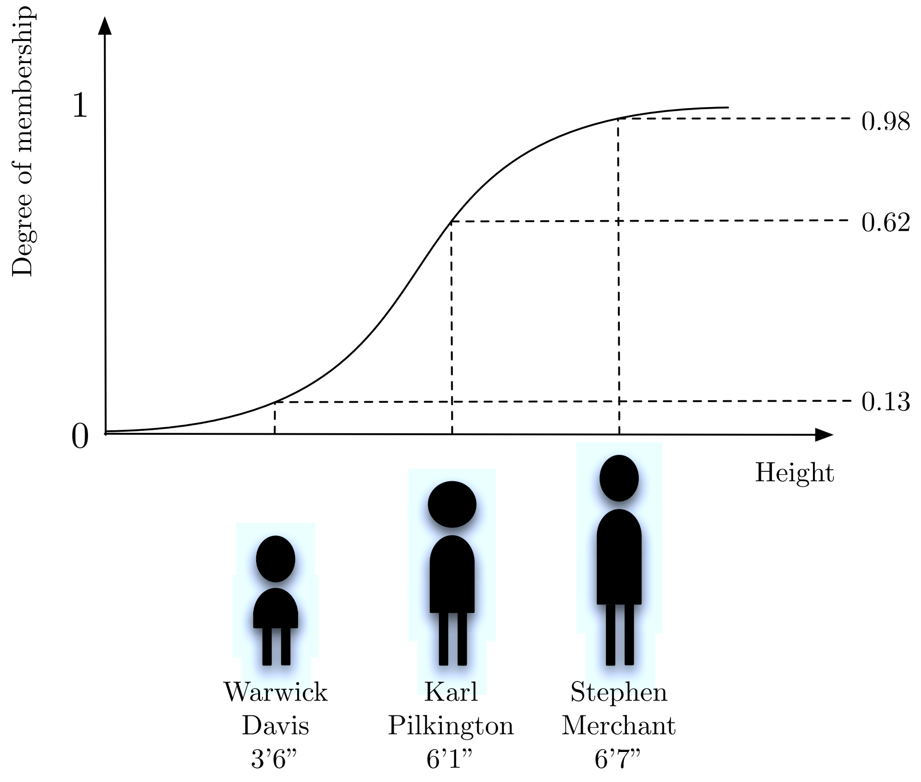

Figure 2.10: Three men with varying membership of a ‘tall men’ membership function

Fuzzy Control diverges from classical control in the sense that system properties are inferred so that they may be split into *Fuzzy Sets* (*fuzzifying*). Analysis of the sets must then occur before combining action decisions (or *defuzzifying*) into a single control policy. It is attractive for a number of applications, as an in-depth analysis of the system and its mathematical properties are not required, simply an understanding of the characteristic behaviour the system exhibits. Fuzzy control systems are commonly used for controlling consumer electronics products such as vacuum cleaners, washing machines and somewhat ironically, the lens focusing on some digital cameras.

For the purposes of this section, an inverted pendulum will be used as an example to demonstrate the fuzzy control design process. An inverted pendulum system is shown in Figure 2.11.

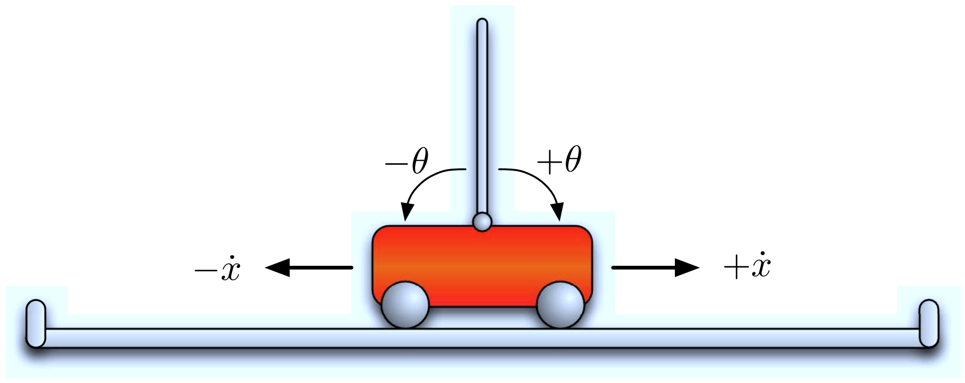

Figure 2.11: Diagram of an inverted pendulum system

The system comprises a cart, located on a track, that can move quickly and freely in one dimension, thus altering the moment on the freely hinged pendulum. Onboard sensors allow for horizontal position/velocity monitoring of the cart and angular position/velocity monitoring of the pendulum. There is a single input which can alter the cart velocity vector, but there is no direct control of the pendulum itself. The aim is to balance the inverted pendulum at a pre-determined location on the track.

2.7.1 Fuzzification

The act of fuzzification is arguably analogous to the analysis and mathematical modelling stage of classical control engineering. It is the process of producing membership functions for each fuzzy set associated with the control process.

Figure 2.12 shows how a combination of triangular and trapezoidal membership functions might be assigned to the θ angle set of the inverted pendulum system.

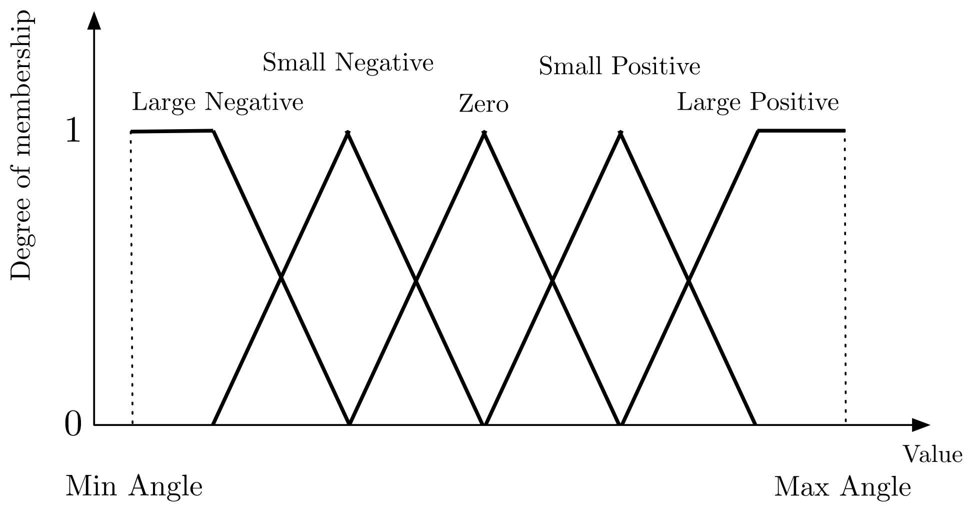

Figure 2.12: Fuzzy membership functions for the angle of an inverted pendulum

With the membership functions assigned to the θ angle set, Figure 2.13 shows how a single value for θ can belong to more than one membership function in that set.

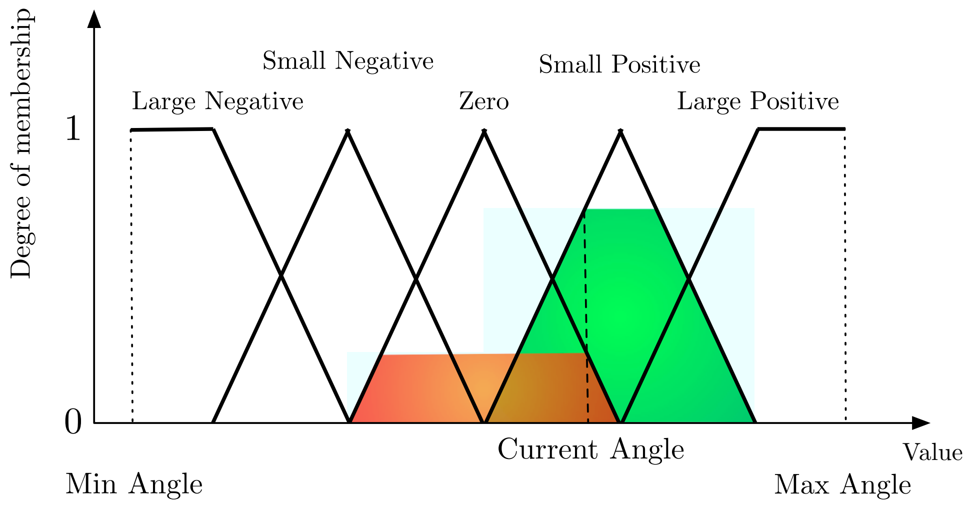

Figure 2.13: Two active membership functions produced by a single θ angle value

Since the membership functions overlap, all values of θ between the minimum and maximum allowed angles can be described in a continuous fashion by a combination of memberships. For the purposes of this thesis any membership function that covers the current value is described as *active*. This means that, based on Figure 2.12, for the majority of the time, two membership functions will be *active*. Most fuzzy systems use uniform, triangular membership functions because they are simple to construct and from them it is mathematically very simple to calculate degrees of membership. Different shapes of membership functions, such as Gaussian and sigmoidal, can be implemented which do affect the operation of the controller. The effect of altering the shape of membership functions is not discussed here, but further information can be found in Driankov et al. (1993).

Another factor that can be altered is the uniformity of the membership functions as depicted in Figure 2.14.

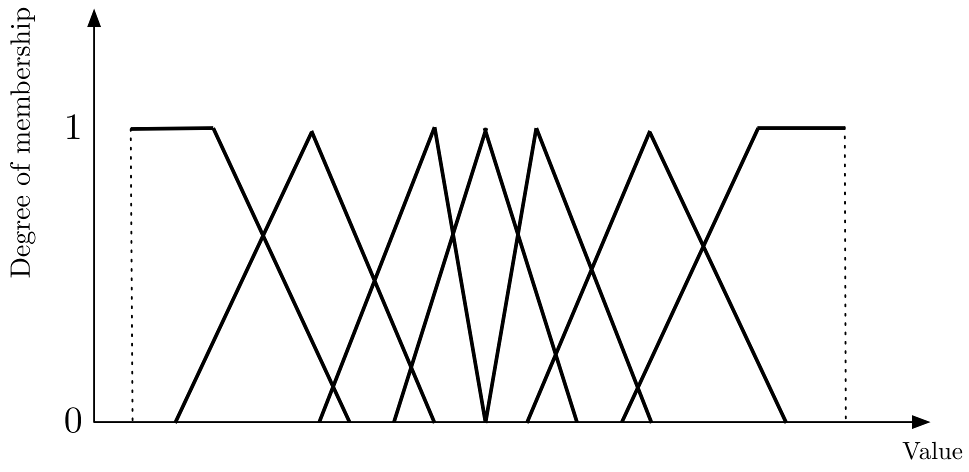

Figure 2.14: Fuzzy membership functions with increased sensitivity around the central point

This can provide more variety when assigning membership functions for defuzzification, but since controllers must be designed on a case-by-case basis, the effect of non-uniform membership functions is not discussed. However, more information can, once again, be found in Driankov et al. (1993).

Once all of the fuzzy sets have been assigned and populated with membership functions, the fuzzy controller output sets may be defined and then defuzzification can be considered.

2.7.2 Assigning Output Sets

To close the control loop, any one of the membership functions in the input sets can be assigned to a membership function in the output sets. In the inverted pendulum example, there is only one input, that of desired cart velocity, so there is only one output set, as depicted in Figure 2.15.

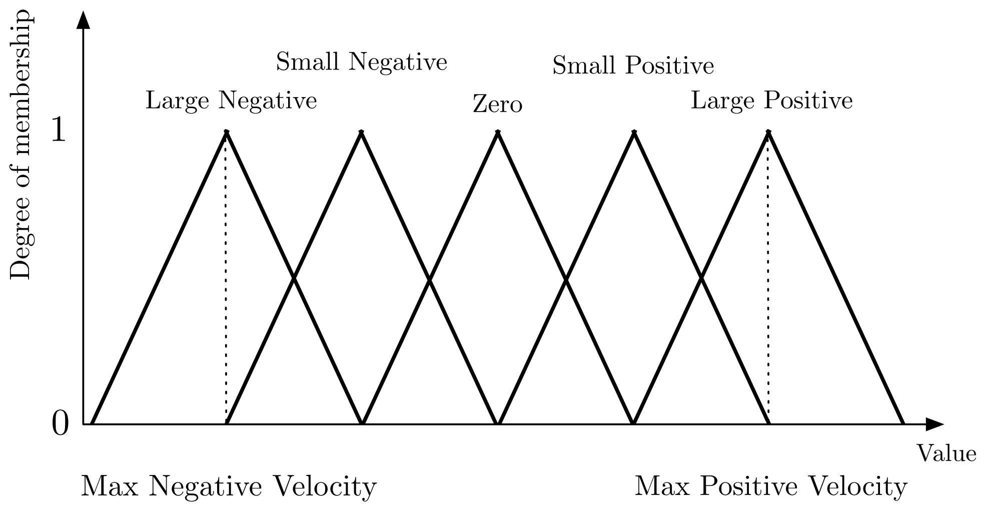

Figure 2.15: An example defuzzification member function set for an inverted pendulum

There is a lot of similarity between this output fuzzy rule set and the input set for the inverted pendulum θ angle. One minor change is that the membership functions at the extremes of operation are now triangular. The reasoning behind this is discussed later in Section 2.7.3. The only other difference is that it deals with the desired cart velocity, rather than pendulum angle θ.

Tables 2.3 and 2.4 show examples of how output membership functions might be assigned to the membership functions of the θ and $\dot{\theta}$ sets respectively. Here, S and L denote *small* and *large* respectively, whilst N and P stand for *negative* and *positive*, so SN refers to a *small negative* value.

Pendulum Angular Position					
Membership Function	LN	SN	Zero	SP	LP
Rule	LP	SP	Zero	SN	LN

Table 2.3: Fuzzy membership rule function for angular position of an inverted pendulum

Pendulum Angular Velocity					
Membership Function	LN	SN	Zero	SP	LP
Rule	LP	SP	Zero	SN	LN

Table 2.4: Fuzzy membership rule function for angular position of an inverted pendulum

If the desired state of the pendulum is assumed to be central, stationary, vertical and not rotating, error conditions can be defined on θ and $\dot{\theta}$. From this, controller actions can be ascribed, in terms of desired cart velocity, to these error terms.

These assignments are based on the input sets using an error term as their input as with standard feedback control. For example, if there is a positive error in θ, the pendulum arm tilts in the negative θ direction. A negative velocity must be imparted to the cart to keep it upright, as depicted in Figure 2.16.

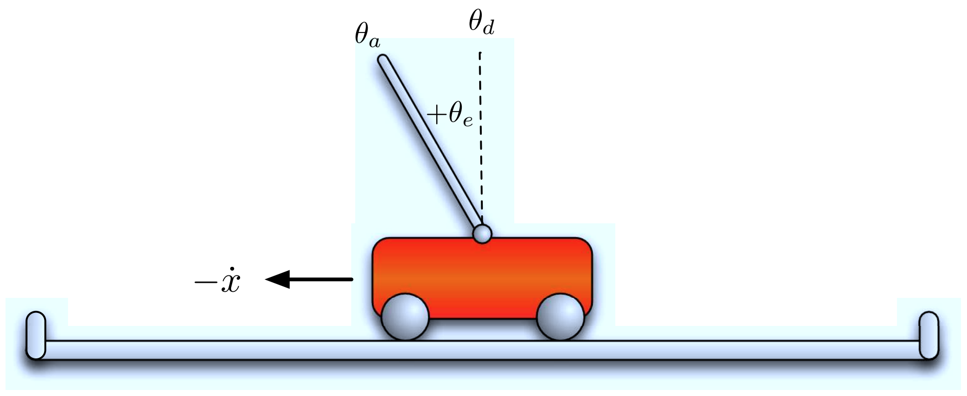

Figure 2.16: Inverted pendulum with a negative θ arm position hence a positive θ error

With the membership functions assigned, the defuzzification process deals with the calculation of the magnitude of the control action. Note that input membership sets can be combined using standard Boolean logic techniques and assigned to the output membership functions. The three standard Not, And and Or operators can be assigned in the following manner.

- Not $A(x)$: $(1 - A(x))$
- $A(x)$ And $B(x)$: $min(A(x), B(x))$
- $A(x)$ Or $B(x)$: $max(A(x), B(x))$

However, in this example, the combination of fuzzy sets is done in the defuzzification process.

2.7.3 Defuzzification

Defuzzification refers to the averaging of all active output membership functions and mapping onto control outputs. Several averaging techniques have been developed to defuzzify the membership functions. The most commonly used, and the one that will be focussed on here, is the centroid defuzzification method. The centres of mass for all active output membership functions are calculated to generate an input to the system based on the rules. It is for this reason that a triangular membership function is used for the extremities of the output set in Figure 2.15, as full membership to these functions should result in a maximum desired output velocity. A formula for calculating the centroid value is

$$x^* = \frac{\int x\mu_i\left(x\right)dx}{\int \mu_i\left(x\right)dx} \tag{2.24}$$

where x^* is the output of the defuzzification process and μ_i is the aggregated membership function. An illustration of this procedure for two values spanning four defuzzification rules is shown in Figure 2.17.

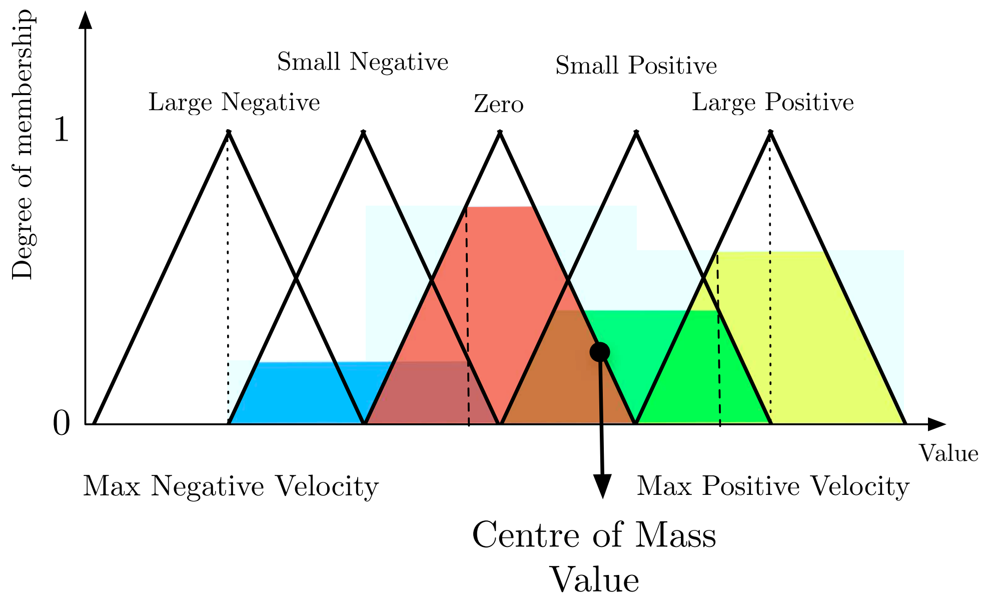

Figure 2.17: Calculating the centre of mass for two values spanning four rules

Here, the two system states are mapped onto the output membership functions based on their degree of membership. Since they each span two membership functions, there are four which are active and used to calculate the centre of mass. When this is applied to the inverted pendulum problem, the results of a small negative θ angle and a small positive $\dot{\theta}$ angular velocity can be seen in Figure 2.18.

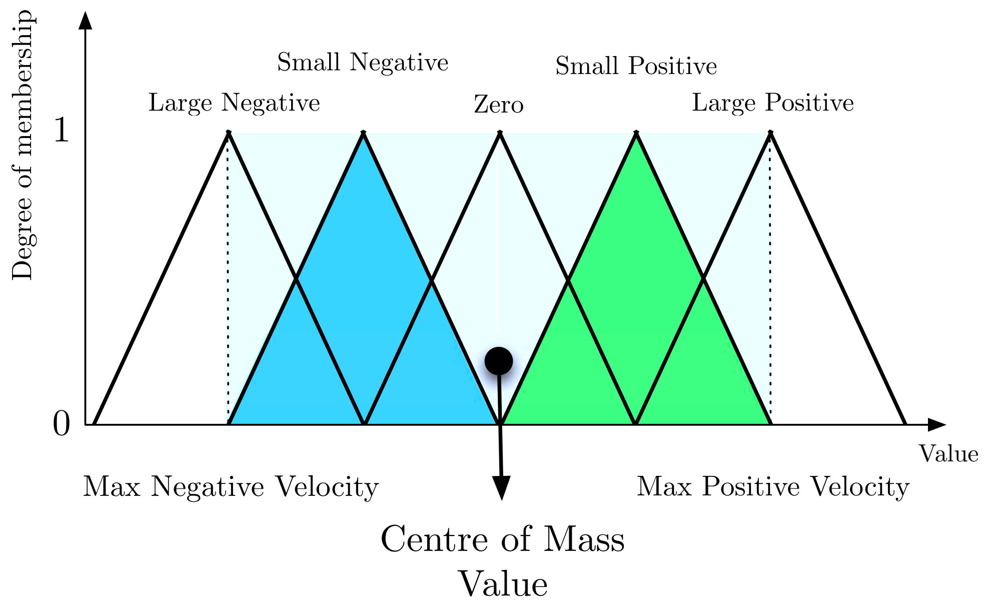

Figure 2.18: Defuzzification of the inverted pendulum ruleset for two values spanning two rules

Here, since the pendulum has a positive angular velocity and a negative angle, it is moving towards the equilibrium position, so the cart does not need move. Scaling factors are often used to get the system to work, but they are generally tuned at a later stage. It is the simplicity of design that makes fuzzy controllers attractive provided that the behaviour of the plant can be broken down into elementary and understandable sets.

2.7.4 Fuzzy Control Examples

There is little doubt that fuzzy logic is a powerful and versatile methodology. Since its introduction as a control methodology, fuzzy logic has been employed on a number of systems. Magana & Holzapfel (1998) produces a fuzzy controller for an inverted pendulum system, similar to the example in Figure 2.11, using a camera system to provide state information. Driankov et al. (1993) provides many examples of the use of fuzzy control for stabilised platforms which isolate vehicle motion. Fuzzy logic can even be adapted to image processing. Young & Krishnapuram (1997) provide examples of image enhancing.

It is easy to see why fuzzy control is so attractive for a wide range of problems, as it can be very powerful and yet simple to grasp. One criticism that is often levelled at fuzzy systems is that it is hard to validate the optimality of the solutions it provides. As such, it splits the research community as to whether it is an appropriate design methodology.

2.8 Conclusions

The motivation for this chapter has been to provide some focus for the later discussion of the dynamics of a target system used in the development of a self-organising controller. It also aimed to give a brief introduction to two control strategies that will be applied, as intermediates to this dynamic system, namely PID and fuzzy control.

2.9 Chapter Bibliography

Dorf, R. C. & Bishop, R. H. (2008), *Modern Control Systems*, 12th edn, Pearson Prentice Hall India.

Driankov, D., Hellendoorn, H. & Reinfrank, M. (1993), *An Introduction to Fuzzy Control*, Springer-Verlag.

Magana, M. E. & Holzapfel, F. (1998), 'Fuzzy-logic control of an inverted pendulum with vision feedback', *IEEE Transactions on Education* **41**(2), 165–170.

Nise, N. S. (2010), *Control Systems Engineering*, 6th edn, John Wiley & Sons.

Ogata, K. (2001), *Modern Control Engineering*, 2nd edn, Prentice Hall.

Young, S. C. & Krishnapuram, R. (1997), 'A robust approach to image enhancement based on fuzzy logic', *IEEE Transactions on Image Processing* **6**(6), 805–825.

Zadeh, L. A. (1965), 'Fuzzy sets', *Information and Control* **8**(3), 338–353.

Chapter 3

The Twin-Rotor MIMO System

Contents

With some key aspects of Control Engineering discussed, an introduction to the plant used for experimentation follows. The aim of this chapter is to give the reader an intuitive feel for how the plant operates, whilst providing a detailed mathematical description of its dynamics. Some of the analysis techniques that were discussed in the previous chapter will be utilised to perform an in-depth analysis of the plant dynamics, endeavouring to provide insight into the magnitude of the control problem at hand.

3.1 The Twin Rotor MIMO System Model

The twin rotor MIMO system (TRMS)[1] comprises a pendulum with two rotational degrees of freedom where the pitch and yaw angles (θ and ϕ respectively) are controlled by the relative speeds of two orthogonal rotors. The setup noted and used generally throughout the literature is designed to act as a simplified version of a helicopter. As such it is comprised of a main rotor and a tail rotor each of which acts at a different distance from the pivot point. This configuration is depicted in Figure 3.1 and is referred to throughout this thesis as the *helicopter* configuration.

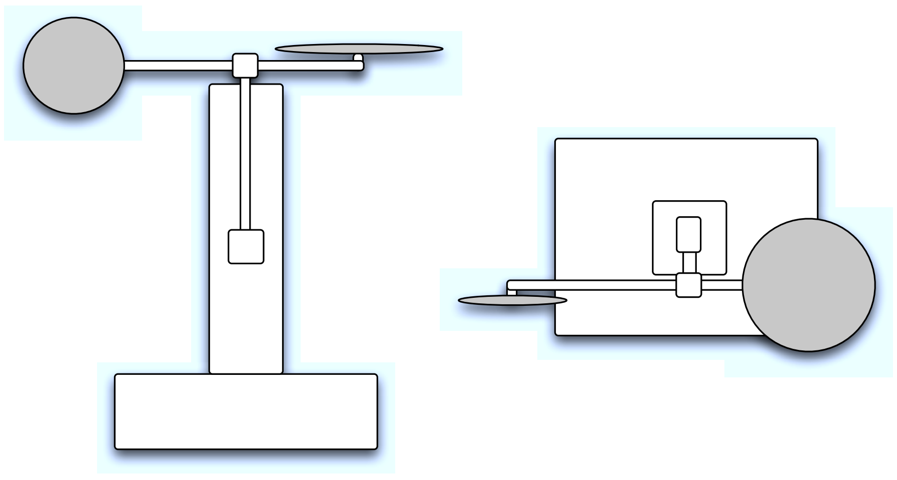

Figure 3.1: Diagram of the standard TRMS setup (helicopter configuration).

Here the fans are orthogonal to their planes of rotation, so there is minimal cross-coupling between the force generated by one fan and the plane with which it is not directly associated.

A new configuration is introduced for the purposes of this thesis. The fans are tilted out of the plane of rotation of the pendulum, the distance from each fan to the pivot point is made equal and the fans are equal in size and power. The design of the system is shown in Figure 3.2 and will be referred to as the *cross-coupled* configuration.

[1]Sometimes in the literature it is termed a twin fan system

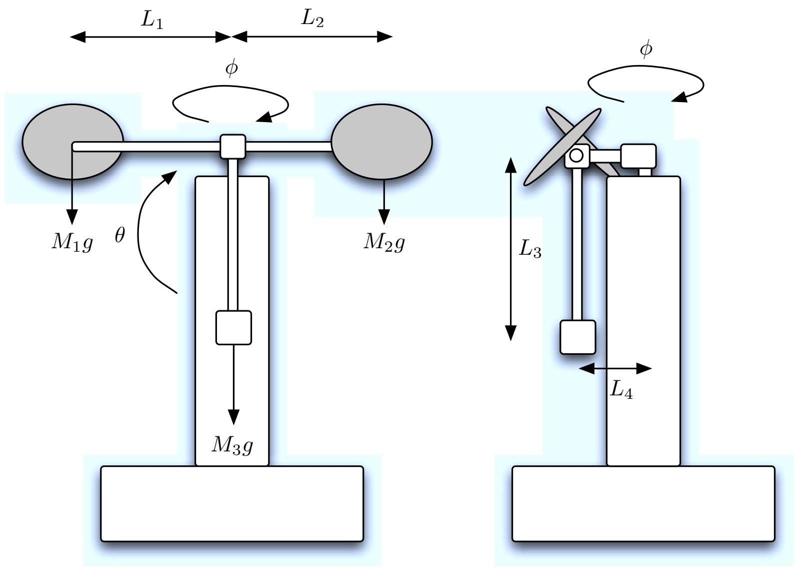

Figure 3.2: Diagram of the twin rotor MIMO system (cross-coupled configuration).

This configuration means that, although the rotors are orthogonal, they each affect both θ and ϕ when they are turned on. This produces cross couplings which make the system more difficult to control than if the rotors were aligned to affect movement in one plane of rotation. Here, the rotational conventions for θ and ϕ are positive clockwise when viewed from the front and the top respectively. The angles are measured between $\pm\pi$ with a vertical bob arm describing zero radians in θ. For ϕ, the zero radian position is arbitrary and set on startup. These conventions are shown in Figure 3.3.

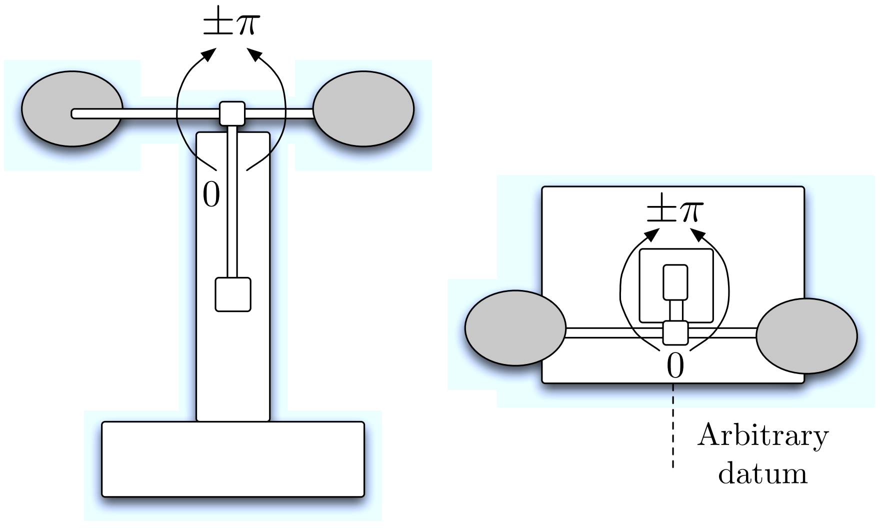

Figure 3.3: The rotational degrees of freedom of the twin rotor MIMO system.

A mathematical model of the TRMS will now be developed using the Lagrangian approach.

3.1.1 The Lagrangian Approach

The Lagrangian, $\mathcal{L}$, is defined as:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{3.1}$$

where $\mathcal{T}$ is the kinetic energy in the system and $\mathcal{U}$ is the potential energy. By summing the energies in the system, the following equation is obtained.

$$\begin{aligned}\mathcal{L} \; = \; & (\frac{1}{2}M_1(L_4^2 + L_1^2\cos^2\theta)\dot{\phi}^2 + \frac{1}{2}M_2(L_4^2 + L_2^2\cos^2\theta)\dot{\phi}^2 + \frac{1}{2}M_3(L_4^2 + L_3^2\sin^2\theta)\dot{\phi}^2 \\ & + \frac{1}{2}M_1L_1^2\dot{\theta}^2 + \frac{1}{2}M_2L_2^2\dot{\theta}^2 + \frac{1}{2}M_3L_3^2\dot{\theta}^2) - (M_1g(h + L_3 + L_1\sin\theta) \\ & + M_2g(h + L_3 - L_2\sin\theta) + M_pg(h + L_3(1 - \cos\theta)))\end{aligned} \tag{3.2}$$

The individual terms are defined in Figure 3.2. The Lagrangian equations for both θ and ϕ are as follows.

$$\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\theta}}\right) - \frac{\partial\mathcal{L}}{\partial\theta} = F_{\theta 1}L_1 - F_{\theta 2}L_2 \tag{3.3}$$

$$\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\phi}}\right) - \frac{\partial\mathcal{L}}{\partial\phi} = (F_{\phi 1}L_1\cos\theta - F_{\phi 2}L_2\cos\theta) - (F_{\theta 1}L_4\sin\theta + F_{\theta 2}L_4\sin\theta) \tag{3.4}$$

Where $F_{\phi 1}$ and $F_{\phi 2}$ are the thrusts of rotor 1 and 2 respectively acting in the ϕ plane while $F_{\theta 1}$ and $F_{\theta 2}$ are the thrusts of the same rotors acting in the θ plane. Figure 3.4 shows how the fan tilt corresponds to these values with the direction of force for F_1 and F_2, both defined as positive thrusts.

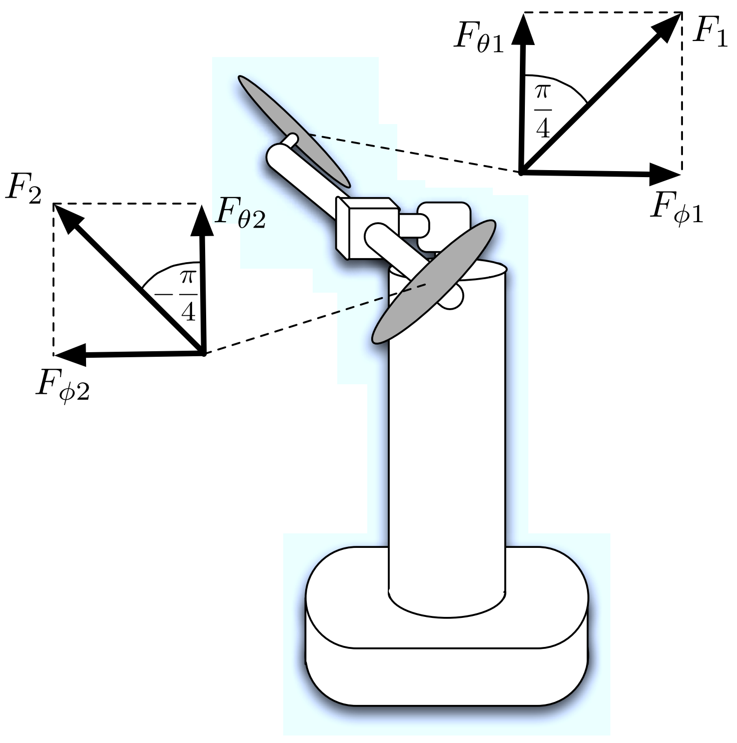

Figure 3.4: Propeller tilt affecting the force in both the θ and ϕ planes.

Since the pivot point in θ is extended away from the pivot point in ϕ by the distance L_4 it would be reasonable to expect that the moment of inertia would have also increased. Indeed, this is the case, as shown in Figure 3.5 and Equation 3.5 where L_{t1} is defined as the distance from the pivot point to fan 1.

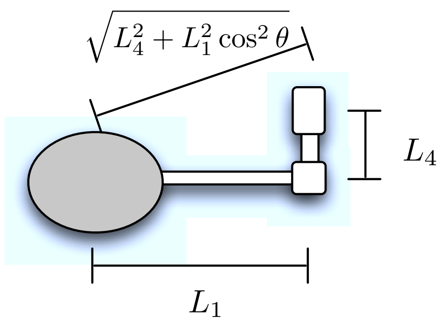

Figure 3.5: The additional distance from the ϕ pivot point due to L_4.

$$L_{t1} = \sqrt{L_4^2 + L_1^2 \cos^2\theta} \tag{3.5}$$

However, because the force is not perpendicular to the plane of rotation, as shown in Figure 3.6, the force is scaled as shown in Equation 3.8 where $F_{S\phi 1}$ is defined as the scaled force acting in the ϕ plane given an input from the fan of $F_{\phi 1}$.

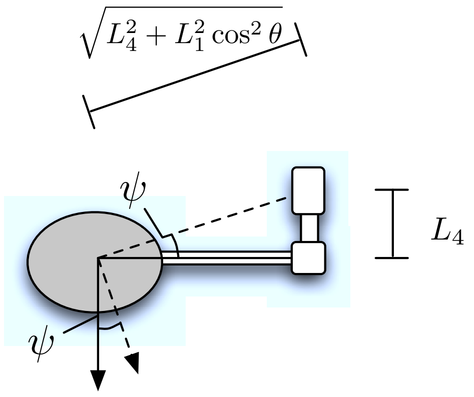

Figure 3.6: The effect of the fan due to it not being perpendicular to the pivot point.

$$F_{S\phi 1} = F_{\phi 1} \cos\psi \tag{3.6}$$

It can therefore be shown that

$$\cos\psi = \frac{L_1}{\sqrt{L_4^2 + L_1^2 \cos^2\theta}} \tag{3.7}$$

and

$$F_{S\phi 1} = \frac{F_{\phi 1} L_1}{\sqrt{L_4^2 + L_1^2 \cos^2\theta}} \tag{3.8}$$

Hence, the new formula for the moment of inertia is as follows for $F_{\phi 1}$.

$$F_{S\phi 1} L_{t1} = \frac{F_{\phi 1} L_1 \sqrt{L_4^2 + L_1^2 \cos^2\theta}}{\sqrt{L_4^2 + L_1^2 \cos^2\theta}} \tag{3.9}$$

So

$$F_{S\phi 1} L_{t1} \equiv F_{\phi 1} L_1 \tag{3.10}$$

Therefore

$$\begin{aligned}F_{\theta 1}L_1 - F_{\theta 2}L_2 &= (M_1L_1^2\ddot{\theta} + M_2L_2^2\ddot{\theta} + M_3L_3^2\ddot{\theta}) + (M_1L_1^2\sin\theta\cos\theta\dot{\phi}^2 \\ &\quad + M_2L_2^2\sin\theta\cos\theta\dot{\phi}^2 - M_3L_3^2\sin\theta\cos\theta\dot{\phi}^2) \\ &\quad -(-M_1gL_1\cos\theta + M_2gL_2\cos\theta - M_3gL_3\sin\theta) \end{aligned} \tag{3.11}$$

$$\begin{aligned}(F_{\phi 1}L_1\cos\theta - F_{\phi 2}L_2\cos\theta) - (F_{\theta 1}L_4\sin\theta + F_{\theta 2}L_4\sin\theta) &= \\ \ddot{\phi}(M_1L_1^2\cos^2\theta + M_2L_2^2\cos^2\theta + M_p(L_4^2 + L_3^2\sin^2\theta)) \end{aligned} \tag{3.12}$$

Rearranging for $\ddot{\theta}$ and $\ddot{\phi}$ gives

$$\begin{aligned}\ddot{\theta} &= \frac{F_{\theta 1}L_1 - F_{\theta 2}L_2 - \sin\theta\cos\theta\dot{\phi}^2(M_1L_1^2 + M_2L_2^2 - M_3L_3^2)}{M_1L_1^2 + M_2L_2^2 + M_3L_3^2} \\ &\quad -\frac{g(M_1L_1\cos\theta + M_3L_3\sin\theta - M_2L_2\cos\theta)}{M_1L_1^2 + M_2L_2^2 + M_3L_3^2}\end{aligned} \tag{3.13}$$

$$\ddot{\phi} = \frac{(F_{\phi 1}L_1\cos\theta - F_{\phi 2}L_2\cos\theta) - (F_{\theta 1}L_4\sin\theta + F_{\theta 2}L_4\sin\theta)}{M_1(L_4^2 + L_1^2\cos^2\theta) + M_2(L_4^2 + L_2^2\cos^2\theta) + M_p(L_4^2 + L_3^2\sin^2\theta)} \tag{3.14}$$

Having the equations in this form allows for a simulation to be constructed easily using mathematical blocks in Simulink (Mathworks Inc., 2011*b*) or simply as equations in an appropriate programming language.

3.2 Modelling the Fans

Systems with propellers are subject to time delays due to the inherent spin-up time required to produce a desired thrust. There is a non-linear relationship between the input current applied to the driving motor and the force generated by the associated rotating propeller. For modelling purposes, this is split up into three components: spin up, current to propeller speed, speed to thrust.

3.2.1 Spin-Up

The spin-up of a propeller can be reasonably modelled as an exponential approach function in angular velocity. In the time domain, this can be expressed as as $V_d(1-e^{-\alpha t})$. Where V_d is the desired angular velocity of the propeller and α is the dynamic time constant. The latter determines the speed at which the propeller approaches V_d. Figure 3.7 shows the response to a unit step function of amplitude V_d, for an exponential approach function with a time constant of 0.5 seconds.

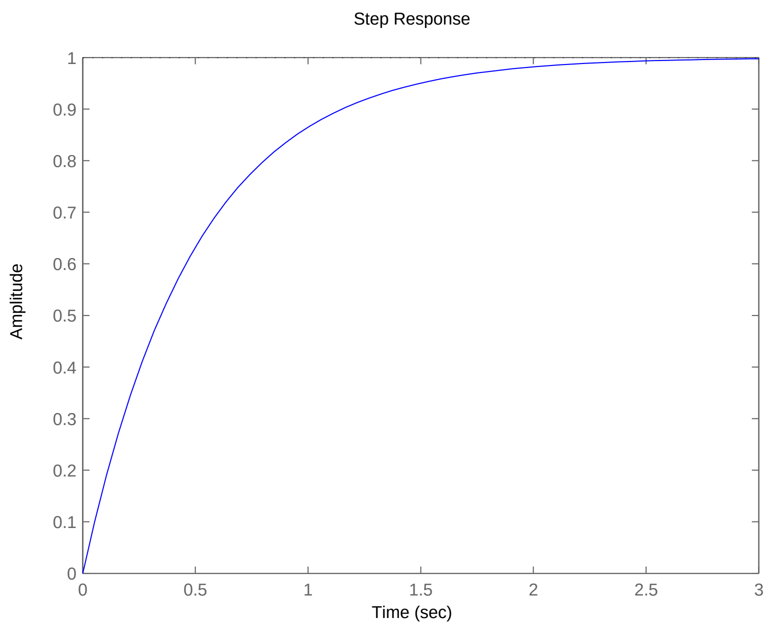

Figure 3.7: Step response to an exponential approach function with a 0.5 second time constant.

After four time constants, i.e. 2 seconds, the amplitude is within 2% of the final value, as would be expected from discussions about settling time in Section 2.5.2 of Chapter 2. When modelled in the Laplace domain the transfer function is as follows:

$$\mathcal{L}\big(1 - e^{\alpha t}\big) \cdot V_d = \frac{\frac{1}{\alpha}}{s\left(s + \frac{1}{\alpha}\right)} \cdot V_d \tag{3.15}$$

So, if the spin-up is modelled as a Simulink block the transfer function used would be:

$$\frac{\frac{1}{\alpha}}{\left(s + \frac{1}{\alpha}\right)} \tag{3.16}$$

After experimentation with the motors combined with various propellers it was found that they all took close to 2 seconds to spin up from stationary so a value of 0.5 for α was selected for the modelling process.

3.2.2 Current to Propeller Speed

As discussed previously, the relationship between the current being passed through the motors and the propeller speed are non-linear. This can be modelled in a number of ways. One possibility is to use a function to simulate the nonlinearity. Another is to use a lookup table based on measured data. The latter is used in one popular commercial demonstrator by Feedback Instruments Ltd. (2002) and subsequently by Wen & Lu (2008) who use the Feedback Instruments Ltd TRMS for their own experimentation. However, there is a difference between that system and the one developed and modelled here. The Feedback Instruments Ltd TRMS uses voltage-controlled propeller motors, whereas the author's system involves current-controlled propeller motors. Nevertheless, in modelling the TRMS used here, a lookup table was employed, based on experimentation described in Section 3.3. Once a lookup table had been established, values of speed or thrust can be determined using interpolation methods. There was scope to use a more complex lookup method. However, it was deemed unnecessary as the potential error from using linear interpolation was negligible.

3.2.3 Speed to Thrust

The non-linear relationship between the propeller thrust and angular velocity is further complicated by the propeller design itself. Since each is optimised as a 'puller', there is less thrust generated in the reverse (or 'pusher') direction. However, following experimentation and measurement, suitable look-up tables were constructed and employed in the simulation.

3.2.4 Current to Thrust

In the case of the Feedback Instruments Ltd. TRMS, the modelling of voltage to thrust was divided into two distinct non-linear equations, as described in Sections 3.2.2 and 3.2.3. Whilst the same method could be used on the TRMS model described in this chapter, the propeller testing, which will be discussed in Section 3.3, provided direct relationships between the input current and the thrust generated. This meant that a single lookup table could be implemented that accurately described the non-linearities of the fans.

3.3 Propeller Testing

In support of a different research project, a TRMS was designed and built. This meant that data could be collected from the actual system for model validation. Figure 3.8 shows a photograph of the finished system.

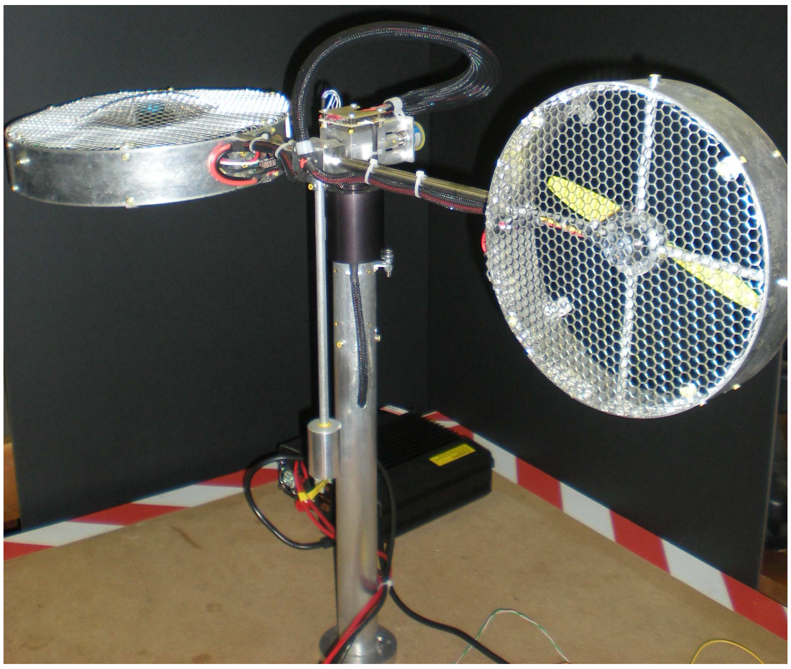

Figure 3.8: The TRMS system

As can be seen from the photograph, the real system is setup in the helicopter configuration. Nevertheless, it did prove useful to help characterise fan behaviour and validate the mathematical model. As such, experimentation was undertaken using the setup depicted in Figure 3.9. The fan forces were calculated based upon measurements taken using a load cell which was mechanically linked to the fan arm as shown.

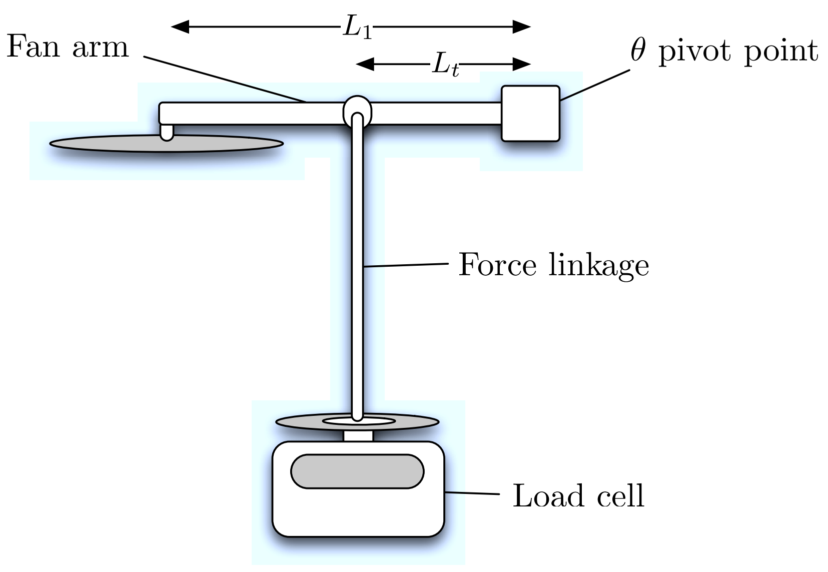

Figure 3.9: Experimental setup for testing the thrust of the fans

Following measurement, the corrected fan thrust in Newtons is given by Equation 3.17 where T_m is the load cell value scaled by the gravitational constant.

$$T_c = \frac{L_t}{L_1} T_m \tag{3.17}$$

Propeller speeds are controlled using pulse width modulation (PWM), a commonly used power modulation scheme for inertial devices. It is effectively a switch that is turned on and off for a set amount of time within a sampling window, specified by the PWM controller. The longer the switch is turned on, the wider the pulse and the more power is delivered to the fans. The percentage of the sampling window that is taken up by a pulse is known as the duty cycle.

A diagram of a PWM scheme with an increasing duty cycle is shown in Figure 3.10 where T_p is the pulse time, T_s is the sampling time of the sample window and V_p is the magnitude of the pulse voltage. The duty cycle D can be calculated simply using Equation 3.18.

$$D = \frac{T_p}{T_s} \times 100 \tag{3.18}$$

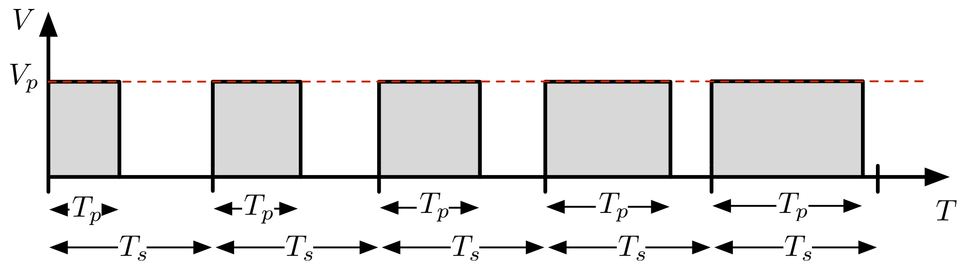

Figure 3.10: Pulse width modulation with increasing duty cycle

In the case of the TRMS the sampling window is 20ms and the minimum and maximum inputs to the windows are 1000μs and 2000μs respectively, which allows for the two servos to be controlled simultaneously within the timeframe. An input of 1000μs to the controller was set to give full power in the reverse direction, 2000μs was set to give full power in the forward direction and 1500μs was set to keep the propeller stationary.

The results of the thrust testing of the propellers is shown in Table 3.1.

Pulse Width (μs)	Current (A)	Rotation Speed (RPM)	Force (N)	Corrected Force (N)
1000	-11.4	-11140	-3.747	-1.874
1100	-5.6	-8520	-2.207	-1.104
1200	-2.7	-6350	-0.765	-0.383
1300	-0.8	-3550	-0.216	-0.108
1400	-0.4	-2300	-0.078	-0.039
1500	0.1	0	0	0
1600	0.4	2208	0.078	0.039
1700	0.8	3502	0.250	0.125
1800	2.7	6070	1.059	0.530
1900	5.9	8227	1.982	0.991
2000	12.2	11000	4.866	2.433

Table 3.1: Results of propeller testing

The absolute value of the forces generated by the propellers relative to the pulse width is shown in Figure 3.11. The use of the absolute value gives insight into the inequality of thrust between the forward and reverse directions.

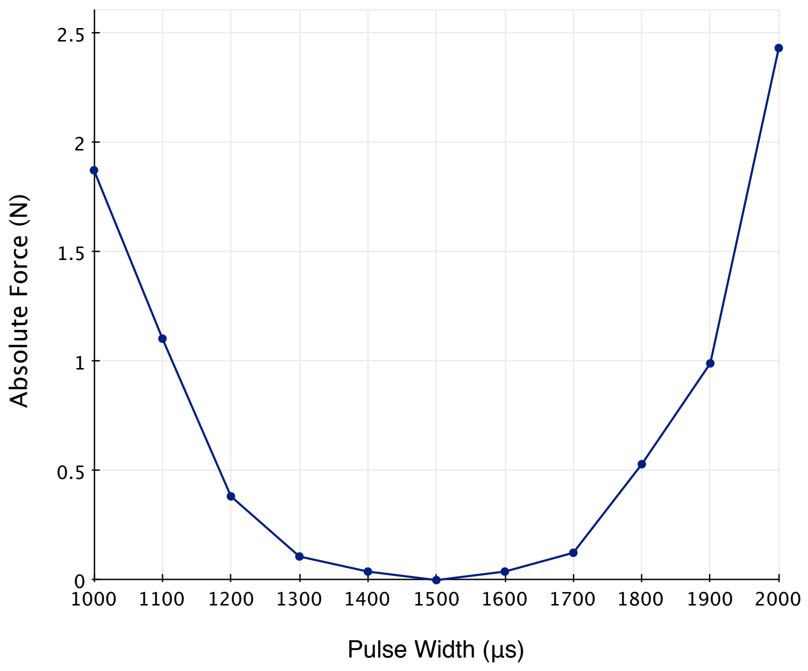

Figure 3.11: Magnitude of the fan thrusts in relation to the pulse width

3.4 Dynamic Analysis of the TRMS Model

Analysing the TRMS model dynamics proved extremely useful as it gives an intuitive perspective from which to validate that the model is correct. It provides a greater understanding of the system and why it behaves as it does, whilst producing insights into the system's operational limits. Finally, as some of the more complicated aspects of the system's dynamics are understood, a greater understanding of the control problem is formed.

This section focuses on two areas, an observation-based analysis of the system, which aims to give the reader a more intuitive feel for how the system responds, and a mathematical analysis of the system properties, based on some of the notions outlined in Chapter 2.

3.4.1 Behavioural Properties of the TRMS Model

From the equations of Section 3.1.1 we note that the value of θ affects the dynamics of the system in all planes but the value of ϕ does not. This is a reasonable observation: neither the perpendicular distances between pivot points nor the directions in which the fan thrusts operate are affected by an alteration in ϕ, except in its own reference frame. In other words, a change in ϕ only affects ϕ. A change in θ, however, alters both the perpendicular distances to the pivot point and the direction in which the fan thrusts operate in the ϕ plane.

Table 3.2 describes the fan polarities necessary to impart a change in θ and ϕ if all the other states are zero. It does not address the issue of absolute thrust nor the inequality of forward and reverse motion, simply the direction.

System Property Change	**Fan 1 Direction**	**Fan 2 Direction**
$+\theta$	+	-
$+\phi$	+	+
$-\theta$	-	+
$-\phi$	-	-

Table 3.2: Required fan direction to alter system states.

This is the case when the system is in a zeroed state because the acceleration in the θ plane is affected by $F_{\theta 1}$, $F_{\theta 2}$, gravity and friction. The acceleration in the ϕ plane is affected by $F_{\phi 1}$, $F_{\phi 2}$ and friction. When the value of θ is non-zero, the acceleration in the ϕ plane is affected by $F_{\theta 1}$ and $F_{\theta 2}$ as demonstrated earlier in Equation 3.4. The denominator defines the effect of the two fans on $\ddot{\phi}$ as represented below in Equation 3.19.

$$(F_{\phi 1} L_1 \cos\theta - F_{\phi 2} L_2 \cos\theta) - (F_{\theta 1} L_4 \sin\theta + F_{\theta 2} L_4 \sin\theta) \tag{3.19}$$

In an extreme situation, when θ is at $\frac{\pi}{2}$, as shown in Figure 3.12, there is a sign flip in the direction the fan thrust acts; the effects of $F_{\phi 1}$ and $F_{\phi 2}$ becomes zero so $F_{\theta 1}$ and $F_{\theta 2}$ dominate.

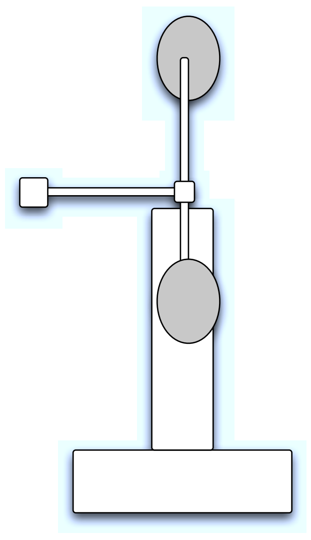

Figure 3.12: The TRMS with θ at $\frac{\pi}{2}$

It is possible to calculate the angle where this sign flip occurs by first considering how the fan thrusts affect the TRMS in both planes. When the system is at an equilibrium point with a positive θ angle, maintaining that θ angle can be achieved by simultaneously altering the fan thrusts by equal but opposite amounts. This way the forces will remain balanced in θ but not in ϕ. Hence the sign flip occurs when Equation 3.19 is equal to zero. This can be rewritten as in Equation 3.20.

$$(F_{\phi 1} L_1 \cos\theta - F_{\phi 2} L_2 \cos\theta) = (F_{\theta 1} L_4 \sin\theta + F_{\theta 2} L_4 \sin\theta) \tag{3.20}$$

Given that the TRMS is in the cross-coupled configuration and the values for L_1 and L_2 are equal, this can be rearranged as in Equation 3.21, where both L_1 and L_2 have been replaced by L_f.

$$L_f \cos\theta (F_{\phi 1} - F_{\phi 2}) = L_4 \sin\theta (F_{\theta 1} + F_{\theta 2}) \tag{3.21}$$

The values of $F_{\phi 1}$, $F_{\theta 1}$, $F_{\phi 2}$ and $F_{\theta 2}$ are calculated from F_1 and F_2 as depicted in Figure 3.4, using Equations 3.22-3.25.

$$F_{\phi 1} = F_1 \sin\left(\frac{\pi}{4}\right) = \frac{F_1}{\sqrt{2}} \tag{3.22}$$

$$F_{\theta 1} = F_1 \cos\left(\frac{\pi}{4}\right) = \frac{F_1}{\sqrt{2}} \tag{3.23}$$

$$F_{\phi 2} = F_2 \sin\left(-\frac{\pi}{4}\right) = -\frac{F_2}{\sqrt{2}} \tag{3.24}$$

$$F_{\theta 2} = F_2 \cos\left(-\frac{\pi}{4}\right) = \frac{F_2}{\sqrt{2}} \tag{3.25}$$

When these values are substituted into 3.21, it becomes

$$L_f \cos\theta \left(\frac{F_1 + F_2}{\sqrt{2}}\right) = L_4 \sin\theta \left(\frac{F_1 + F_2}{\sqrt{2}}\right) \tag{3.26}$$

or simply

$$L_f \cos\theta = L_4 \sin\theta \tag{3.27}$$

Rearranging gives

$$\theta = \arctan\left(\frac{L_f}{L_4}\right) \tag{3.28}$$

The values for L_f and L_4 in the model are $0.327m$ and $0.05m$ therefore

$$\theta = \arctan\left(\frac{0.327}{0.05}\right) \approx 1.4191^c \tag{3.29}$$

(where the superscript ^{c} denotes radians)

3.4.2 Mathematical Analysis of the TRMS Model

To enable a mathematical analysis to be carried out on the TRMS model, a Simulink block model was implemented, as shown in Figure 3.13. It comprises a transfer function block for each fan to simulate the spin up time, an embedded MATLAB (Mathworks Inc., 2011*a*) function block which contains MATLAB code to calculate the values for $\ddot{\phi}$ and $\ddot{\theta}$ from the current values of all of the inputs and a series of integrators which convert the calculated acceleration values into velocities and positions. The embedded MATLAB function block also deals with converting the fan inputs into thrusts by using linear interpolation of a lookup table as discussed in Section 3.2.4.

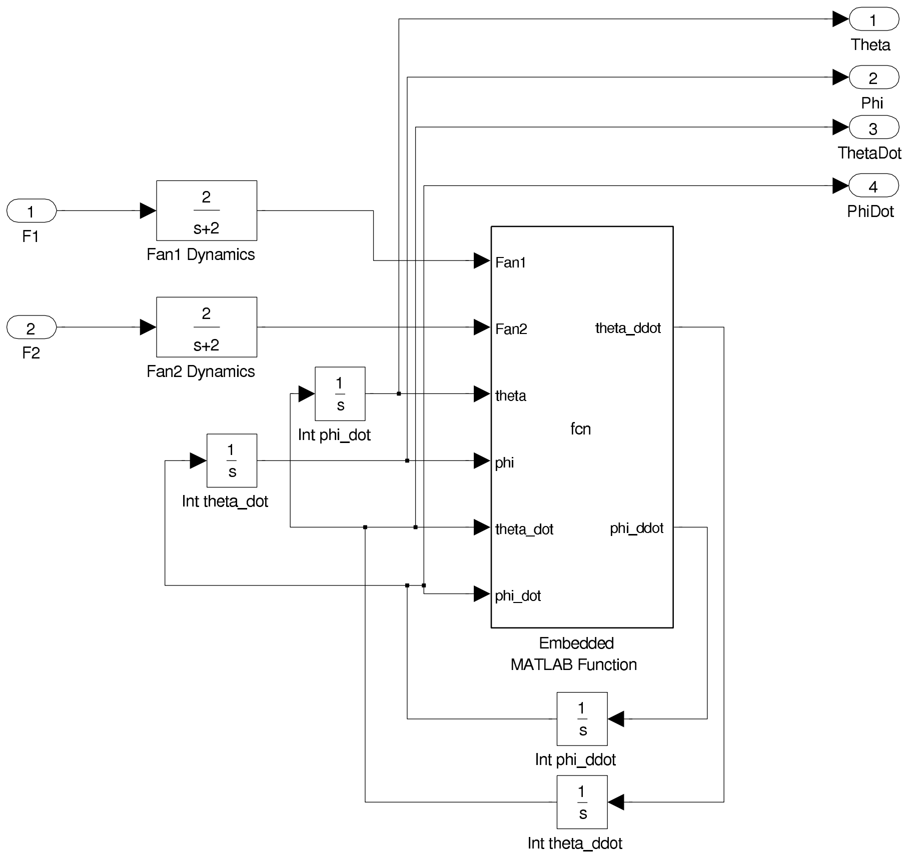

Figure 3.13: The Simulink model of the TRMS system

The development of this Simulink model allows for utilisation of the *trim* and *linmod* functions in MATLAB which calculate the required inputs for the system to remain at a particular state and then produces a linearised model about that point so that the system poles at that operating point can be elicited. For the purposes of analysis, however, the fan dynamics were removed from the system, this is because that they hinder MATLAB's ability to find the trim condition due to the additional lag they produce. This does not affect the validity of the analysis because they appear on a pole zero plot as two identical poles at $s = -2$ regardless of where the other states are. N.B. for any root locus analysis, they would need to be reintroduced. However, no such analysis is performed here.

Like most systems, the TRMS has limits on the magnitudes of the inputs based upon maximum propeller speed. Figure 3.14 shows the operational envelope of the TRMS in terms of θ and $\dot{\phi}$ where the letters **A** to **F** denote inadmissable regions. This provides information about the maximum and minimum $\dot{\phi}$ that can be achieved for a range of θ values, given these limits. It should be noted that the position in ϕ is considered unimportant in these tests as it is only the velocity $\dot{\phi}$ that is under analysis. The $\dot{\theta}$ term will always be equal to zero since, for the TRMS to be trimmed at a given θ position, it must be stationary in that plane.

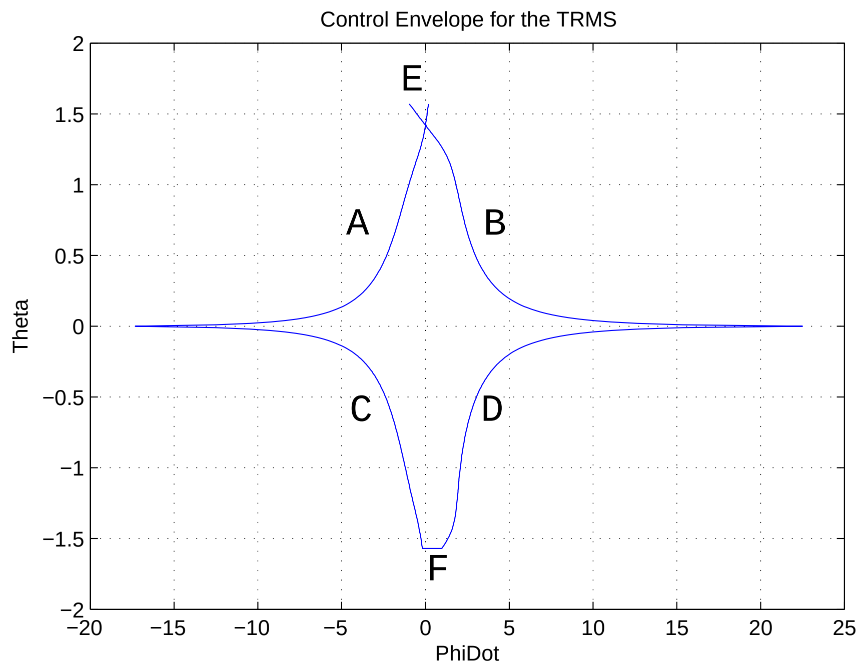

Figure 3.14: Envelope of the TRMS in θ and $\dot{\phi}$

The envelope is not symmetrical about the θ axis. The inadmissible regions **B** and **D** are further away from the $\theta = 0$ line than regions **A** and **C**. This is because of the inequality in the thrusts generated in the forward and reverse directions. Hence, greater angular velocities can be achieved in the positive $\dot{\phi}$ direction because both fans are producing larger thrusts. A point to note is that, for negative values of θ close to $-\frac{\pi}{2}$ (region **F**), there is more potential for a higher $\dot{\phi}$ value due to $F_{\theta 1}$ and $F_{\theta 2}$ aiding the acceleration in the ϕ plane instead of counteracting it.

Based on the calculation in Equation 3.29, there should be a point at $\theta = 1.4191$ (region **E**) where it becomes impossible for the fans to induce a velocity in the ϕ plane and this can clearly be seen as the maximum value of $\dot{\phi}$ decreases to zero before increasing again. Figure 3.15 shows in more detail the envelope at that point.

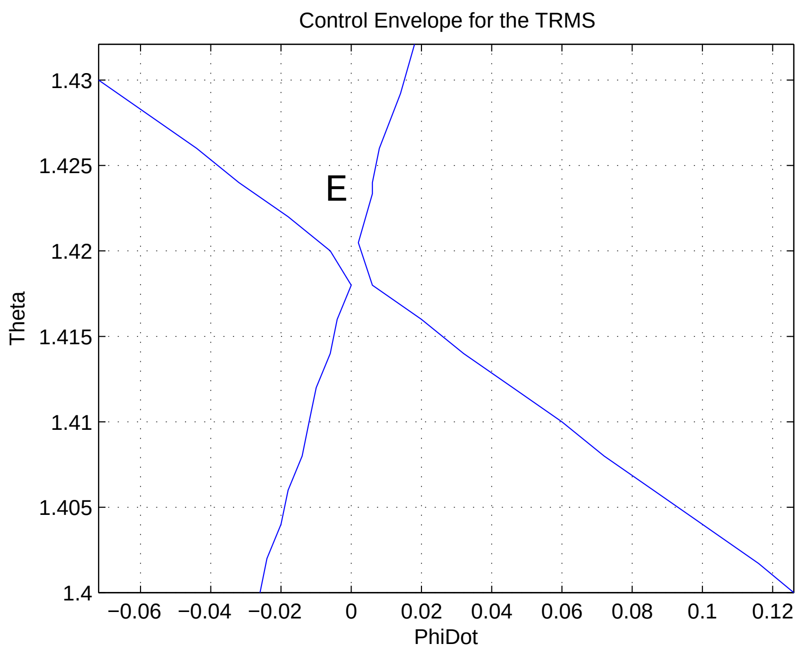

Figure 3.15: Envelope of the TRMS in θ and $\dot{\phi}$ around the control flip value (region E)

Figure 3.16 shows how the system poles of a linear approximation of the system in a specific state migrate as the value of θ ranges from 0 radians to 1.5 radians at 0.1 radian increments. The rate of pole migration can be determined by the distance between each point. It can be seen that, for lower θ values, the poles do not migrate particularly quickly. However, in the range between 0.9 radians to 1.5 radians the pole migrations are more prominent. This shows that the non-linearity of the system is more pronounced as the system is further away from its equilibrium state.

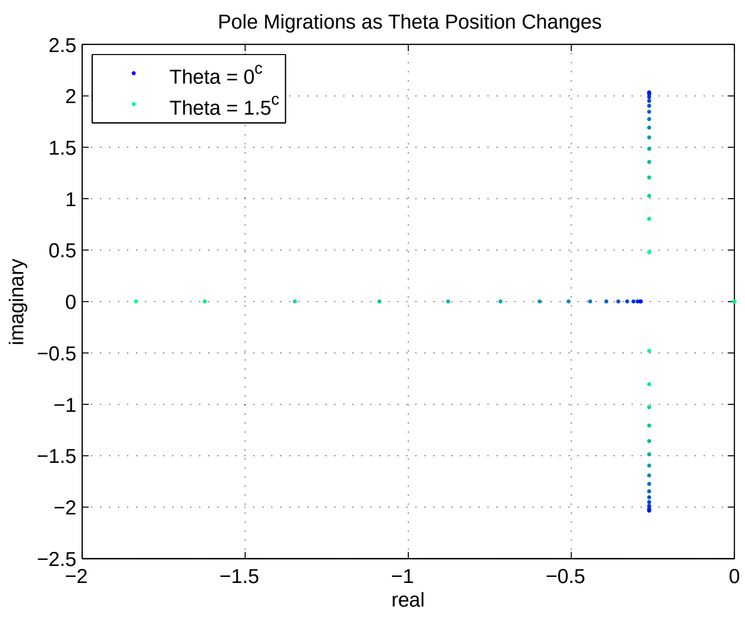

Figure 3.16: Pole migrations as θ ranges from 0 to 1.5

Figure 3.17 shows how the poles migrate as the value of θ is varied from 1.5 radians and 1.6 radians with an increase of 0.001 radians per evaluation. Given that $\frac{\pi}{2} \approx 1.57$, the speed at which the complex poles converge to become real, and one branch migrates into the right half plane of the diagram is interesting. The system has become unstable, as it moves outside of the admissible region depicted earlier in Figure 3.14.

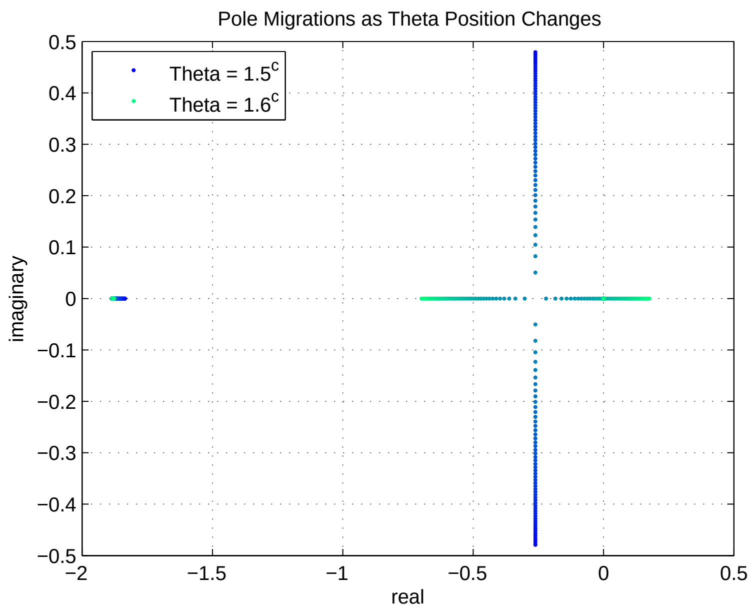

Figure 3.17: Pole migrations as θ ranges from 1.5 to 1.6

Figures 3.18 and 3.19 show the pole migrations when the range of $-1.5^c \leq \theta \leq 0^c$ and $-1.6^c \leq \theta \leq -1.5^c$ are examined.

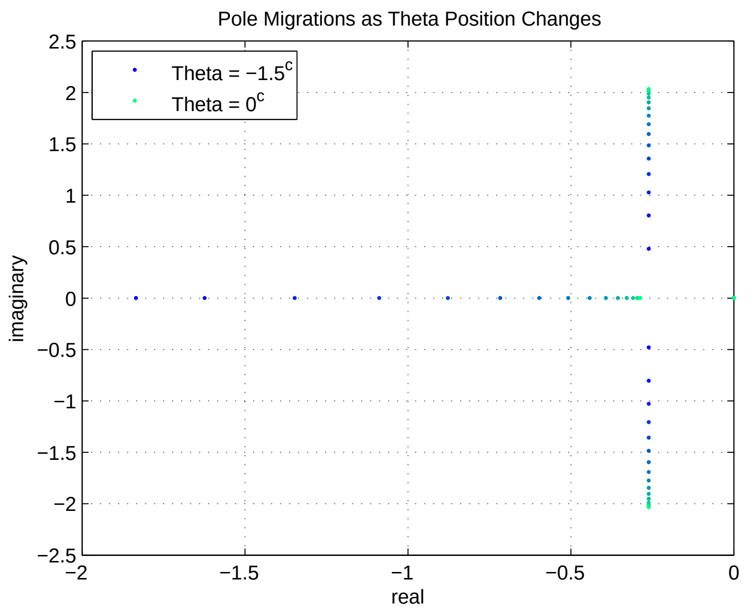

Figure 3.18: Pole migrations as θ ranges from 0 to -1.5

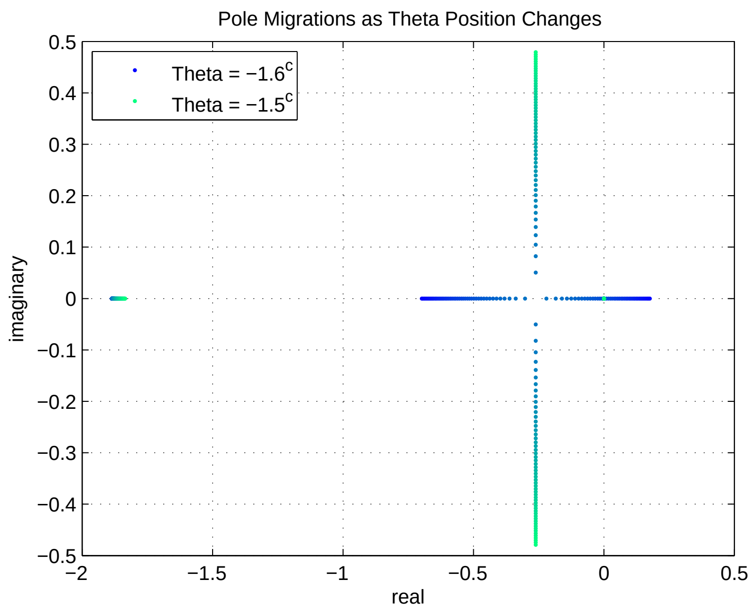

Figure 3.19: Pole migrations as θ ranges from -1.5 to -1.6

It can be seen that the migrations are very similar to those shown in Figures 3.18 and 3.17 indicating that the TRMS system properties are symmetrical about $\theta = 0$. When a similar test is performed in the ϕ plane, the poles do not migrate at all. This is to be expected as there are no ϕ terms in the system equations so changing its value should not affect the TRMS dynamics. This also makes sense from a rational perspective as the current ϕ would not alter any of the fan effects on the system, other than the ϕ position itself, and that would be based on its initial conditions.

However, altering the values of $\dot{\phi}$ does make a difference to the system dynamics, as is shown in Figure 3.20. The complex poles diverge away from the real axis which is analogous to Figure 2.7a. From this, it is possible to deduce that, as the value for $\dot{\phi}$ increases, the damping of the system decreases with the settling time remaining constant. Once again this can be rationalised by considering the centrifugal force generated by the spinning system. Because this linearised model is constructed by analysing the effects of small perturbations around the operating conditions, the velocities involved are also very small. This means that the centrifugal force adds to the force due to gravity to increase the natural frequency. However, because the magnitude of the friction force is proportional to the angular velocity and the angular velocities are negligible, the settling time does not change.

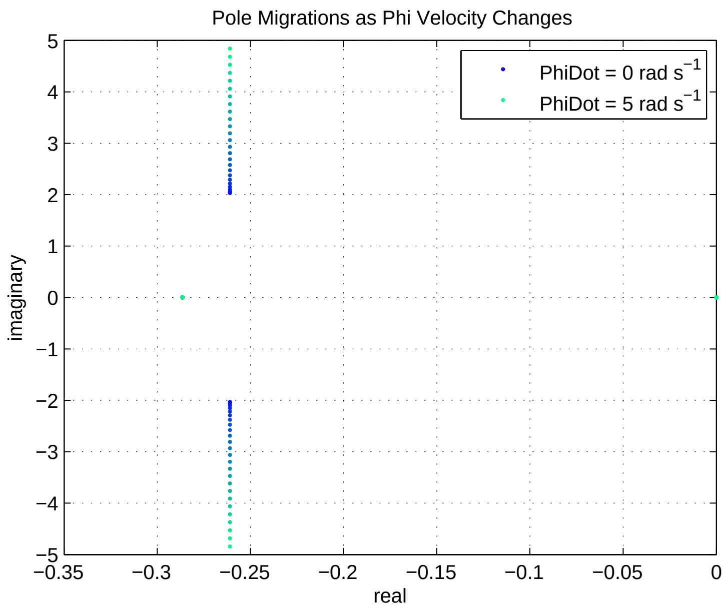

Figure 3.20: Pole migrations as $\dot{\phi}$ ranges from 0 to 5 rad s^{-1}

Figures 3.21, 3.22 and 3.23 show the effects of altering the theta angle. They depict the pole migrations when the $\dot{\phi}$ value is increased with θ set to $\frac{\pi}{32}$, $\frac{\pi}{16}$ and $\frac{\pi}{8}$ respectively.

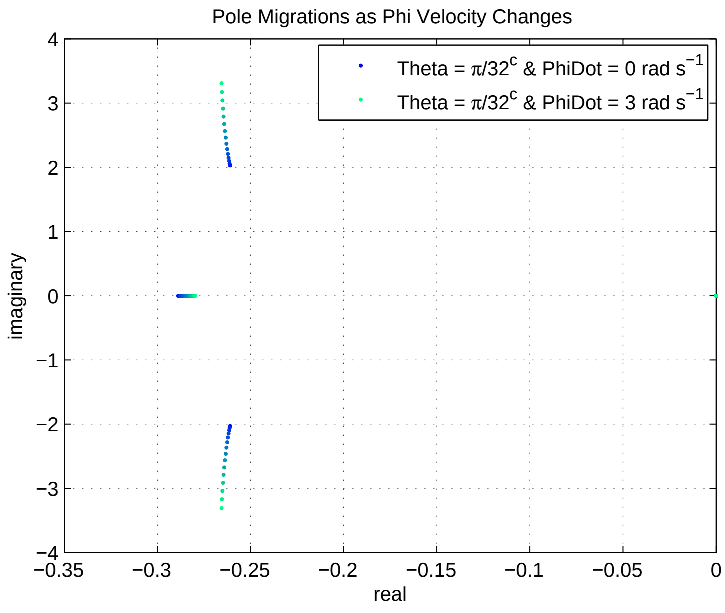

Figure 3.21: Pole migrations as $\dot{\phi}$ ranges from 0 to 3 rad s^{-1} with θ set at $\frac{\pi}{32}$

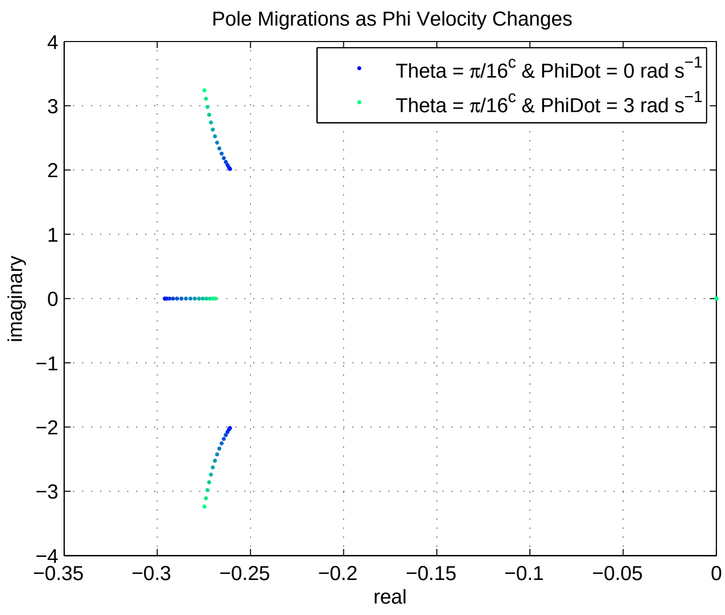

Figure 3.22: Pole migrations as $\dot{\phi}$ ranges from 0 to 3 rad s^{-1} with θ set at $\frac{\pi}{16}$

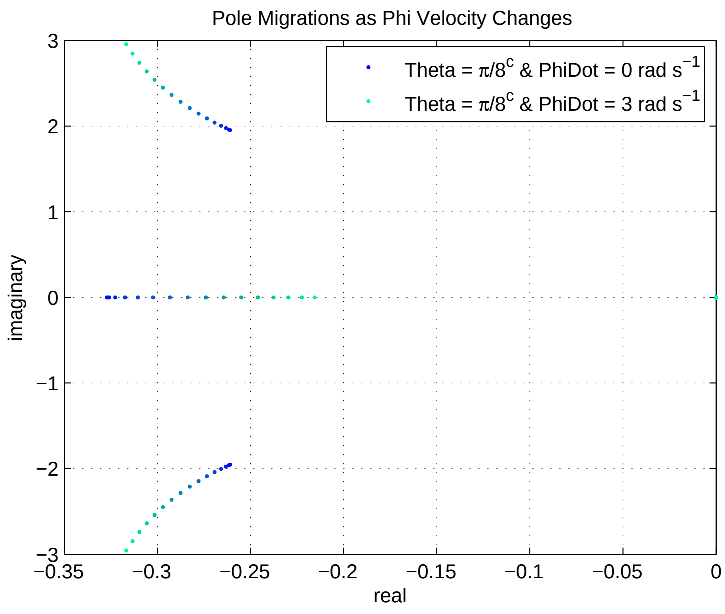

Figure 3.23: Pole migrations as $\dot{\phi}$ ranges from 0 to 3 rad s^{-1} with θ set at $\frac{\pi}{8}$

As the value of θ increases, it has a marked effect on how the poles migrate with increasing $\dot{\phi}$. The real part of the complex conjugate pole pairs becomes more negative with a larger θ value due to the centrifugal force caused by the mass distributions of the fan assemblies. However, the real pole around $s = -0.3$ moves towards the origin causing the settling time to decrease. This is due to the increased effect of the pendulum bob which creates a swing-out moment towards a θ value of $\frac{\pi}{2}$ when it is rotating.

Figure 3.24 shows the effect of altering the θ angle whilst a constant rotational velocity of 1rad s^{-1} is maintained in the ϕ plane.

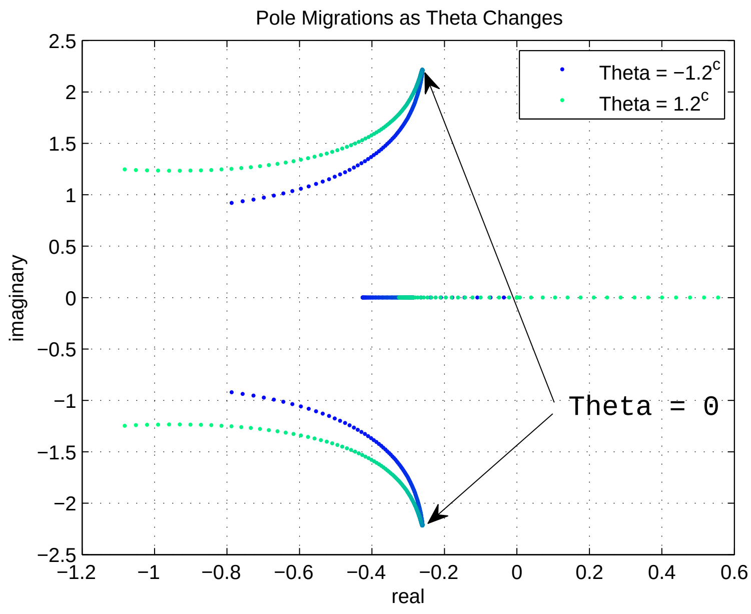

Figure 3.24: Pole migrations as θ ranges from -1.2 to 1.2 *radians* with $\dot{\phi}$ set at 1rad s^{-1}

The first thing to note is that it is not symmetrical about the $\theta = 0$ point due to the difference in the force that the fans can generate in the *pusher* and *puller* directions. However, the poles for both positive and negative values of θ move in similar ways. The complex conjugate pole pair move to increase the damping and reduce the settling time. However, the single real pole moves towards the origin, indicating a slowing down of the system dynamics and eventually indicating that the TRMS becomes unstable, particularly when θ is positive and large.

This section has investigated how the idealised linear dynamics of the system alter as its state changes. There are control engineering techniques that can be employed to tackle this non-linearity by adapting the control laws as the system state changes. Examples of such techniques include, but are not limited to, gain scheduling and sliding mode control. However, these techniques are not considered further for the purposes of this work as control of the TRMS can be achieved using simpler techniques as described in Section 3.5.

3.5 Controlling the TRMS

3.5.1 Controllers for Similar Systems

Whilst a number of controllers have been reported in the literature for the TRMS, the arrangement of the fans for the controlled system has always been in the helicopter configuration, described in Section 3.1.

Ul Islam et al. (2003) develop a fuzzy controller for a TRMS in this configuration with effective results. The TRMS responds quickly to errors in both rotational planes. However, the motivation behind the research appears to be in the furtherance of fuzzy control at the expense of more conventional control. Their fuzzy controller performs 'better' than the PID controller that they have implemented. However, the degree of optimality of the PID gain values is mentioned only briefly and they state that the gain values could be improved but "still have their limitations".

Liu et al. (2006) also produce a fuzzy controller that performs 'better' than a PID controller they have developed. The gains of the PID controller are selected using *the optimal method*. However, this method is not explicitly presented so the degree of optimality of the controller cannot be verified.

A combined fuzzy PID controller is developed by Rahideh & Shaheed (2006). Once again the fuzzy-PID controller outperforms a PID controller. The design is not documented fully so evaluation is difficult.

Despite the, sometimes, questionable design methodologies and verification strategies of these works, in all cases a controller for the system is produced which appropriately reduces errors in a timely fashion.

3.5.2 A Dual-PID Controller

To ensure the feasibility of controlling the TRMS reported earlier in the cross-coupled configuration, a PID controller was developed by splitting the system into its two planes of operation, θ and ϕ, and producing a PID controller for each. The outputs of the two controllers were then added together to produce a control scheme. Before this could be done, however, there were several modelling issues that needed addressing.

Error Correction

Since the TRMS positions in both the θ and ϕ planes range from $-\pi$ to π radians, a situation can arise where the desired position is very close to the actual position but the error term could be *perceived* as large. An example of this would be if the desired position was 3^c and the actual position was -3^c. The *generated* error term would be 6^c where, as would be obvious to anyone observing the system, the actual error would be $< -0.3^c$. The correction can easily be made using Equation 3.30 where P_e is the position error term.

$$P_e = \begin{cases} P_e & \text{if } -\pi \leq P_e \leq \pi \\ P_e - 2\pi & \text{if } P_e > \pi \\ P_e + 2\pi & \text{if } P_e < -\pi \end{cases} \tag{3.30}$$

For this to work, however, it does rely on the values for θ and ϕ being between $-\pi$ and π at all times, therefore an angle wrapping algorithm is required.

Angle Wrapping

In the modelling process, the positions and velocities of the TRMS are calculated in absolute terms, based on the acceleration calculations. For the error terms to be useful in a control sense, a wraparound function is required to ensure that the positions stay within the $-\pi$ to π bounds. Equation 3.31 provides a method for doing this using the ϕ plane as an example, where P_ϕ is the position of ϕ at a given instant and $P_{\phi wrap}$ is the corrected position.

$$P_{\phi wrap} = \begin{cases} (P_\phi - \pi) \bmod (2\pi) - \pi & \text{if } P_\phi > 0 \\ (P_\phi + \pi) \bmod (-2\pi) + \pi & \text{if } P_\phi < 0 \end{cases} \tag{3.31}$$

Building the PID Controller

Figure 3.25 shows the Simulink model of the TRMS with the PID controllers added.

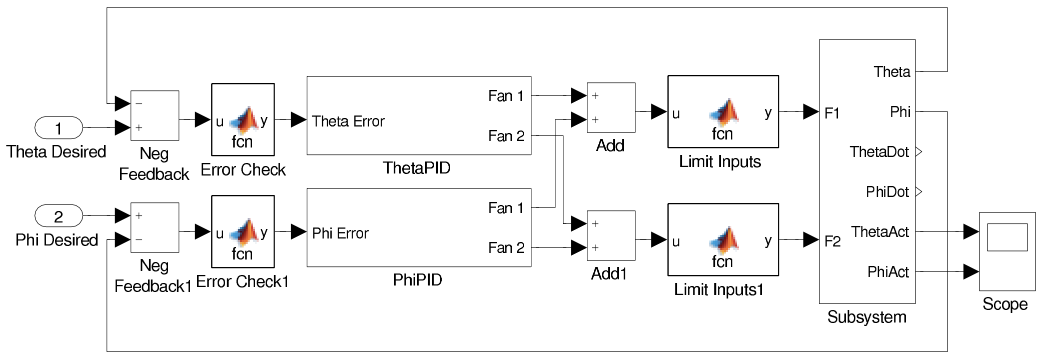

Figure 3.25: Simulink model of the TRMS with a PID controller

The *Error Check* blocks act to ensure that the errors are not altered by the sign flip about π by using Equation 3.30. The *Limit Inputs* blocks limit the maximum bidirectional input values to the fans so that they cannot exceed the 1000-2000μs range.

There are 6 outputs from the TRMS model, *theta*, *phi*, *thetaDot*, *phiDot*, *thetaAct* and *phiAct*, but *theta* and *phi* are the only outputs used in the control loop. These values relate to the angle wrapped values of θ and ϕ whereas *thetaAct* and *phiAct* relate to the unaltered values of θ and ϕ. The values of *thetaDot* and *phiDot* could have been used to feed into the PID controller as the differential term. However, since Simulink provides a derivative block, this was used instead. This more closely matches the PID structure depicted in Figure 2.9 in Chapter 2. The gain values chosen for the PID parameters are given in Table 3.3. These were achieved by manually tuning the PID parameters against the responses of the non-linear simulation.

	Theta	**Phi**
Proportional Gain K_p	400	100
Integral Gain K_i	700	50
Differential Gain K_d	500	200

Table 3.3: Gain values for the PID controller

PID Controller Results

The system was tested using an input trajectory of desired positions within the control envelope that altered every 40 seconds, allowing the system transients to settle. The results of this testing is given in Figures 3.26 and 3.27 which show simultaneous plots in θ and ϕ.

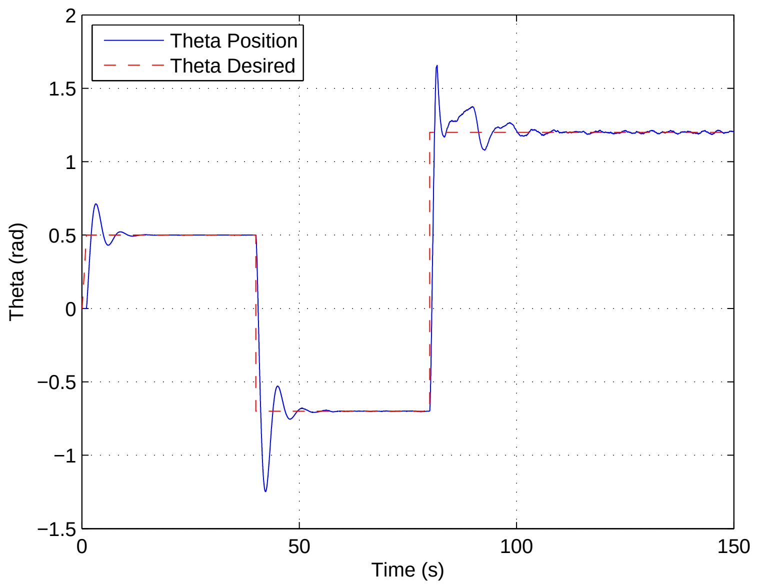

Figure 3.26: Results of the PID controller with varying θ

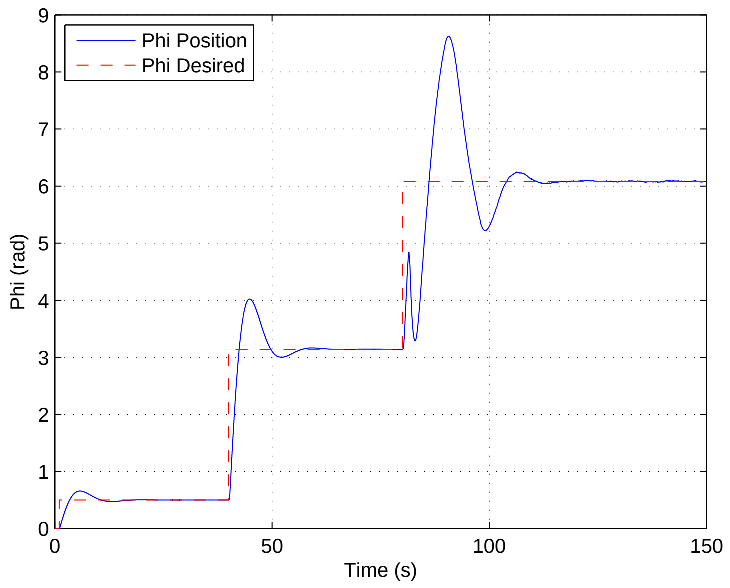

Figure 3.27: Results of the PID controller with varying ϕ

Whilst it might appear that the final settling position in the ϕ plane is around 2π it is actually an artifact of the angle wrapping and the error correction. The desired ϕ position at this point is set to -0.2 radians. However, Figure 3.27 shows the unaltered value of ϕ rather than after a wraparound function has been applied, the final position is 2π radians from the desired position which in real terms is equivalent. The error correction had an effect due to the absolute value of the error being greater than π. The Figure is shown in absolute terms due to the wraparound function producing discontinuities around the π and $-\pi$ values which makes the plot difficult to analyse.

The plots show that although some questions could be raised about the rise time and percentage overshoot values, the system does settle to the desired positions. Due to the inherent delays of the motor dynamics coupled with the non-ideal PID gain values this is an encouraging result. Using the PID algorithm, it is possible to control a *difficult* system relatively easily. The fact that the transient responses are erratic, especially for large values of θ, is understandable given the unstable nature of the poles for large θ values (see Figure 3.24). The same is true for the settling time and $\%OS$ in the ϕ plane, as shown in Figure 3.27, which steadily increases as the values for θ become more extreme.

3.5.3 Building A Fuzzy Controller

Using the same Simulink model of the TRMS, a fuzzy controller was constructed using the same proportional, integral and derivative error signals used in the PID controller, and the MATLAB fuzzy logic toolbox. Figure 3.28 shows the Simulink model of the TRMS with a fuzzy controller implemented[2].

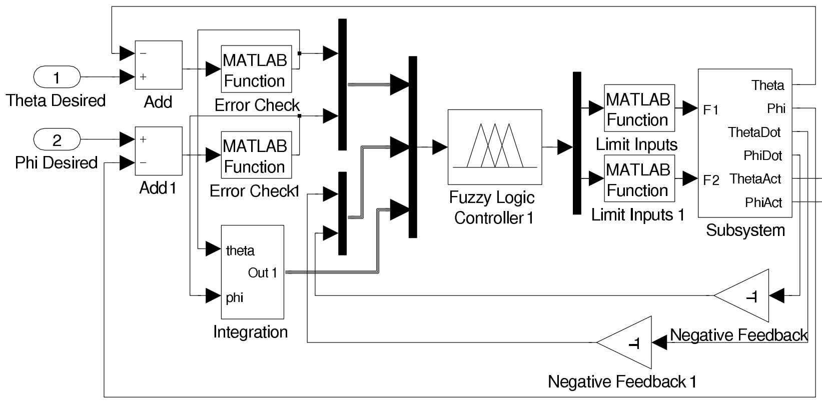

Figure 3.28: Simulink model of the TRMS with a fuzzy controller

This controller uses a centroid defuzzification procedure, with the membership functions detailed in Appendix B. The results of this controller are shown in Figures 3.29 and 3.30.

[2]Any small differences in the presentation of certain blocks in this model over the model depicted in Figure 3.25 are due to the use of different versions of MATLAB.

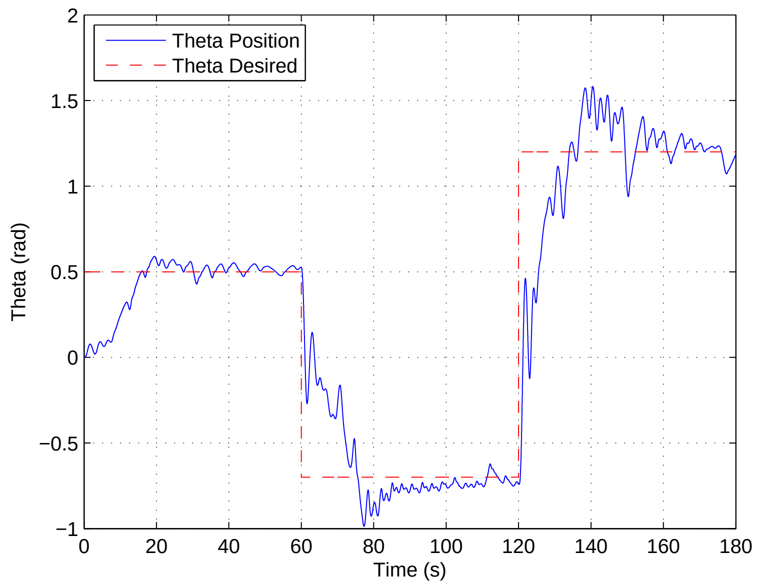

Figure 3.29: Results of the fuzzy controller with varying θ

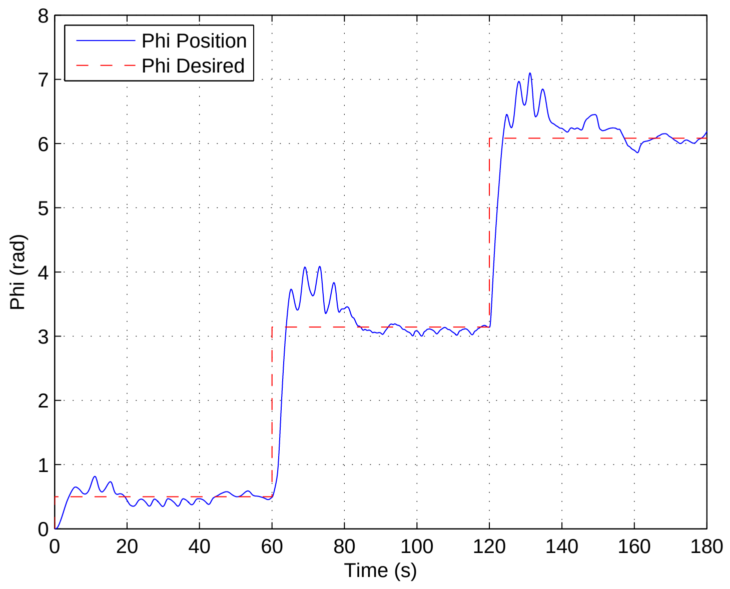

Figure 3.30: Results of the fuzzy controller with varying ϕ

As can be seen, this controller is much less effective than the PID controller previously described. It also has a longer rise time as can be seen by the 60 second windows chosen to test desired position, as opposed to the 40 second windows selected for the PID controller. However, it is very difficult to determine optimality when it comes to fuzzy control. The controller design was once again implemented heuristically so there is every chance that a more stable controller *could* be developed.

3.6 Summary

This chapter has presented the TRMS. It is a non-linear, cross-coupled system that presents a significant challenge as a target system for feedback control strategies both conventional (in the form of a PID algorithm) and less traditional (in the form of a fuzzy algorithm). The comprehensive analysis has been presented showing how the non-linearity and cross-couplings present themselves. Despite these challenges, moderately good controlled system performance has been demonstrated, although this is degraded away from the nominal design point.

So, the TRMS presents itself as a suitable and amenable challenge for a self-organising, self-learning, multi-agent controller scheme. In later chapters, this is put to the test. However, beforehand, further preliminary material is required. Chapter 4 will introduce a major component of the final scheme: reinforcement learning. Following on, Chapter 5 will discuss, within the field of distributed artificial intelligence, the topic of multi-agent systems. Highlighted in this discussion are some of the more appealing features which motivate the choice of this implementation, plus the inherent difficulties of implementing a practical, working system of communicating agents.

3.7 Chapter Bibliography

Feedback Instruments Ltd. (2002), *Twin Rotor Mimo System: Advanced Teaching Manual 1*, Crowborough, UK.

Liu, C.-s., Chen, L.-r., Li, B.-z., Chen, S.-k. & Zeng, Z.-s. (2006), Improvement of the Twin Rotor MIMO System Tracking and Transient Response Using Fuzzy Control Technology, *in* '1st IEEE Conference on Industrial Electronics and Applications', IEEE, pp. 1–6.

Mathworks Inc. (2011*a*), *MATLAB Version 7.12.0*, Natick, MA.

Mathworks Inc. (2011*b*), *Simulink Version 7.7*, Natick, MA.

Rahideh, A. & Shaheed, M. (2006), Hybrid Fuzzy-PID-based Control of a Twin Rotor MIMO System, *in* '32nd Annual Conference on IEEE Industrial Electronics', IEEE, pp. 48–53.

Ul Islam, B., Ahmed, N., Bhatti, D. & Khan, S. (2003), Controller design using fuzzy logic for a twin rotor MIMO system, *in* '7th International Multi Topic Conference', pp. 264–268.

Wen, P. & Lu, T.-W. (2008), 'Decoupling Control of a Twin Rotor MIMO System using Robust Deadbeat Control Technique', *IET Control Theory & Applications* **2**(11), 999 – 1007.

Chapter 4

Reinforcement Learning

Contents

Reinforcement learning (RL) is a large and diverse field of artificial intelligence that focusses on computational approaches to learning. Sitting within the more general class of Machine Learning, the principle behind it lies in the assignment of reward and punishment based on an agent's actions and how those actions affect convergence to a learning goal. There are many approaches to reinforcement learning but all involve an agent building some understanding of state-action pairs in order to maximise their reward. This chapter investigates the implementation of several techniques and their properties, and, much like Chapter 2, is designed as an introduction to Reinforcement Learning for readers with limited or no former experience of the field. Readers with a good understanding of the RL may wish to move on to Chapter 5.

4.1 Reinforcement Learning

The starting point for any reinforcement learning algorithm is to define a goal to be achieved and then producing an appropriate reward function based on how an action has affected the achievement of that goal. From there, a learning algorithm must be implemented such that individual agents can build up a strategy for maximising their reward, based on the state that they are in, the action taken, the transition to a new state and so on until a final state is achieved.

4.1.1 Proposing the Horizon of the RL Problem

Before discussing algorithms, it is important to define the scope of the problem and therefore how it should be viewed. For an agent to maximise reward, it must first know over what timescale the action decision is assessed. Maximising reward over a limited set of time steps is known as a *finite-horizon* model. For h steps, the expected reward for any set of actions is given by

$$E\left(\sum_{t=0}^{h} r_t\right)$$

where r_t is the scalar reward received at time-step t and E is the expectation operator. This might not always be appropriate as it might be difficult to determine an appropriate size for h in advance. An *infinite-horizon* model is an alternative. However, due to issues with infinite sums, γ is used to denote a discount factor of future expected rewards where $0 \leq \gamma \leq 1$. This means that, based on the value of γ, expected rewards in the very distant future do not outweigh more immediate rewards. Using the discounted *infinite-horizon* model the expected reward is altered to

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

The Markov Property

Something that is talked about a great deal in the field of reinforcement learning is the Markov property. Put simply, a system is said to possess the Markov property if its current state and available state actions are independent of its previous states or actions. Sutton & Barto (1998) suggest that, in the case of a flying cannonball, the position and velocity of the cannonball would be enough to satisfy the Markov property because, even though some information is not conveyed, such as how the cannonball came to be in this state, this information is irrelevant to its future trajectory. This is often referred to as an "independence of path" property. The current state is all that matters. If the Markov property is satisfied then a state transition function can be defined as a function that determines the probability of the next state being reached given the current state and the action taken. The state transition function is often written as

$$T\left(s, a, s'\right) \tag{4.1}$$

where s is the current state, a is the action taken and s' is the next state.

4.1.2 Determining a Policy from a Model

When a system model is known, there are methods for determining an optimal state-action policy given that an optimal deterministic stationary policy exists for the *infinite-horizon* discounted model (Bellman, 1957). With this in mind, an optimal value function, (which is defined as the expected maximum reward if an optimal set of actions, or policy, is taken), can be written as

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \tag{4.2}$$

where π is the full state action policy and t is the current iteration. This can be rewritten as

$$V^*(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) V^*\left(s'\right)\right), \forall s \in \mathcal{S} \tag{4.3}$$

where $R(s,a)$ is the expected instantaneous reward of action a when in state s. From this, the optimal policy $\pi^*(s)$ can be defined as

$$\pi^*(s) = arg\max_{a}\left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) V^*\left(s'\right)\right) \tag{4.4}$$

From this equation it is possible to determine an optimal policy using a variety of algorithms with a range of complexity such as the *value iteration* algorithm, the *policy iteration* algorithm and Putterman's *modified policy iteration* algorithm Puterman & Shin (1978). The details of these algorithms are not presented here, but are discussed in Kaelbling et al. (1996).

4.1.3 Model Free Probabilistic Learning

In the previous examples, the reinforcement learning techniques have all been derived from a position where the system model is known. When the system is unknown there are two options:

- **Model-free:** learn a controller without learning a model.
- **Model-based:** learn the model, and use it to derive a controller.

The following sections focus on model-free approaches as model-based approaches have not been utilised in the development of the frameworks detailed later in Chapters 6 and 7. For an overview of model-based techniques Kaelbling et al. (1996) once again provide a detailed analysis. Model-free approaches have been chosen because they enable the most generic and the simplest expositions of the learning problem in the context of this work.

All forms of reinforcement learning involve a stochastic selection procedure at some point. However, some reinforcement learning techniques employ entirely probability based selection where successful actions have their probability of selection increased.

The Linear Reward Inaction Algorithm (LRIA)

Developed by Bower & Hilgard (1980) and later discussed by Kaelbling et al. (1996), the LIRA is one of the simplest reinforcement learning techniques that can be implemented. If p_i is the probability that action a_i is taken and α is a small positive scaling factor then, when action a_i succeeds, the probabilities are updated as follows:

$$\begin{aligned} p_i &:= p_i + \alpha\left(1 - p_i\right) \\ p_j &:= p_j - \alpha p_j \quad \text{for} \quad j \neq i \end{aligned} \tag{4.5}$$

This means that when an action is successful, the probability of it being selected next time increases whilst the probability that any other action is selected decreases.

4.1.4 Delayed Reward

In a large number of cases, maximising immediate rewards is extremely limiting. Situations often arise where a suboptimal move in the short term can yield greater rewards in the long term. An example of such a situation might be in a game of chess, where sacrificing a piece could lead to a victory. One method for circumventing this problem would be to wait until the 'end' of an exploration trial and only then reward or punish good or bad actions. However, it can be difficult to identify the 'end' of an exploration trial, especially on an infinite horizon problem. A way around this is to estimate the value of the next state and provide rewards based on immediate rewards and expected rewards. This type of RL uses what are known as "temporal difference methods" (Sutton, 1988).

Temporal Difference (TD) Learning

The first challenge of TD learning is to develop an appropriate estimate for future state values. The adaptive critic element (ACE) of Barto et al. (1983) provids a starting point for what was later to be called an adaptive heuristic critic (AHC). This calculates an estimated value using an algorithm called TD(0). The structure of how a TD learning scheme can be implemented is given at Figure 4.1 (where v is the value of the current state) with its mathematical representation given in Equation 4.6.

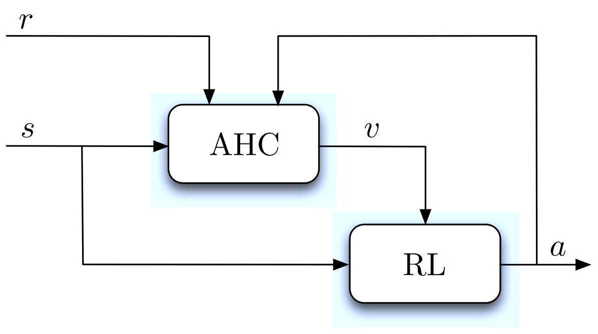

Figure 4.1: Architecture for the adaptive heuristic critic. Adapted from Kaelbling et al. (1996)

Using the AHC in this way to estimate future rewards is analogous to the shaping of control inputs described in Chapter 2. It means that the reinforcement learning algorithm works in exactly the same way except now it includes estimated future rewards when it updates, and not just immediate rewards. Whilst it might seem appropriate, since there are essentially two learning algorithms in this structure, that each algorithm learns independently of the other, most incarnations of this algorithm apply learning simultaneously. The TD(0) update rule developed by Sutton (1988) is given by

$$V(s) := V(s) + \alpha\left(r + \gamma V\left(s'\right) - V(s)\right) \tag{4.6}$$

TD(0) has been shown to converge to the optimal value function if the learning rate α $(0 < \alpha \leq 1)$ is properly adjusted (slowly decreasing) and the policy is fixed. However, TD(0) is a single example of a more general type of temporal difference method called TD(λ) where only the next step is considered when adjusting value increments. The update rule for TD(λ) is similar to that of TD(0) and is given by

$$V(u) := V(u) + \alpha\left(r + \gamma V\left(s'\right) - V(s)\right) e(u) \tag{4.7}$$

But here the update is applied to each state, denoted by u rather than just the previous state s, with the effectiveness of the update being subject to its eligibility or $e(u)$. A common definition of the eligibility is given by

$$e(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k}\, \delta_{s,s_k} \text{ , where } \delta_{s,s_k} = \begin{cases} 1 & \text{if } s = s_k \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

The eligibility of a state is defined by the amount the state has been visited in the recent past. Varying λ affects the extent to which the past visits to states affect the learning. When λ is set to 0 this becomes TD(0) learning and with λ set to 1 the eligibility is simply a factor of how many times a state has been visited. The eligibility can be updated online using the following update rule

$$e(s) := \begin{cases} \gamma\lambda e(s) + 1 & \text{if } s = \text{current state} \\ \gamma\lambda e(s) & \text{otherwise} \end{cases} \tag{4.9}$$

Q-Learning

First proposed by Watkins (1989) and refined in Watkins & Dayan (1992), Q-learning essentially combines the AHC and RL algorithms of the temporal difference methods into one scheme but further terminology needs to be introduced. $Q^*(s, a)$ is defined as the expected discounted reinforcement of adopting action a in state s and then selecting the optimal action for all subsequent states. It can therefore be stated that $V^*(s) = \max_a Q^*(s,a)$ since $V^*(s)$ is the value of s if the best action is taken initially. This allows for the recursive formula for $Q^*(s,a)$ to be written as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T\left(s,a,s'\right) \max_{a'} Q^*\left(s',a'\right) \tag{4.10}$$

This means that the selected action can simply be the maximum Q value for the current state making the update rule

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q\left(s',a'\right) - Q(s,a) \right) \tag{4.11}$$

The simplicity of Q-learning allows the state action pairs to be discretised and an $n \times m$ matrix to be formed where n is the number of states and m is the number of actions that can be performed. The learning algorithm, if assigned to an appropriate problem and properly weighted, can simply update the values in the Q matrix, selecting the maximum expected value until a solution is converged upon. It is this simplicity that makes Q-learning one of the most adopted schemes in the field of reinforcement learning.

4.2 Summary

This chapter has aimed to give the reader a grounding in the field of reinforcement learning so that the terminology can be used freely throughout the remainder of this thesis. The brevity of this chapter reflects the simplistic nature of the RL algorithm chosen for use in the framework, described in Chapter 7. However, it is important for the reader to have some understanding of potential RL implementations for a later discussion of further work. For a more detailed investigation of reinforcement learning, the reader is directed towards Sutton & Barto (1998).

4.3 Chapter Bibliography

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983), 'Neuronlike adaptive elements that can solve difficult learning control problems', *IEEE Transactions on Systems, Man, and Cybernetics* **13**(5), 834–846.

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press.

Bower, G. H. & Hilgard, E. R. (1980), *Theories of Learning (The Century psychology series)*, Pearson Education.

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), ‘Reinforcement Learning: A Survey’, *Journal of Artificial Intelligence Research* **4**(1), 237–285.

Puterman, M. L. & Shin, M. (1978), ‘Modified Policy Iteration Algorithms for Discounted Markov Decision Problems’, *Management Science* **24**(11), 1127–1137.

Sutton, R. S. (1988), ‘Learning to Predict by the Methods of Temporal Differences’, *Machine Learning* **3**(1), 9–44.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, The MIT Press.

Watkins, C. J. C. H. (1989), Learning from Delayed Rewards, PhD thesis, Kings College, Cambridge, UK.

Watkins, C. J. C. H. & Dayan, P. (1992), ‘Technical Note: Q-Learning’, *Machine Learning* **8**(3), 279–292.

Chapter 5

Distributed AI and Multi-Agent Systems

Contents

The use of distributed artificial intelligence for complex systems is very attractive due to its potential for adaptivity and robustness. For an early survey and discussion of fundamental properties see (Bond & Gasser, 1988). However, due to the additional complexity requirements at the design phase over standard engineering techniques, the development of clear design methodologies has been limited. That is not to say that distributed AI has not produced some elegant and impressive solutions to complex problems.

This chapter aims to investigate the potential advantages and the design challenges when using distributed AI systems, focussing primarily on multi-agent systems. It will also investigate some of the methodologies that have been used and their effectiveness in solving complex problems. Due to the size of the research field, this will by no means be an exhaustive survey. Nevertheless, it should provide the reader with sufficient information to understand the appeal of such an approach in the context of the work presented later.

5.1 Collective Intelligence

The term collective intelligence refers to groups of entities, be they biological or computational, collaborating to produce a system that, in itself, produces seemingly intelligent behaviour. It is an extremely simple idea and throughout the world there are many examples that rely on its use. The MIT Center for Collective Intelligence (2006) define collective intelligence as

> . . . groups of individuals doing things collectively that seem intelligent.

Whilst this might seem nondescript, any further rigidity of definition depreciates the concept and limits the number of examples that collective intelligence aptly describes. The notion of collective intelligence focusses on the principle that, in any complex system where full understanding is either impossible or impractical, a great deal of information processing can be done by multiple entities observing and interacting with the environment and communicating specific information with other entities as it is requested. This idea was formalised by Jennings (2000) who described multiple agents having a limited "sphere of visibility and influence" over an environment making collaboration critical for an agent influencing parts of the environment that are outside its sphere. Figure 5.1 illustrates the notion of an agent's *sphere of influence*.

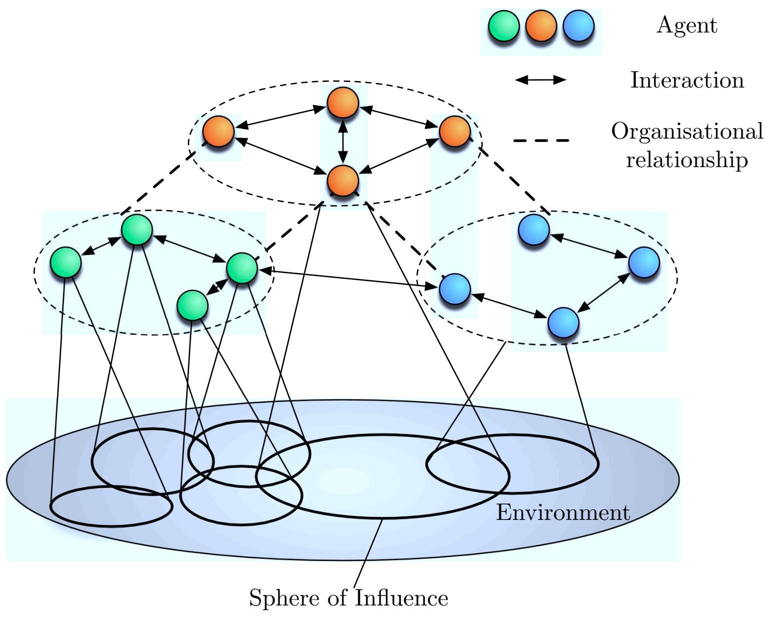

Figure 5.1: Multiple agents collectively observing and interacting with an environment. Adapted from Jennings (2000)

A structure, such as this, shows how multiple spheres of influence, combined with adequate, but not fully connected, communication protocols, can afford agents much more information about the environment without the need to monitor it in its entirety.

Possibly the most famous example of modern day collective intelligence comes from the Wikipedia Foundation (2001). Wikipedia is a free, online encyclopedia that is entirely written, reviewed and edited by the collective intelligence of its millions of users worldwide. At the time of writing, it has over 3.7 million articles in English alone and this is progressively increasing. Whilst the accuracy of some of its articles are often called into question and its authoring style is open to abuse, the wealth of information available is a triumph of collective intelligence and crowdsourcing.

Other examples emerge from the rise in popularity of social networking websites such as Twitter (2006) and Facebook (2004) over the past few years. They have given users the ability to quickly and easily share information, articles, videos and music with each other. This means that, without the need for standardised media outlets, the most interesting or shocking stories get propagated quickly.

5.2 Adaptivity

The allure of distributed AI for adaptivity is that, as long as there is no single point of failure, a system with multiple interacting components can lose functionality in some of them without losing that functionality completely. A biological example of a distributed system is nest building by large numbers of termites, all carrying out simple actions. The system can afford to lose some of the termites because the task is a combination of multiple simple actions that all of the termites have the functionality to perform.

In his PhD thesis, Mendham (2006) describes the *'Scale of Adaptivity'* as shown in Figure 5.2 where the four levels of adaptivity are defined as:

> **Autonomy:** 'A system that allows *goals* to be chosen or refined, with the provision that they do not conflict with a core set of predefined goals.'
>
> **Intelligence:** 'The ability to act appropriately in an uncertain environment, where an appropriate action is that which increases the probability of success, and success is the achievement of behavioural subgoals that support the system's ultimate goal' from (Antsaklis, 1994).

Robustness: 'The ability of a system to react appropriately to changes in operating conditions retaining correct behaviour as completely as possible. Where it is not possible to operate correctly, the system should fail gracefully in a manner appropriate to the operating conditions and the system function'; a combination of definitions by Meyer (1997) and extended by Gribble (2001).

Fault Tolerance: 'The built-in capacity (without external assistance) to preserve the continued correct execution of its programs and input/output function, in the presence of operational faults' Avižienis (1975).

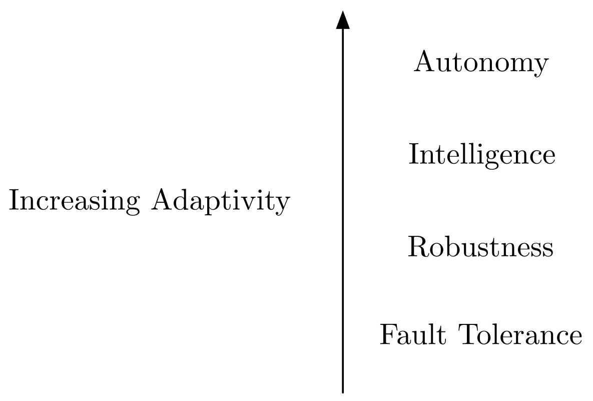

Figure 5.2: The scale of adaptivity. Adapted from Mendham (2006)

To revisit the termite example, this we would classify as a robust system since the removal of termites may slow down the production of the nest but does not halt its progress unless a critical point is reached. This would be an example of graceful degradation.

5.3 Self-Organisation

Self-organisation is apparent throughout the sciences, be it in the structures formed by molecular self assembly (for example DNA), the automatic regulation of a cellular organism or even the *invisible hand* driving economic markets to an equilibrium price, as described by Smith (1776). When describing Smith's *invisible hand*, Friedman (2006) describes it as "the possibility of cooperation without coercion". This describes the goal of designed self-organising systems so simply and completely that it is the definition that will be used throughout this thesis.

The field of computational self-organisation is focussed on the development of systems wherein the local interaction between elements within the system produces structures or processes without global organisation. The scales at which the terms **global** and **local** are defined varies greatly, depending on the system, and the scheme that is being employed. Needless to say, there is a wide variety of techniques that can be adopted to produce a self-organising system. Some of the more strident examples of algorithmic self-organisation are found in studies on swarm systems, as discussed in Section 5.4.

5.4 Swarm Intelligence

5.4.1 Particle Swarm Optimisation (PSO)

First introduced by Kennedy & Eberhart (1995), particle swarm optimisation relies upon the principles of swarm behaviour (introduced in Section 1.1 of Chapter 1) to determine the characteristics of some parameter space or function. Particles represent points within a sometimes high-dimensional parameter space. The parameters characterise the solution to some optimisation problem. A cost, or fitness, is ascribed to each particle. The objective is for the particle to migrate towards some optimal location in the parameter space under the influence of rules of interaction with other, similar particles. The underlying principle of PSO is that each particle evaluates the fitness of the parameter space at its current position. Its movement through the parameter space is determined by a combination of its own fitness history and the fitness of other, nearby particles, combined with some random perturbation. The particles do not interact directly with one another. However, the movement of particles does affect the movement of their neighbours. One of the most attractive qualities of PSO schemes is their adaptability without the need for meticulous parameter tuning. Eberhart & Shi (2001) note that

> One of the reasons that particle swarm optimization is attractive is that there are very few parameters to adjust. One version, with very slight variations (or none at all) works well in a wide variety of applications.

Another attractive quality is that PSO has the ability to track dynamic optima. Hu & Eberhart (2002) and Parrott (2006) have produced PSO schemes that locate and track multiple optima in a dynamic landscape. Whilst these examples are not targeted at dynamic control problems, they do have implications for the field of adaptive control, where degradation of sensors and actuators can trigger catastrophic failures.

PSO for Control Applications

PSO has been used in a number of research fields including communications networks, image and video analysis, power systems/plants, as detailed in Poli (2008). Analysis of the application in most of these fields is beyond the scope of this thesis. Nevertheless, the use of PSO for control applications is relevant and will be considered further.

In attempting to produce an optimal controller for an automatic voltage regulator (AVR), Gaing (2004) used a PSO technique to find PID gain values. The particles assess their fitness by analysing the closed loop time domain performance using the steady state error, the percentage error, the rise time and the settling time. The scheme does indeed produce gain values that control the AVR. However, it is difficult to establish the extent to which it is optimal as it uses a poorly documented genetic algorithm as a comparison. Whilst it is possible that this genetic algorithm is the optimal evolutionary technique for solving the parameter assignment, it seems unlikely and is therefore an inconclusive comparison. Whilst this example demonstrates some interesting characteristics, it requires, a priori, the PID structure and, as such, more closely resembles an optimisation problem. For this reason, it not self-organising and is inappropriate in the context of this work.

5.4.2 Biologically Inspired Distributed Systems

With so much apparent organisation in the world around us, from the chemical regulation of organisms to the complex interactions in herds, flocks, swarms and communities, it is no surprise that there is a wide variety of distributed AI techniques that draws inspiration from Biology. This section aims to provide an overview of some of these techniques and the situations in which they are useful.

Ant Colony Pheromone Trails

One biological example of a distributed, self-organising system is an ant colony that displays stigmergy. Ants use pheromone trails to communicate a best foraging route for the other ants to follow. The principles of the ant pheromone trail was adapted to produce an optimisation algorithm introduced by Colorni et al. (1991) and subsequently refined by Dorigo et al. (1996).

The elegance of the algorithm lies in its simplicity. Each ant, when exploring an environment, will, with a large probability, choose a path with the most ant pheromone already laid. If there is no ant pheromone already laid (or its probability-based selection algorithm selects the low probability action) the ant will choose a path randomly. In a situation where one path is shorter than another and no pheromone has yet been laid, the throughput of the shortest path will be greater than that of the longer path, leading to more pheromone and a greater likelihood that it will be selected in future. Memory in an ant colony system is introduced by implementing an evaporative decay of pheromone trails so that a trail that has been built up repeatedly will cease to be dominant if a more useful path emerges.

Using both symmetric and asymmetric travelling salesman problems (TSPs) as a testbed, Dorigo & Gambardella (1997) investigate the application of their ant colony pheromone system. The purpose of the TSP is to attempt to find the shortest distance that has to be travelled between a set of cities where each city in the list is visited exactly once. In a symmetric TSP the distance between any two cities is always the same regardless of the direction of travel. In asymmetric TSPs the distance between two cities can differ depending on the direction of travel.

Alterations are made from their original ant colony system due to the computational load when the complexity of the TSP increases. They note that,

> Although ant system (sic) was useful for discovering good or optimal solutions for small TSPs (up to 30 cities), the time required to find such results made it unfeasible for larger problems.

The updated system waits until all ants have completed their search and all ants have a complete solution, at which time a global pheromone updating rule is applied, where only the ant with the best solution is allowed to deposit pheromones.

This updated method provided impressive results. The availability of *a priori* knowledge of the system allowed for this. However, a large proportion of what made the system self-organising was lost.

5.4.3 Swarm Robotics

Steels (1990) developed a distributed agent subsumption architecture for a simulation of a collection of robot explorers on a distant planet. The structure of the subsumption architecture is detailed in Section 5.10.1. The robots' objectives were to collect precious rock samples from an unknown and treacherous terrain where communication with an operator is unfeasible due to time delays. A set of rules were developed for the subsumption architecture that provided a terrain search algorithm where, at the highest level, the robots were searching for rocks and, at the lowest, they were avoiding canyons. The communication between robots consisted of dropping and picking up crumbs left by other robots. The crumb handling algorithm is simple.

1. If I carry a sample, I drop 2 crumbs.
2. If I carry no sample and crumbs are detected, I pick up one crumb.

This method of pathfinding is analogous to ant colony pheromone trails described in Section 5.4.2. The results show a robust and seemingly autonomous system. However, this system does rely on a fairly reliable sense of where canyons are. The robustness is inherent in the distributed nature of the system since a single robot failure would not cause a complete system failure but, rather, the system degrades gracefully as robots are lost.

5.5 Multi-Agent Systems

The term multi-agent system (MAS) can be used to define many different types of systems due to the ambiguity of the term *agent*. Shoham (1993) notes the following about the term *agent*:

> Although increasingly popular, the term has been used in such diverse ways that it has become meaningless without reference to a particular notion of agenthood.

For the purposes of this thesis, the term *agent* refers to a software entity that has its own thread of execution and communicates with other agents using a well-defined communication protocol.

For many years the use of MASs to solve complex problems has been attractive due to the perceived potential for adaptivity and emergence. We next examine and analyse what is meant by these terms and how they are used across the literature. We will also examine what is meant by a MAS and how MASs have been deployed for various tasks in multiple research fields. We first introduce Agent Oriented Programming and note how it differs from Object Oriented Programming.

5.6 Agent Oriented Programming (AOP)

There are a number of similarities between Object Oriented Programming (OOP) and Agent Oriented Programming (AOP), both agents and objects use the idea of encapsulation to "hide" information from other parts of the system. The use of inheritance and message passing are often used in AOP as they are in OOP. There are differences, however. The rigidity of process flow is relaxed in AOP in an attempt to give more autonomy to the agents. This alteration in design of agents from objects is described by Shoham (1993) in his seminal paper on AOP:

> Intuitively, whereas OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and that have individual ways of handling incoming messages, AOP specializes the framework by fixing the state (now called mental state) of the modules (now called agents) to consist of components such as beliefs (including beliefs about the world, about themselves, and about one another), capabilities, and decisions, each of which enjoys a precisely defined syntax.

The main differences lie in the formalisation of the communication and the concept of generally more *human* traits such as beliefs and decision-making intent. As the line is crossed between OOP and AOP, the software designer strays away from being in full control of how the system operates at a sequential level and, instead, more closely resembles a curator of the system as a whole. This has parallels with economic theory in democratic countries where governments cannot force the population to save or invest, but they can provide incentives to drive people's choices towards a certain decision. Table 5.1 shows the differences between OOP and AOP as it was first proposed:

	OOP	**AOP**
Basic unit	object	agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, capabilities, choices, ...
Process of computation	message passing and response methods	message passing and response methods
Types of message	unconstrained	inform, request, offer, promise, decline, ...
Constraints on the message	none	honesty, consistency ...

Table 5.1: OOP versus AOP. Adapted from Shoham (1993)

This shows the principles behind AOP in its infancy and, although a relatively short time has past since this was published, it sparked great interest. Much work has since been done in progressing towards formalisation of frameworks and protocols. Iglesias et al. (1998) details some of the techniques that were being formalised for AOP, but the biggest contribution in the field was made by by FIPA, as will be described later in Section 5.7.

5.6.1 AOP for Incomplete Knowledge

As discussed in Chapter 1, difficulties arise when designing a controller for a system where some of the characteristics are unknown. However, the use of AOP has the potential to overcome some of these difficulties. In his paper on *Ascribing Mental Qualities to Machines*, McCarthy (1979) notes the following:

> To ascribe certain beliefs, knowledge, free will, intentions, consciousness, abilities or wants to a machine or computer program is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never logically required even for humans, but expressing reasonably briefly what is actually known about the state of a machine in a particular situation may require ascribing mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans and later applied to humans. Ascription of mental qualities

is most straightforward for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is very incompletely known.

This idea has parallels with the notion of fuzzy logic which were discussed in Section 2.7. The development of fuzzy controllers relies on the assignment of apparent system attributes to fuzzy sets without the necessity for full understanding of the mathematics of the system. It goes a little way towards self-organisation – but not far enough.

5.7 FIPA Compliance

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit association established in 1996 to attempt to develop a collection of standards relating to agents and agent based technologies. The process of standardisation has been somewhat mosaic with some ideas gaining traction and becoming FIPA standards whilst others, for one reason or another, have fallen by the wayside. The intricacies of these standards are not analysed in depth here. However, the core principles of the FIPA standards, which focus on agent management, agent communication and agent architecture, will be discussed.

5.8 Agent Management

For a multi-agent system to function there needs to be some sort of agent management structure which interprets the behaviour of the agents and also enables communication between them. The management of a multi-agent system is defined in FIPA (2002*b*), with Figure 5.3, adapted from Bellifemine et al. (2007), giving insight into this structure and how multi-agent systems operate. The terminology is described in the following text.

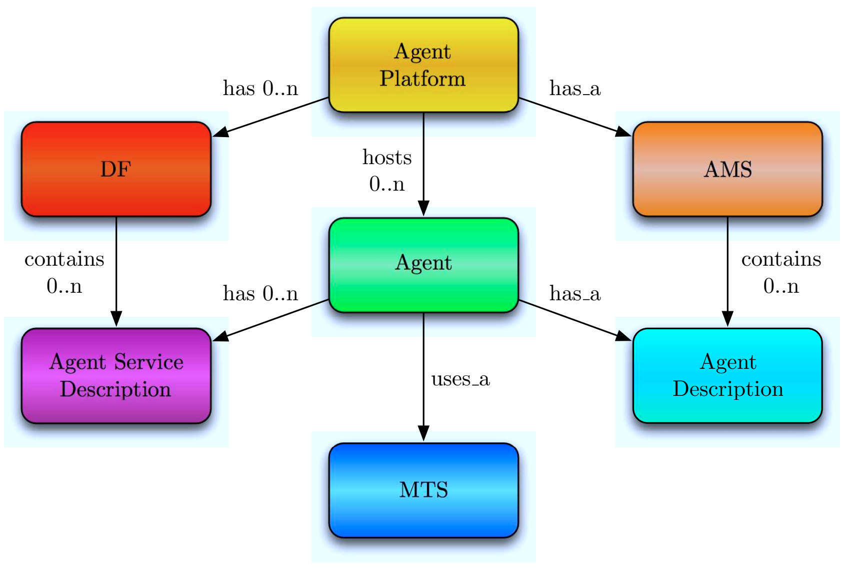

Figure 5.3: Depiction of the agent management ontology. Adapted from Bellifemine et al. (2007)

5.8.1 Agent Platform (AP)

The Agent Platform is the framework in which the agents are deployed. It can run on a single machine or across multiple computers. The *implementation* of an AP is not standardised, so the interpretation of how it should be designed is left to the developers of the multi-agent system.

5.8.2 Agent

In terms of defining what an agent is or does, there is, at first glance, surprisingly little standardisation by FIPA. However, this is an intentional choice so as to give developers greater freedom to design agents to suit the system. An agent is a software entity that inhabits the AP. The communication between agents is standardised, but the functionality and implementation are, once again, left to the developer. However, FIPA does define that an agent must have some form of unique identification code and the FIPA Agent Identifier (AID) is proposed as a solution. It is a requirement that the `name` parameter of an AID is present and unique, with optional `address` and `resolvers` allowing for differing methods of communication to be available to other agents wanting to send a message to that agent. The simplest methods for construction of an AID is described by the `name` parameter and the address of the local host where the agent resides separated by an @ symbol. More information on the AID parameter can be found in FIPA (2002*b*).

5.8.3 Directory Facilitator (DF)

Whilst not being an essential component of a multi-agent system, the DF is a useful tool when developing multi-agent systems. It essentially acts as a yellow pages service, allowing agents to search for other agents that can perform the services that they require. This allows for a developer to create agents with different capabilities without the need to keep every agent updated on which agents can perform what service.

5.8.4 Agent Management System (AMS)

The AMS, as the name suggests, deals with the actual management of the agents in the AP. It is responsible for creating and deleting agents in the AP. It keeps track of all of the AIDs of all of the agents in the system, along with their current state.

5.8.5 Message Transport System (MTS)

The MTS is responsible for communications between agents on an AP using an agent communication channel (ACC). It also uses a message transport protocol (MTP) to handle the delivery of messages between APs if there are more than one. An ACC and MTP are analogous to a national mail service and an aeroplane where the national mail service ensures the desired recipient receives their message and the aeroplane is a transport medium between countries. The MTS is further analysed in Section 5.9.

5.9 Agent Communication

Given the level of ambiguity about what an agent is, it would be possible to develop any kind of inter-agent communication protocol so long as both the sender and the recipient use the same encoding/decoding process. However, due to the nature of multi-agent systems being often difficult to design, implement and debug, a communication protocol that is illegible to the programmer can be problematical. For this reason, agent communication languages are predominantly based on speech act theory, (Searle, 1969), which posits that intent can be derived from simply specified performatives. This allows the system designer to be able to read and understand the messages being sent from one agent to another and decipher their meaning.

5.9.1 The FIPA Agent Communications Language (FIPA-ACL)

The FIPA Agent Communications Language (FIPA-ACL) (FIPA, 2002*a*) uses a set of performatives to define the kind of message that is being sent. The performative is the only required part of an ACL message. However, it is assumed that there will also be, at least, a sender, receiver and message content. Table 5.2 shows all of the fields that can be used in an ACL message.

Parameter	**Category of Parameters**
`performative`	Type of communicative acts
`sender`	Participant in communication
`receiver`	Participant in communication
`reply-to`	Participant in communication
`content`	Content of Message
`language`	Description of Content
`encoding`	Description of Content
`ontology`	Description of Content
`protocol`	Control of conversation
`conversation-id`	Control of conversation
`reply-with`	Control of conversation
`in-reply-to`	Control of conversation
`reply-by`	Control of conversation

Table 5.2: FIPA ACL message parameters. Adapted from FIPA (2002*a*)

The ACL messages are delivered by the message transport service (MTS), as described in Section 5.8.5 using a FIPA standardised message transport protocol (MTP). Details of the MTP are not covered; the inner workings of the protocol do not aide understanding of the developed systems. Detail can be found in FIPA (2002*c*).

The structure of a FIPA standardised message is depicted in Figure 5.4

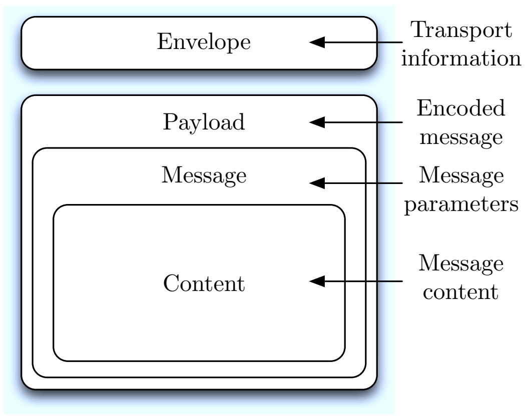

Figure 5.4: FIPA message structure. Adapted from Bellifemine et al. (2007)

Despite the rigidity of this message structure, it is still necessary for agents to be sure that they can understand one another, so ontologies must be introduced to ensure that they all understand a parameter to mean the same thing. A note in FIPA (2002*a*) states that:

> The `ontology` parameter is used in conjunction with the `language` parameter to support the interpretation of the content expression by the receiving agent.

There is no specification for the format that ontologies must take or even how they are interpreted by the agent. In the context of this work, ontologies have been employed to ensure that information communicated between agents is properly and appropriately interpreted and handled.

5.9.2 The FIPA Contract Net Protocol

Another of the FIPA standards is called the Contract Net Protocol FIPA (2002*d*), which gives agents a standardised method for negotiation. A diagram of the structure of this protocol is given at Figure 5.5. The terminology is explained in the text.

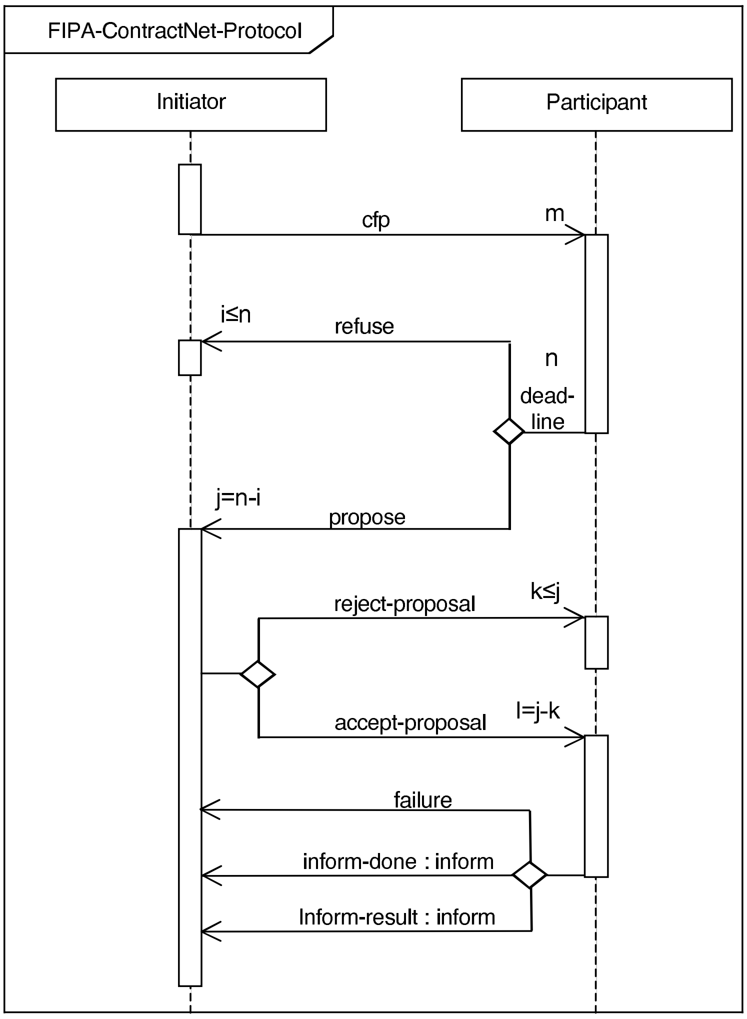

Figure 5.5: Contract Net Protocol. From FIPA (2002*d*)

This structure allows for an agent (the Initiator) to *shop around* for a best price before accepting a service proposal. For this protocol to be useful it assumes that there is a single initiator agent and multiple participant agents take part.

Call For Proposals (CFP)

The initiator agent first sends out a call for proposals (CFP) message to all the participants to see which one can provide the service the initiator is requesting for the best price. It is assumed that the initiator knows the details of the participant agents and that they are able to provide the required service, whether this be from previous experience or using a DF service within the MAS itself. The CFP messages include a *reply-by* field which specifies a deadline by which proposals should be received.

Proposals

Once a participant has received a CFP, it is up to them to decide whether to propose an offer to the initiator. The decision-making process for this action is unspecified, but if the participant decides to make an offer, and it is before the *reply-by* deadline, it replies with a *propose* message to the initiator. This message will usually contain a price, or utility, that the participant wishes to receive as payment which the initiator can then use to determine the best offer.

Handling Proposals

The next stage of the contract negotiation takes place when either all of the participants that were sent a CFP message have responded, or the deadline has passed. If the deadline has passed, then the initiator uses any proposals it has received, within the time-frame, to make a decision. Depending on the task that is being contracted out, it is possible for more than one participant to be selected for a task. However, usually only one participant is selected and, to them, an *accept-proposal* message is sent. To all of the other participants, a *reject-proposal* message is sent.

Informing the Initiator of the Result

At this point, the contract has been successfully negotiated and all that remains is for the participant to attempt to complete the task and report back to the initiator on the outcome. This can be done in three ways, depending on the nature of the task: a *failure* message if the the task is Boolean in nature and could not be completed; an *inform-done* message if the the task is Boolean in nature and has been completed successfully; an *inform-result* if the task is not Boolean in nature and has result that needs conveying.

Exceptions to the Protocol Flow

The participant in a contract net negotiation can, at any point, send a *not-understood* message to the initiator to cancel their involvement with the task. The initiator can also cancel the interaction by sending a *cancel* message using the same conversation id. This allows for both parties to reset the protocol to its initial state and prepare for a new negotiation to be initiated.

5.10 Agent Architecture

In Section 5.8.2 it was stated that there is very little FIPA standardisation of how an agent should be implemented, granting a lot of flexibility to the developer. This section investigates some agent design methodologies.

5.10.1 Reactive Agents

A reactive agent will make action decision based entirely on the state of the environment, rather than using deductive reasoning techniques to determine the best action. This is often achieved using a lookup table, which matches states to actions. However, there are other methods that can be used.

Subsumption Architecture

The subsumption architecture is a commonly-used framework for developing reactive agents in a multi-agent system. It was developed by Brooks (1991), with key principles being introduced in Brooks (1986), and states the following three hypotheses:

1. Intelligent behaviour can be generated without explicit representations of the kind that symbolic AI proposes.
2. Intelligent behaviour can be generated without explicit abstract reasoning of the kind that symbolic AI proposes
3. Intelligence is an emergent property of certain complex systems.

The principle behind the subsumption architecture is simple; the behaviour of an agent is determined by the state it is in. However, behaviours are arranged into layers which determine their priority. The action taken will always be that of the highest priority state whose conditions are not met. An example of a subsumption architecture was first introduced in Brooks (1986) for robust control of a mobile robot that seeks to develop maps of its surroundings by exploration of its environment. A diagram of the control system is depicted at Figure 5.6.

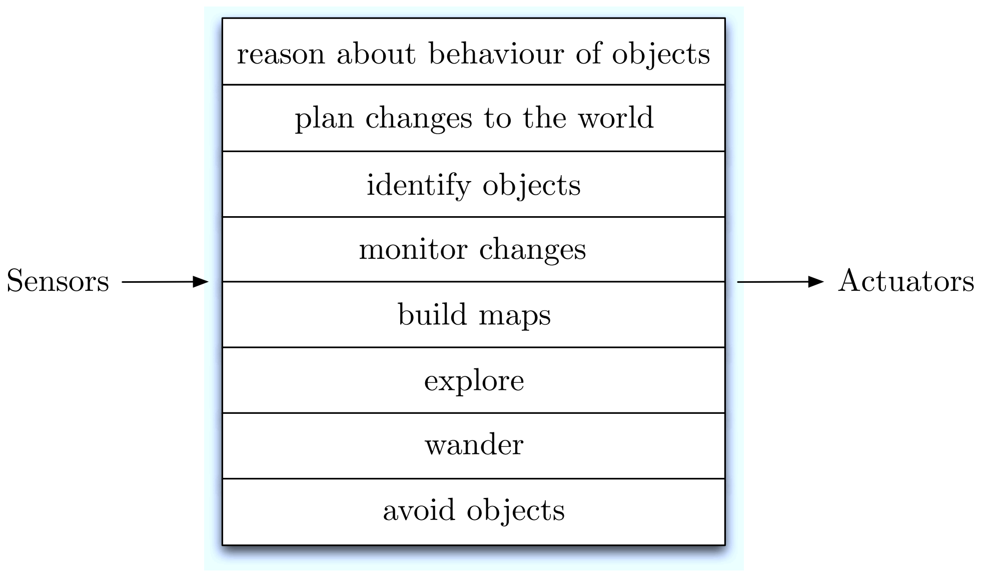

Figure 5.6: Subsumption architecture for a mobile robot control system. Adapted from Brooks (1986)

The power of the subsumption architecture lies in its simplicity, both computationally and conceptually. It can be used to produce seemingly intelligent behaviour from simple rules, similar to the principles discussed in Section 1.1.

Fuzzy Rule Sets for Decision Making

Another approach that falls under the area of reactive agents is the use of fuzzy rules to determine the most appropriate action. This can be implemented by developing a rule set for each action and, based on a defuzzification process, determining which action has the highest output and hence the most urgency. This was the approach taken by Pay (2008) when developing a behavioural action decision process for lions and wildebeest in a predator-prey savannah landscape simulation.

The rule sets were informed by animal state. Parameters such as *fear*, *hunger* and *tiredness* were inputs to the fuzzy inference engine which gave a score to each action which included *eat*, *rest*, *run* for the wildebeest and additional actions such as *hunt* for the lions.

Whilst the resulting simulation did model the behaviour of the animals fairly well, the more engaging result of the research was the ability to alter and fine tune the emergent behaviour of many agents in the system with minor changes to the rule set.

5.10.2 Reasoning Agents

Reasoning agents are most commonly split into two categories: deductive reasoning agents and practical reasoning agents. Deductive reasoning agents act as theorem provers, purely using logical operators to come to a conclusion about a situation, based on the information they have. Practical reasoning agents may also use logical operations to come to conclusions about certain aspects of the situation, but there is also an aspect of balancing conflicting decisions towards an end goal. In this sense, it is much more goal-orientated than deductive reasoning. Bratman (1990) describes practical reasoning as follows:

> Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what it believes.

Whilst the study of deductive reasoning is interesting, it falls more into the category of predicate logic and will not be considered further. Therefore, from this point onwards, the use of the term reasoning agents will refer to practical reasoning agents.

BDI architecture

The abbreviation BDI denotes the three attitudes Beliefs, Desires and Intentions and is grounded in the theory that intent, beliefs and desires are inexorably linked. An example of how this is characterised might be a man deciding whether or not he wants to buy an apple. If he has a desire to eat an apple, believes apples to be cheap and his wealth to be large enough then his intention may well be to buy one. However, if the apple is replaced with a Ferrari motor car, then he may still have a desire to buy the car, but if he believes it to be expensive and his wealth too small for such a purchase, his intention would be not to buy it. This is obviously a very simple example, but it gives some scope to the terms. However, it does rely, as is pointed out by Bratman (1987), that "...our commonsense conception of intention is inextricably tied to the phenomena of plans and planning". In other words, an agent needs to have some concept of the consequences of performing certain actions, otherwise it would be incapable of doing any such thing.

A structure for a BDI architecture was first proposed by Bratman et al. (1988) as a way of modelling *rational* behaviour and attempting to provide a methodology for "automating means-end reasoning" within the AI community. In their paper, the concept of *plans* was also proposed, comprising sets of intentions so that the expected rewards of the *plans* could be analysed for the greatest utility. The structure was highly theoretical with implementation details left unspecified. When attempting to plug the gaps in the BDI architecture, Rao & Georgeff (1995) identify and summarise two criticisms of the structure so that they can be overcome:

> First, the having of these three attitudes is attacked from both directions: classical decision theorist and planning researchers question the necessity of having all three attitudes and researchers from sociology and Distributed Artificial Intelligence question the adequacy of these three alone. Second, the utility of studying multi-modal BDI logics which do not have complete axiomatizations and are not efficiently computable is questioned by many system builders as having little relevance in practice.

The testbed for their scheme was an air traffic control system which provided a setup that is both nondeterministic in its plant and in its interaction. There are potentially many different solutions to problems and often conflicting objectives. Plus, the system state can only be sensed locally rather than with a single global observer. This setup is similar to the example depicted in Figure 5.1 of Section 5.1. They detail different ways of representing the connection between beliefs, desires and intentions and use decision trees to link them so that agents can perform actions quickly and re-evaluate if the system state changes. This work, whilst focussed on a specific system, takes big steps towards formalising a structure for building BDI systems. However, it does emphasise the need for application based design rather than a one size fits all solution.

5.10.3 Hybrid Agents

Hybrid agents combine the features of reactive agents and reasoning agents for situations where the requirements state that they must be capable of both *reactive* and *proactive* behaviour. This inherently leads to the need for some sort of behaviour structure so that an agent can switch between reactive and proactive behaviours. Müller et al. (1995) introduce a layering method, either horizontal or vertical for implementing hybrid agents, as shown at Figure 5.7.

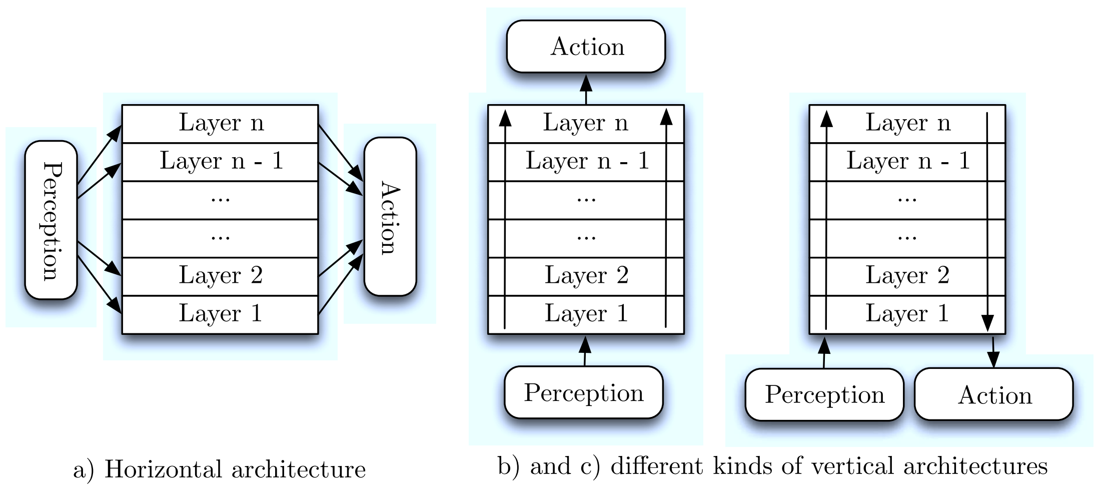

Figure 5.7: Hybrid agent architectures with layered action suggestions. Adapted from Müller et al. (1995)

In these examples, it is assumed that at least one of the layers will be reactive and at least one will be proactive. In Figures 5.7b and c, the control passes through all of the layers with the perception and action only interfacing with one of the layers. This allows for each layer to provide information about the strategy so that the layer that deals with action output can make a decision on what action to take. This vertical structuring provides a distinct hierarchy but lacks robustness as failure of a single layer would cause the structure to fail completely.

For the horizontal layer in Figure 5.7a, Wooldridge (2009) notes that it has appeal in its structural simplicity but that "because the layers are each, in effect, competing with one another to generate action suggestions, there is a danger that the *overall* behaviour of the agent will not be coherent." For this reason, a *mediator* is often required to decide which layer has priority at a given time.

An example of a horizontal layered hybrid agent architecture is called the TouringMachines architecture introduced by Ferguson (1992). The structure is depicted at Figure 5.8.

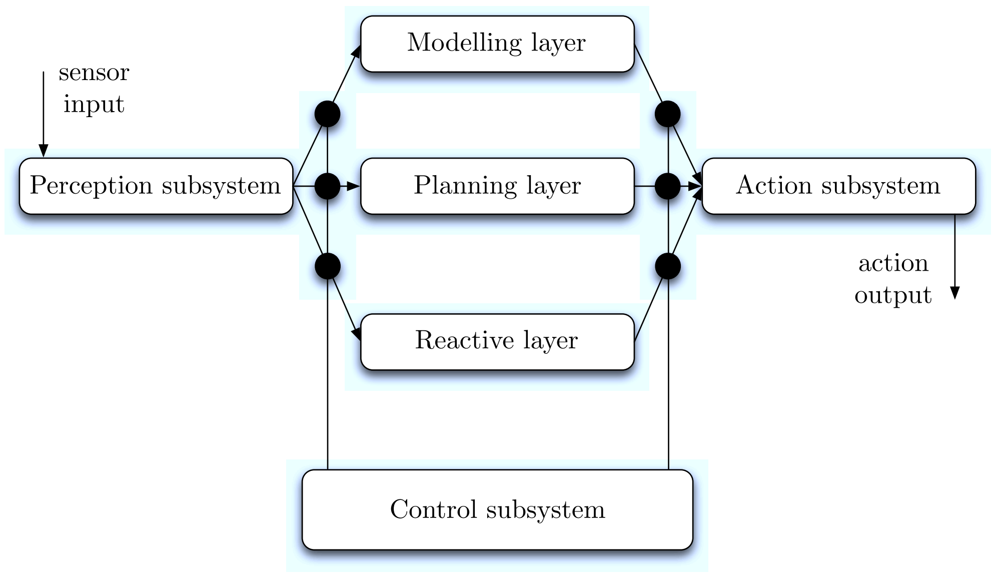

Figure 5.8: Hybrid agent horizontal layered TouringMachines architecture. Adapted from Wooldridge (2009)

The TouringMachines architecture was first implemented to provide a rule set for autonomous vehicles driving between locations where there were other vehicles. The *Reactive layer* is implemented as a set of state action pairs that simply suggest an action based on the situation, such as collision avoidance. The *Planning layer* deals with attempting to satisfy the agent's goals from a set of reference schemas. Based on these schemas and the current situation, the planning layer will adopt a strategy that is most beneficial for fulfilling the current goal. The *Modelling layer* renders a representation of the other agents in the system in an attempt to predict conflicts and generates goals to try to avoid them. These new goals are then passed to the *Planning layer* which will provide a strategy, based on these goals. These three layers make up the *Control subsystem* which decides which layer should have control of the agent at the present moment.

5.11 Java Agent DEvelopment Framework (JADE)

JADE is a Java based platform for developing FIPA compliant agent based systems. It comprises a set of classes that allow the user to produce a multi-threaded agent system and a set of tools for testing the functionality of both individual agents and the system as a whole. It is distributed under an open source Library GNU Public Licence (LGPL) allowing for a large degree of collaboration both with projects using the platform and with the development of the platform itself. Unlike JAVA, where objects have *methods* to perform actions, JADE provides a set of *behaviours* that can be utilised and extended to allow the agent to act in the desired manner.

5.11.1 Behaviours

It is possible, with JADE, to run several behaviours concurrently. However, the scheduling, unlike with JAVA, is not preemptive, so it is up to the programmer to decide when a behaviour should switch and take precedence over another. This can be done by making a behaviour *active* or *blocked* using JAVA *methods* within JADE. The ability to do this makes programming in JADE more flexible for scheduling. However, it does become more challenging from a complexity point of view. A description of some of the more useful behaviours provided by JADE are given below.

One Shot Behaviour

A *One Shot Behaviour*, as the name might suggest, is an action that is performed only once. It can be used to set up parameters for agents when they are initialised or to form subroutines of a *Sequential Behaviour*, as will be discussed later in this section.

Cyclic Behaviour

A *Cyclic Behaviour* is one of the more commonly used behaviours, especially with regard to agent communication. For message receiving, a *Cyclic Behaviour* can be implemented to lie dormant in the background until a new message is received. Once it receives a message, the behaviour can become *active* to process the message information.

Ticker Behaviour

Another commonly used behaviour is the *Ticker Behaviour* which performs an action periodically as defined by a specified duration. This is useful for values that need to be updated periodically or for messages to other agents, informing them of the current state of the system.

Sequential Behaviour

A *Sequential Behaviour* is used when the type of action to be performed becomes more complex and a sequence of smaller actions is required. An example of when this might be useful would be if two agents need to cooperate to perform a task, but both are reliant on the other performing one stage of the task before they can start on theirs. However, in this example, safeguards would need to be implemented to avoid deadlock. It would be possible to produce a *Contract-Net Protocol* routine using a *Sequential Behaviour*. However, JADE provides two Java classes that can be extended to provide this functionality called the *ContractNetInitiator* class and the *ContractNetResponder* class.

Whilst there are more behaviours that JADE provides, they are essentially extensions of the ones detailed here and therefore will not be discussed here.

5.11.2 Agent Communication in JADE

One of the most useful elements of JADE is its handling of message passing between agents without the need for direct location knowledge or handling of complex string parsing. The agent's location is all handled by the AMS and the MTS. The addition of an appropriate AID to the `receiver` field of the message is all that is required from a programming viewpoint. The message content requires encoding and decoding in a string format and, while this could be done by systematically adding parameters to Java strings, JADE provides message content tools that does this automatically, provided that there is a fully formed set of ontologies that JADE can encode into and parse from strings. This feature allows for any Java objects to be passed from one agent to another whilst still maintaining the principles of user readable messaging.

5.11.3 Debugging Tools

JADE provides a number of debugging tools to allow the user to check the operation of their multi-agent system. All of the debugging tools are implemented as agents within the AP so they are not global inspectors, but cohabitants of the system. This section details some of the more useful debugging agents.

The Sniffer Agent

A particularly useful debugging tool provided by the JADE framework is the sniffer agent which allows the system designer to view all agent communication within the AP, including communication flow and message content. Since messages are grounded in speech theory, as discussed in Section 5.9, the sniffer agent acts as an invaluable tool to ensure the correct operation of the engineered system.

The Introspector Agent

Whilst the sniffer agent deals with agent communication within the system, the introspector agent provides details of the functionality of individual agents and what behaviours are active throughout their lifecycle. It provides useful insights into the incoming messages in an agent's message queue and what behaviours are activated to process the information.

The Dummy Agent

For a more simple analysis of an agent's responses, the dummy agent provides the functionality for a user-defined stimulus in the form of an ACL-Message to be sent to the agent under test. This allows for its responses to be analysed for unexpected behaviour.

5.12 Multi-Agent Systems of Note

As previously stated, the ambiguity of the term multi-agent system makes it fruitless and practically impossible to provide a comprehensive review of research into the area. This section aims to provide a review of some multi-agent systems that relate to the work carried out in this research project.

5.12.1 Multi-Agent Reinforcement Learning (MARL)

Since many reinforcement learning tasks do not require sequential computation, a distributed methodology can be applied. A multi-agent approach is appealing, since agents can run in parallel, computing various aspects of the system. This gives the potential for adding robustness to the task. Failure of an agent to perform a task can be overcome if another agent takes up the mantle.

Whilst there are potential advantages in applying multi-agent techniques to RL problems, there are also numerous challenges that must be addressed. In standard RL, there is often a clear goal to be achieved, but with MARL the problem is usually distributed, so assigning goals to individual agents can be difficult.

When multiple agents are evaluating an environment, there is potential for an agent's learning to be affected by the learning of others, making the learning problem non-stationary. Similarly, if the learning problem is not specifically defined, agents can begin to build up information about the other agents as well as the environment they are supposed to be investigating.

Another issue to be addressed is whether to adopt either a selfish scheme or a cooperative scheme for learning and whether or not agents must keep track of the learning of others.

All of these issues must be considered. However, there is no 'one size fits all' solution, so the considerations must be made on a case by case basis.

The Goal of MARL

Defining a clear goal for MARL is difficult, as discussed by Busoniu et al. (2008), because a lot depends on the scope of the problem being investigated. Broadly speaking, however, he defines the two main goals of MARL as *stability* and *adaption* where *stability* is convergence to a stationary policy and *adaption* deals with maintaining performance as other agents change their policies.

MARL for Shared Information

The ability of agents to share learnt information can be an interesting prospect as, in many situations, agents have a limited sphere of influence. Tan (1993) investigates this for a predator prey environment which was used as a testbed for MARL with and without information sharing. The three cases under investigation were sharing of sensory information, sharing of learnt policies and sharing of sensory information for joint tasks.

His results show a slight improvement of convergence in the first case and marked improvements in the second and third cases. Despite the research being grounded in a simulation where parameter choices inherently affect the learning results, a strong case is made for information sharing to aid convergence to a solution.

Price & Boutilier (2003) introduce an approach called *implicit imitation* where the actions of a mentor agent inform a strategy for an imitator agent. The advantages of such a scheme would be most beneficial in systems where direct communication between agents is either difficult or impossible. The scheme is tested on a number of simple MDPs and shows improvements in convergence times over standard reinforcement learning techniques.

Whilst these examples give no explicit methodology or framework for information sharing in all situations, the techniques are interesting and should be considered at the design time of a MAS.

MARL for Distributed Control Problems

Very little work has been done in the field of MARL for distributed control. Of these, most examples of MARL are focussing on static games and small grid worlds. Busoniu et al. (2008) notes that:

> Most MARL algorithms are applied to small problems only, like static games and small grid worlds. As a consequence, these algorithms are unlikely to scale up to real-life multiagent problems, where the state and action spaces are large or even continuous. Few of them are able to deal with incomplete, uncertain observations. This situation can be explained by noting that scalability and uncertainty are also open problems in single-agent RL.

Despite this, however, there are a few examples of MARL being used for distributed control. Gross et al. (2000) use a multi-agent neural function approximator approach to control an industrial hard-coal combustion process in a power plant. The choice to use a neural function approximator was made due to the continuous state and action spaces in the system and the, then, prohibitive memory cost to store state action pairs in memory for standard techniques such as Q-learning.

The control system comprises four agents, each with access to information from the six burners of the combustion system. This is realised in the form of a camera system that observes the colour, shape and size of the flame. From the images, information can be determined about the temperature, coal distribution and the makeup of the emissions. The control inputs allow alteration of the distribution of air between burners, the distribution between primary and secondary air (where secondary air is recycled air from a previous burn) and the overall air amount. An emphasis is put on agent scheduling so that agents can be sure that the outputs are a direct result of their change in inputs and not the effect of another agent's control strategy.

The results of their system indicate that use of their multi-agent control scheme does give improvements over the standard control scheme adopted in the plant. Similar results can be achieved by consuming less air. However, the results do not appear to be significant and, since the learning is scheduled, the use of a multi-agent approach seems superfluous.

Wiering (2000) demonstrates a different kind of MARL to increase efficiency in a traffic light control problem where the traffic lights and the cars are modelled as agents. In the simulation each car has three parameters, the traffic light they are at, their place in the queue at the specified traffic light and their destination. There are 48 traffic lights ($\mathbf{tl} \in [1..48]$), 20 positions in a queue ($\mathbf{place} \in [1..20]$) and 10 destination addresses ($\mathbf{des} \in [1..10]$). This huge parameter set is the reason that a multi-agent approach was adopted, since the number of potential system states makes it intractable as a global RL problem. It is assumed that the cars can pass information about these three states to the traffic light controller so that it can make locally optimal decisions about an appropriate action to take.

The decisions of the traffic light controllers were tested using a range of techniques including random selection, fixed rate decisions, largest queue first, highest intersection throughput and three reinforcement learning based controllers. The fitness metric was the average waiting time of a car within the system. The RL algorithms used are not detailed here as they are standard RL optimisation algorithms and the substance of this work is in the way in which it is distributed. The results of the tests showed that the RL controllers slightly outperformed the other systems for low traffic loads and had marked improvements over the other controllers for high traffic volumes. Whilst the results are promising and show once again that RL can provide improvements to the decisions made in a distributed system, this is, in essence, an optimisation problem. The fact that a random function selector works at all is a testament to this fact.

5.12.2 Multi-Agent Control (MAC) Architectures

As with the term *agent*, the word *control* has many meanings, especially when it comes to AI systems. There is little research into control, as defined in Chapter 2, using multi-agent systems – especially when investigating non-linear mechanical plants. However, in addition to the few examples where control is examined (examples include: Stothert & Macleod (1997), MacLeod & Stothert (1998), Voos & Litz (2000), Veres & Luo (2004)), there is research into similar fields such as resource allocation for controlling the operation of large scale structures, many of which have parallels with the work presented in this thesis. This section details some of the more interesting examples.

For control of a dynamic distributed computer control system, Stothert & Macleod (1997) present an approach that focusses on dynamically assigning agents from templates to a system when required to produce a distributed PID controller. The controller is made up of dynamically assigned **Proportional**, **Integrator** and **Differentiator** agents capable of acting collectively as a stable controller. The structure of the controller is given at Figure 5.9.

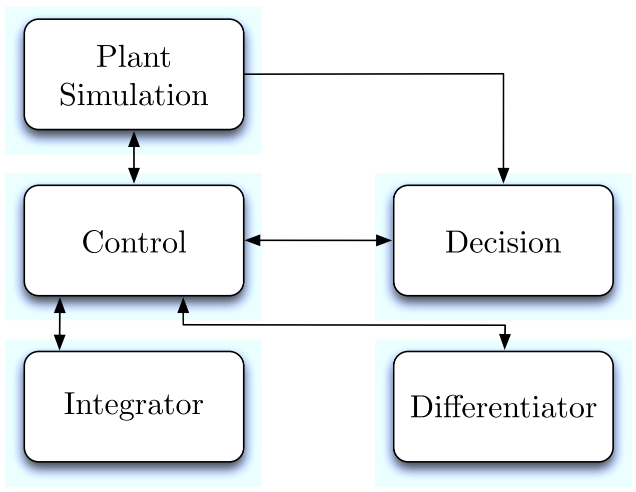

Figure 5.9: A dynamically assigned distributed PID controller. Adapted from Stothert & Macleod (1997)

The effect of this dynamic assignment was that, when an agent failed or was removed, the operation of the plant suffered temporarily but as soon as a new agent was assigned, functionality was restored. This system did require *a priori* knowledge of the plant but demonstrated a certain amount of adaptability as the requirements were altered.

Continuing from this work, full distribution of a controller was discussed by MacLeod & Stothert (1998) in an attempt to produce a controller for a laboratory scale mine refrigeration plant. The plant is proposed due to the high temperatures that are found in deep mines ($> 3km$) where the rock temperature typically reaches $50°C$ so refrigerated water is used to cool the ambient temperature (van der Walt & Whillier (1978), Bailey-McEwan (1991)). As in Stothert & Macleod (1997), agent templates are used to define the abilities of an agent, such as the "cooperation mechanisms that the agent is capable of using." All of the sensor information and control inputs are controlled through a supervisory control and data acquisition (SCADA) unit. The structure of their proposed controller is given at Figure 5.10 where the SCADA unit is thought of as an interface agent to the system.

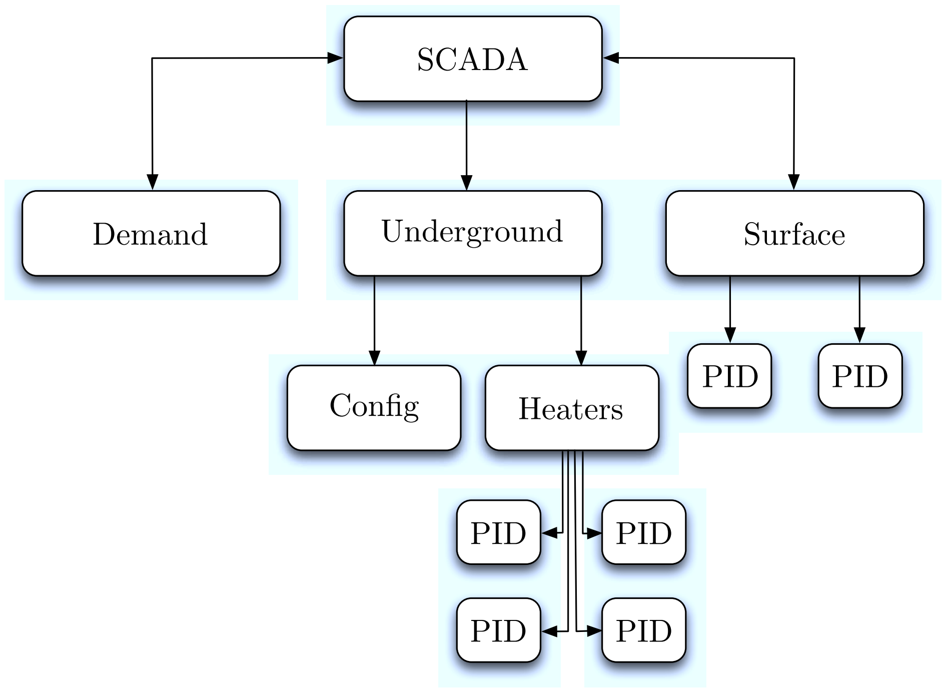

Figure 5.10: Agents for refrigeration plant controller using SCADA. Adapted from MacLeod & Stothert (1998)

The Demand, Underground and Surface agents are responsible for system planning which works using simple sets of rules in each agent. The system has a bottom up topology as the flow of water to the underground system only flows one way. As such, an agent will refer the problem to the agent above it if it cannot satisfy the demand locally. The configuration agent uses a fuzzy ruleset to produce its decision, with water flow and temperature as the two inputs. Once a strategy has been decided upon, the system launches PID agents to control the heaters using a standard PID structure, as described in Section 2.6 of Chapter 2.

Under testing, the system does indeed provide appropriate control strategies for the plant described. However, there is a single point of failure in the form of the SCADA unit so, whilst the actions are distributed, the data acquisition and control inputs are not. The mine refrigeration plant provides an interesting testbed for the system, but it is relatively slow moving and it is difficult to imagine such a system being adaptable to a fast, non-linear dynamic plant.

In an attempt to produce a market-based controller for dynamic systems Voos & Litz (2000) (see also Voos (1999*a*) and Voos (1999*b*)) produce an algorithm to assign utility to a water tank distribution problem. The task involves a series of water tanks with associated agents that each attempts to fill its water tank to the desired level by negotiation with the other tank agents. An agent is either a *producer* or a *consumer*, depending upon whether they have a surplus or deficit of water. Depending on the total water in the system, an equilibrium price is reached through contract negotiation. Despite the seemingly multi-agent nature of this system, all parameters are known in advance so all of the optimisation could be achieved globally using standard optimisation algorithms so a multi-agent approach is not essential or even necessary.

Veres & Luo (2004) introduces a much more complex control architecture in the form of a *cautiously optimistic* controller agent (COCA). The structure of the COCA architecture is given at Figure 5.11.

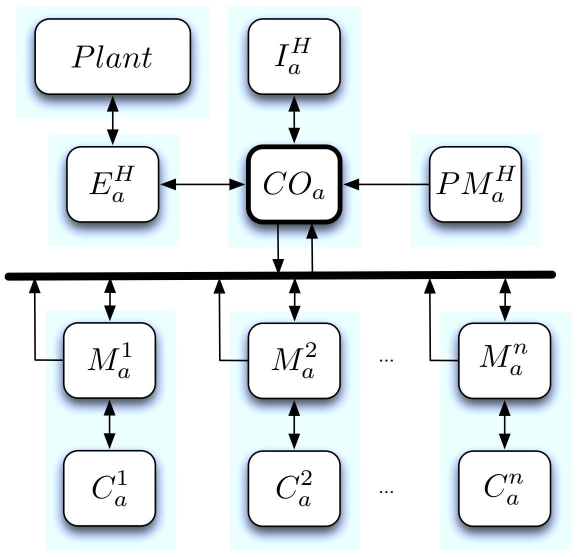

Figure 5.11: The COCA architecture of autonomous agents. Adapted from Veres & Luo (2004)

Each block is described as an instance of an agent type given below.

CO_a Main supervisor of a cautiously optimistic (CO) controller agent

PM_a^H Physical Modeller agent

M_a^k Modeller for control agent for model structure indexed by k : optimistic robust controller search. This is a nonlinear optimization problem in the model parameter space.

$C_a^k(\nu)$ Computational procedure agent of a controller of type ν

E_a^H Experimenter agent that implements, initializes and applies a controller to the *Plant* and records i/o measurements

I_a^H Interface agent for human operators, normally residing on a terminal of some kind (PC, hand-held touchpads, PDAs etc.)

Where all agents annotated with an ^{H} superscript have a GUI associated with them. The architecture relies on a set of control law modeller agents which suggest appropriate control strategies given the control problem to the CO_a agent. The CO_a agent then evaluates all suggestions and assigns them based on the associated cost. It then interacts with the experimenter agent to implement the solution.

This scheme is interesting because it distributes the control problem to allow for multiple agents to explore the possibilities of different control strategies simultaneously. However, as with the example given by MacLeod & Stothert (1998), it still relies on a centralised coordinator agent which is a single point of failure for the entire system.

Whilst some of these schemes present interesting methodologies and techniques for implementing a MAC system, all have their weaknesses. Predominantly, a recurring theme appears to be a single point of failure within the structure of the system. When discussing decentralised control De Wolf & Holvoet (2003) makes the following statement:

> Decentralised control systems consist of controllers that are designed and operated with limited knowledge of the complete system. There is no central decision maker and the information flow stays local. Decentralised control is actually a self-organising emergent property of the system.

According to this definition of decentralised control, the above systems do not meet the criteria because of this recurrence of single failure points. One reason for this might be that it is difficult, as a designer of a control system, to completely remove the plant that is to be controlled from the controller design process. This derives from an understanding of the plant that is essential for standard control techniques, but can be highly detrimental for distributed control design.

5.13 Summary

Whilst there is a lot of interesting work being produced in the field of distributed artificial intelligence, there is nothing that addresses the notion of a fully distributed control scheme that displays redundancy. Some work deals with self-organisation but the goals they attempt to achieve are too high level to be considered for a controller. Some work looks at the possibility of distributing reinforcement learning problems, but this work can be split into two categories: parallelizing standard reinforcement learning to speed up convergence; optimisation problems, neither of which are easily adaptable to control engineering problems. The few examples of multi-agent control that are presented are not robust as they all either have single points of failure or multiple points of failure.

Chapters 6 and 7 detail the development of two multi-agent controllers that attempt to ameliorate some of the difficulties presented in previous work.

5.14 Chapter Bibliography

Antsaklis, P. J. (1994), Defining intelligent control, *in* 'Report of the Task Force on Intelligent Control, P.J Antsaklis, Chair', John Wiley & Sons, Inc.

Avižienis, A. (1975), Fault-tolerance and fault-intolerance, *in* 'Proceedings of the International Conference on Reliable Software', ACM.

Bailey-McEwan, M. (1991), 'Use of the chiller computer program with conventional water chilling installations on south african gold mines', *Journal of the Mine Ventilation Society of South Africa* **44**(1), 2–21.

Bellifemine, F. L., Caire, G. & Greenwood, D. (2007), *Developing Multi-agent Systems with JADE (Wiley Series in Agent Technology)*, 1st edn, Wiley-Blackwell.

Bond, A. H. & Gasser, L. (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kauffman Publishers.

Bratman, M. E. (1987), *Intention, Plans, and Practical Reasoning*, Harvard University Press.

Bratman, M. E. (1990), What is intention?, *in* 'Intentions In Communication', MIT Press, pp. 15–32.

Bratman, M. E., Israel, D. J. & Pollack, M. E. (1988), 'Plans and resource-bounded practical reasoning', *Computational Intelligence* **4**(3), 349–355.

Brooks, R. A. (1986), 'A robust layered control system for a mobile robot', *IEEE Journal of Robotics and Automation* **2**(1), 14–23.

Brooks, R. A. (1991), 'Intelligence Without Representation', *Artificial Intelligence* **47**(1-3), 139–159.

Busoniu, L., Babuska, R. & De Schutter, B. (2008), 'A comprehensive survey of multiagent reinforcement learning', *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews* **38**(2), 156–172.

Colorni, A., Dorigo, M. & Maniezzo, V. (1991), Distributed optimization by ant colonies, *in* 'European Conference on Artificial Life', Elsevier.

De Wolf, T. & Holvoet, T. (2003), Towards autonomic computing: Agent-based modelling, dynamical systems analysis, and decentralised control, *in* 'IEEE International Conference on Industrial Informatics', IEEE.

Dorigo, M. & Gambardella, L. M. (1997), 'Ant colony system: A cooperative learning approach to the traveling salesman problem', *IEEE Transactions on Evolutionary Computation* **1**(1), 53–66.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996), 'The Ant System: Optimization by a Colony of Cooperating Agents', *IEEE Transactions on Systems Man and Cybernetics Part B Cybernetics* **26**(1), 29–41.

Eberhart, R. & Shi, Y. (2001), Particle swarm optimization: developments, applications and resources, *in* 'Proceedings of the 2001 Congress on Evolutionary Computation', IEEE.

Facebook (2004), 'Facebook'.
URL: *http://www.facebook.com*

Ferguson, I. A. (1992), TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents, PhD thesis, Clare Hall, University of Cambridge, UK.

FIPA (2002*a*), 'FIPA ACL Message Structure Specification'.
URL: *http://www.fipa.org/specs/fipa00070/*

FIPA (2002*b*), 'FIPA Agent Management Specification'.
URL: *http://www.fipa.org/specs/fipa00023/*

FIPA (2002*c*), 'FIPA Agent Message Transport Service Specification'.
URL: *http://www.fipa.org/specs/fipa00084/*

FIPA (2002*d*), 'FIPA Contract Net Interaction Protocol Specification'.
URL: *http://www.fipa.org/specs/fipa00029/*

Friedman, M. (2006), Afterward, *in* 'I, Pencil: My Family Tree as told to Leonard E. Read', Foundation for Economic Education, p. 18.

Gaing, Z.-L. (2004), 'A Particle Swarm Optimization Approach for Optimum Design of PID Controller in AVR System', *IEEE Transactions on Energy Conversion* **19**(2), 384–391.

Gribble, S. (2001), Robustness in Complex Systems, *in* 'Proceedings Eighth Workshop on Hot Topics in Operating Systems', IEEE Computer Society.

Gross, S. D., Stephan, V., Debes, K., Gross, H. m., Wintrich, F. & Wintrich, H. (2000), A Reinforcement Learning based Neural Multi-AgentSystem for Control of a Combustion Process, *in* 'IEEE-INNS-ENNS International Joint Conference of Neural Networks', IEEE Computer Society, pp. 217–222.

Hu, X. & Eberhart, R. (2002), Multiobjective optimization using dynamic neighborhood particle swarm optimization, *in* 'Proceedings of the 2002 Congress on Evolutionary Computation CEC02', IEEE Computer Society, pp. 1677–1681.

Iglesias, C. A., Garijo, M. & Centeno-González, J. (1998), A Survey of Agent-Oriented Methodologies, *in* 'Proceedings of the 5th International Workshop on Intelligent Agents', Springer-Verlag, pp. 317–330.

Jennings, N. R. (2000), 'On Agent-Based Software Engineering', *Artificial Intelligence* **117**(2), 277–296.

Kennedy, J. & Eberhart, R. (1995), Particle Swarm Optimization, *in* 'IEEE International Conference on Neural Networks', IEEE, pp. 1942–1948.

MacLeod, I. & Stothert, A. (1998), 'Distributed Intelligent Control for a Mine Refrigeration System', *IEEE Control Systems Magazine* **18**(2), 31–38.

McCarthy, J. (1979), Ascribing mental qualities to machines, *in* 'In Philosophical Perspectives in Artificial Intelligence', Humanities Press.

Mendham, P. D. (2006), Multi-Agent Control for the Skylon Spaceplane, PhD thesis, University of York.

Meyer, B. (1997), *Object-Oriented Software Construction*, 2nd edn, Prentice Hall.

MIT Center for Collective Intelligence (2006), 'MIT Center for Collective Intelligence'.
URL: *http://cci.mit.edu/index.html*

Müller, J. P., Pischel, M. & Thiel, M. (1995), Modelling Reactive Behaviour in Vertically Layered Agent Architectures, *in* 'Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents', Springer-Verlag, pp. 261–276.

Parrott, D. (2006), 'Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model using Speciation', *IEEE Transactions on Evolutionary Computation* **10**(4), 440–458.

Pay, M. (2008), Kill the Wildebeest, Master's thesis, University of York.

Poli, R. (2008), 'Analysis of the Publications on the Applications of Particle Swarm Optimisation', *Journal of Artificial Evolution and Applications* **2008**(2), 1–11.

Price, B. & Boutilier, C. (2003), 'Accelerating Reinforcement Learning through Implicit Imitation', *Journal of Artificial Intelligence Research* **19**(1), 569–629.

Rao, A. S. & Georgeff, M. P. (1995), BDI Agents: From Theory to Practice, *in* 'Proceedings of the first international conference on multiagent systems ICMAS95', pp. 312–319.

Searle, J. (1969), *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press.

Shoham, Y. (1993), 'Agent-oriented programming', *Artificial Intelligence* **60**(1), 51–92.

Smith, A. (1776), *An Inquiry into the Nature and Causes of the Wealth of Nations*, Liberty Fund.

Steels, L. (1990), Cooperation between distributed agents through self-organisation, *in* 'IEEE International Workshop on Intelligent Robots and Systems', pp. 8–14.

Stothert, A. & Macleod, I. M. (1997), 'Using intelligent agent templates for dynamic structuring of distributed computer control systems', *Engineering Applications of Artificial Intelligence* **10**(4), 335–343.

Tan, M. (1993), Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, *in* 'Proceedings of the tenth international conference on machine learning', pp. 330–337.

Twitter (2006), 'Twitter'.
URL: *http://www.twitter.com*

van der Walt, J. & Whillier, A. (1978), 'Considerations in the Design of Integrated Systems for Distributing Refrigeration in Deep Mines', *Journal of the Mine Ventilation Society of South Africa* **31**(12), 217–243.

Veres, S. M. & Luo, J. (2004), A Class of BDI Agent Architectures for Autonomous Control, *in* '43rd IEEE Conference on Decision and Control', pp. 4746–4751 Vol.5.

Voos, H. (1999*a*), Market-based Algorithms for Optimal Decentralized Control of Complex Dynamic Systems, *in* 'Proceedings of the 38th IEEE Conference on Decision and Control', pp. 3295–3296.

Voos, H. (1999*b*), Market-based control of complex dynamic systems, *in* 'International Symposium on Intelligent Control Intelligent Systems and Semiotics', pp. 284–289.

Voos, H. & Litz, L. (2000), Market-based optimal control: a general introduction, *in* 'Proceedings of the 2000 American Control Conference', pp. 3398–3402 vol.5.

Wiering, M. (2000), Multi-Agent Reinforcement Learning for Traffic Light Control, *in* 'Proceedings of the 25th International Conference on Machine learning', pp. 1151–1158.

Wikipedia Foundation (2001), 'Wikipedia'.
URL: *http://www.wikipedia.org*

Wooldridge, M. (2009), *An Introduction to MultiAgent Systems*, 2nd edn, John Wiley & Sons.

Chapter 6

Towards a Multi-Agent PID Controller

Contents

An important stepping stone in the process of developing an agent-based, self-organising, distributed control system is combining some of the necessary technologies to demonstrate their viability. As such, a multi-agent PID controller was developed to test the capabilities of JADE in a realtime situation using a TRMS simulation as plant. It also allowed familiarity with the coding practices of JADE to be gained so that implementation difficulties could be overcome.

6.1 Simulation Setup

Before the structure of the multi-agent system is examined, there are several aspects of the TRMS simulation that need to be discussed. This section looks at some of the more practical aspects of implementing a multi-agent system using JADE.

6.1.1 Using a Realtime Kernel

Early on in the development stage it became clear that, although JADE offers some useful tools for the development of multi-agent systems, it is not necessarily designed for speed of interaction. This is a potential pitfall as the data rate of the control signals for such a dynamic system would need to be fast enough to handle the quickest system dynamics, otherwise control would be ineffective. However, it became obvious quite quickly that the limiting factor was not, as first expected, the capabilities of JADE but rather the update interval of the operating system clock, which was erratic and slow. For this reason, simulations were carried out on a computer with an Ubuntu (Canonical Ltd., 2009) operating system with a realtime kernel. This yielded a much faster update rate of the system clock and hence the ability to produce more timely control signals.

6.1.2 TRMS Visualisation

During the initial design phases of the system, it became clear that there would be difficulties in producing a controller if the system could not be effectively monitored.

LiveGraph

LiveGraph is a free and open source Java library that allows for realtime plotting of parameters via a text file which is, in turn, read by the library. Several GUIs enable the user to alter the manner and speed in which the data is plotted. This allowed for all of the system states to be visualised to ensure the correct operation of the system.

3D Model of the TRMS

Despite the use of LiveGraph for *data* visualisation, the plots did not communicate the *dynamics* as well first envisaged. For this reason a 3D model of the TRMS was developed, using Java 3D$^{\text{TM}}$(Oracle Corporation, 2011), so that the control of the system could be monitored in real time. Java 3D$^{\text{TM}}$is an open source extension to Java, licensed under the open source GNU General Public License (GPL), version 2, with the CLASSPATH exception allowing for full use of its libraries without cost. It enables the user to create simple 3D shapes using single lines of code and provides simple functions to translate or rotate them. Creating complex landscapes would be extremely difficult and time consuming using this package. However, for the TRMS, which can be depicted using only a handful of shapes, it is ideal. The 3D model is shown in Figure 6.1.

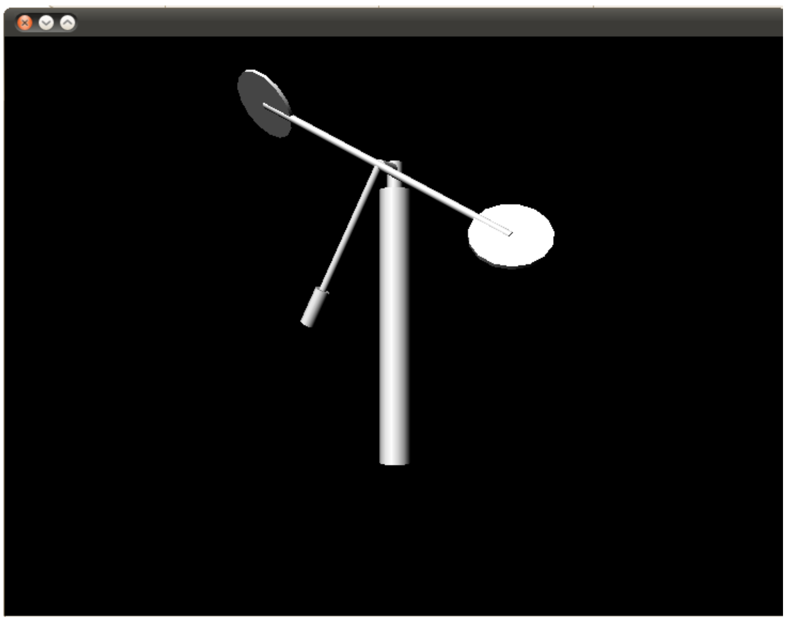

Figure 6.1: Java 3D model of the TRMS System

6.2 The PID framework

In the development of the PID framework, the decision was made to distribute the system as much as possible to test the speed of communication between agents and determine if it would be fast enough to handle the required controller activity. The final PID structure is depicted in Figure 6.2.

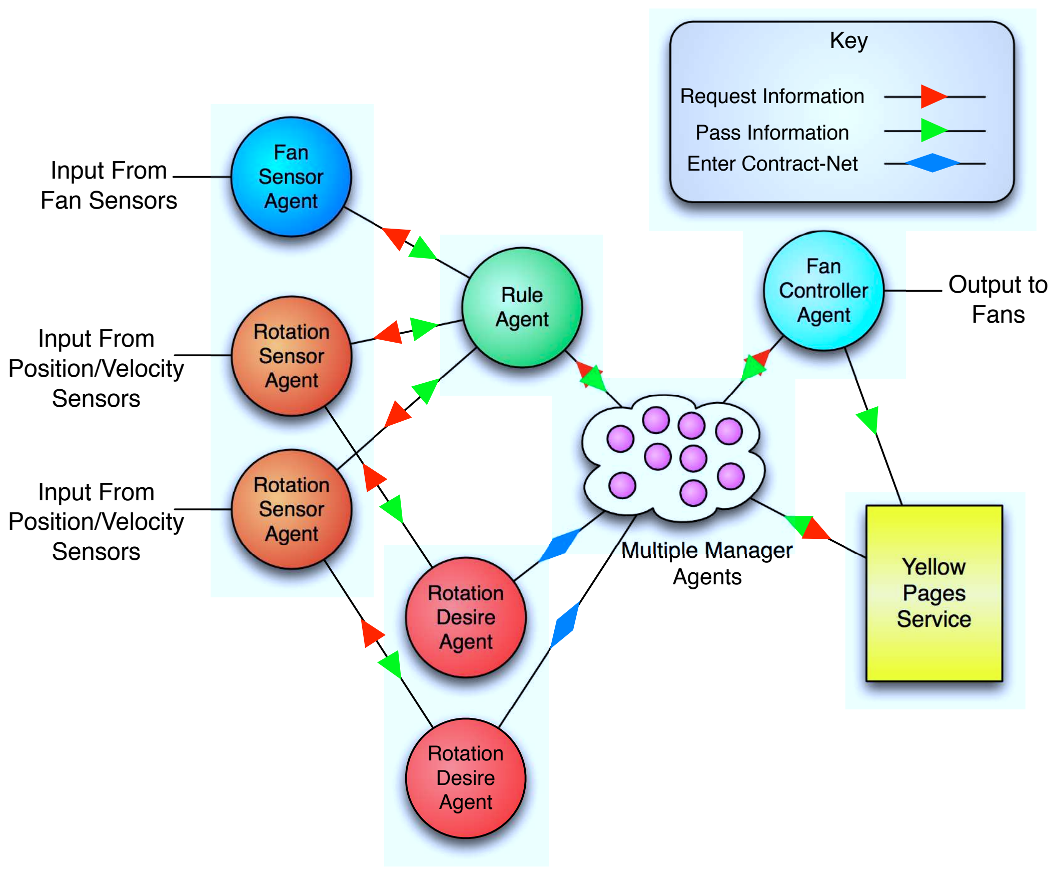

Figure 6.2: The structure of a multi-agent PID controller

To understand the workings of the structure, the capabilities of the agents are first elaborated.

6.2.1 Fan Sensor Agent

The fan sensor agent is one of the few agents that interacts with the TRMS in any way. It periodically receives information about the propeller speed from the plant, which allows it to relay that information to any agent that might require it.

6.2.2 Rotation Sensor Agent

Similar to the fan sensor agent, the rotation sensor agents periodically receive information about how the system is moving in a given plane. This gives it constantly-updated knowledge of the position, velocity and acceleration in a given plane. Due to the two rotational frames in the TRMS, two rotation sensor agents are required.

6.2.3 Rotation Desire Agent

The rotation desire agent holds values for the desired TRMS position in a given plane. It also has the capability to calculate an error term, based on the current state of the system in that plane. The existence in the system of two operational planes necessitates two rotation desire agents.

6.2.4 Rule Agent

The rule agent is much more adaptable in its design and implementation than the other agents. It has the capacity for being a complex machine learning agent. It might learn about how the system reacts to various inputs or simply employ a lookup table for matching outputs to desired inputs. Its purpose is to propose actions to reduce the error in the given rotational plane.

6.2.5 Manager Agents

The manager agents are essentially a group of middlemen that organise the process of reducing errors. They do not perform any computational operations on either the error terms or the fan inputs. However, they do have information about the agents that can affect the plant. They coordinate with multiple agents to get an action performed.

6.2.6 Fan Controller Agent

The fan controller agent can change the fan speed. It is a purely reactive agent, responding to fan control requests. These agents execute a procedure similar to defuzzification. Their objective is to ensure that requests do not simply get stuck in a cycle of contradictory control inputs that exceed the physical response limits of the fans.

6.3 Agent Interaction

For the purposes of understanding how the interaction between the agents leads to the control structure, it can be assumed that all of the sensor agents are frequently updating their information and that they have accurate data about the system state. Given this assumption, the computational flow can be split up into several, well-defined sections as depicted at Figure 6.3.

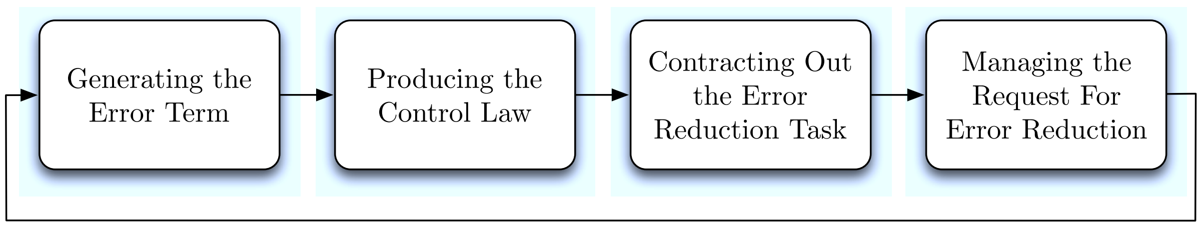
Generating the Error Term
Producing the Control Law
Contracting Out the Error Reduction Task
Managing the Request For Error Reduction

Figure 6.3: Agent interaction flow

6.3.1 Generating the Error Term

The starting point for this agent-based scheme is the generation of the error terms. Each rotation desire agent processes a set-point value, representing the desired values for the position and velocity of the TRMS in its associated rotational plane. Through appropriate communication with its associated rotation sensor agent, the desire agent calculates the difference between set-point and actual values for position and velocity in that plane, presenting them as error terms.

6.3.2 Producing the Control Law

The rule agent was initially developed to provide a set of PID control strategies for the system to enact. This was a natural starting point, as it has already been shown that a PID controller can be used to control the TRMS in Section 3.5.2. It was also felt that the rule agent had the potential to be an RL agent, constantly updating its ruleset to make its action suggestions increasingly accurate. However, an RL version of the rule agent was never developed due to underlying issues with the multi-agent PID structure. It was felt that since the scheme was structured so sequentially, with all agents vital to operation, the framework would never be able to provide the emergence that was sought. These issues are discussed further in Section 6.7.

6.3.3 Contracting Out the Error Reduction Task

With the error term generated, the rotation desire agent invokes the contract-net protocol with the manager agents, selecting one task based on cost. Why was this implemented? Firstly it allows for parallel error reduction. If one manager agent is busy performing an error reduction task, another can be selected. Secondly, it allows for potential implementation of competition between the manager agents later on in the development process.

6.3.4 Managing the Request for Error Reduction

Once a manager agent has been selected, there is a three-step process to bring the propellers to an appropriate speed to control the plant. Firstly, this agent sends the error term to the rule agent and asks for an appropriate propeller speed from its lookup table. Once it has received a nominated value, it utilises the yellow pages service to find the AID of the appropriate fan controller agent. Finally, it requests that the fan controller agent effect the proposed propeller speed on the plant. The manager agent receives a message from the fan controller agent indicating whether this process has been successful and relays it to the rotation desire agent. The whole process then repeats.

6.4 Input Combination

Since the control is distributed and there are two desire agents producing two contradictory control strategies, there has to be a method for input combination to ensure that both sets of desires are met. The simplest way to do this is to average the desired inputs from both desire agents. Using a buffer, the TRMS model determines how many desire agents are attempting to control the system. It then combines their total inputs and averages. In an early implementation, there was a direct link between a fan control request and the inputs to the system. This resulted in fast switching between fan control requests. However, the inertial properties of the fans made it impossible for these requests to have the desired effect. It takes up to two seconds for the fans to spin up to full speed. Also, for proper input combination, fan control requests would need to occur with constant time spacing, otherwise control action demands would have different effects depending on the interval. Therefore, the error combination finally used is analogous to the defuzzification process described in Section 2.7.3.

6.5 Tuning the PID Gain Values

The gain values of a PID controller need to be tuned if it is to operate effectively (Section 2.6). In the case of the multi-agent PID controller the values chosen were the same as the heuristically selected values shown in Table 3.3 of Chapter 3 so that a comparison between the two control structures could be made. These values are presented once again in Table 6.1.

	Theta	**Phi**
Proportional Gain K_p	400	100
Integral Gain K_i	700	50
Differential Gain K_d	500	200

Table 6.1: Gain values for the multi-agent PID controller

These values convert the error into PWM input values for the fans in the range of -500μs to 500μs. Since the TRMS accepts a range of 1000μs to 2000μs for its fan inputs, the constant value of 1500μs is added.

6.6 Results of the Multi-Agent PID Controller

The multi-agent PID controller is able to provide effective stabilisation and position control of the TRMS. A representative test is now described. Table 6.2 shows the desired positions in both θ and ϕ over the course of a 120 second test. Figures 6.4 and 6.5 show the time responses.

Time (s)	**Theta** (rad)	**Phi** (rad)
0	0.5	0.5
40	-0.7	$-\pi$
80	1.2	-0.2

Table 6.2: Desired θ and ϕ positions for the multi-agent PID controller

The set-point values chosen were chosen to lie within the system's feasible response envelope which was discussed in Chapter 3 (Figure 3.14). Attempting to drive the system to a value outside this envelope is impossible.

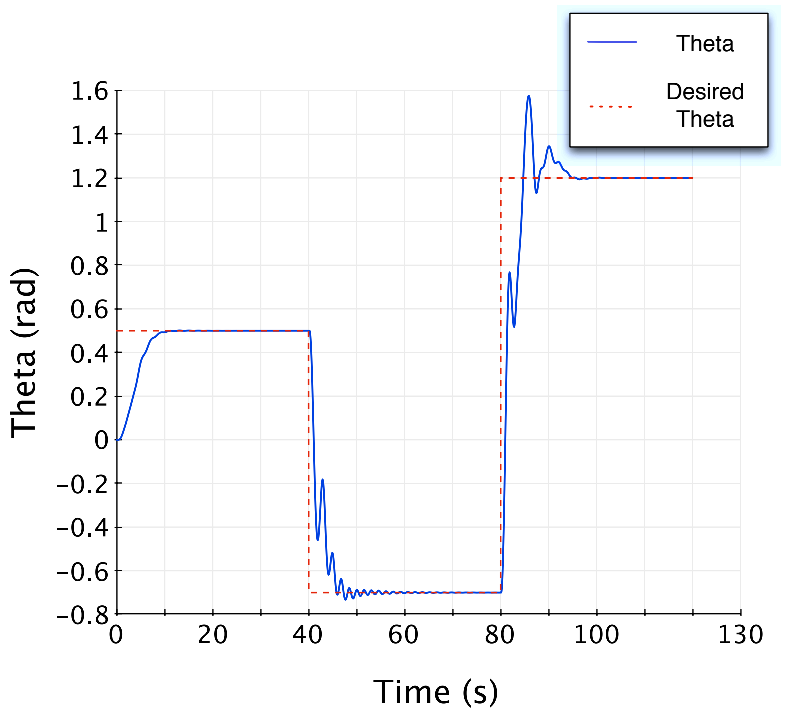

Figure 6.4: Results of the multi-agent PID controller with varying θ

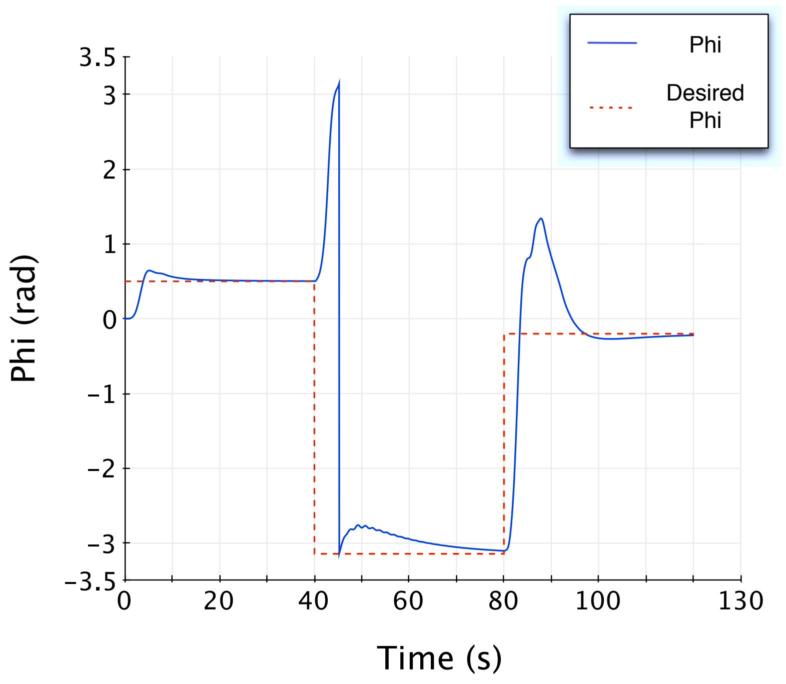

Figure 6.5: Results of the multi-agent PID controller with varying ϕ

Bearing in mind that control of this system is not trivial, being quite non-linear in character, the time responses are satisfactory. The high cross-coupling plus the asymmetry of the responses in θ and ϕ presents the multi-agent PID system with a considerable challenge. The reader is referred back to Chapter 3 where the TRMS dynamics are analysed. When these results are compared to those of the MATLAB simulated PID controller, shown again in Figures 6.6 and 6.7[1], the results are comparable, and when analysing the performance in the ϕ plane, arguably better when the multi-agent PID controller is employed. The speed of response is slower, but because of this, the detrimental effect of the θ response on the ϕ plane is reduced and the %OS is greatly improved.

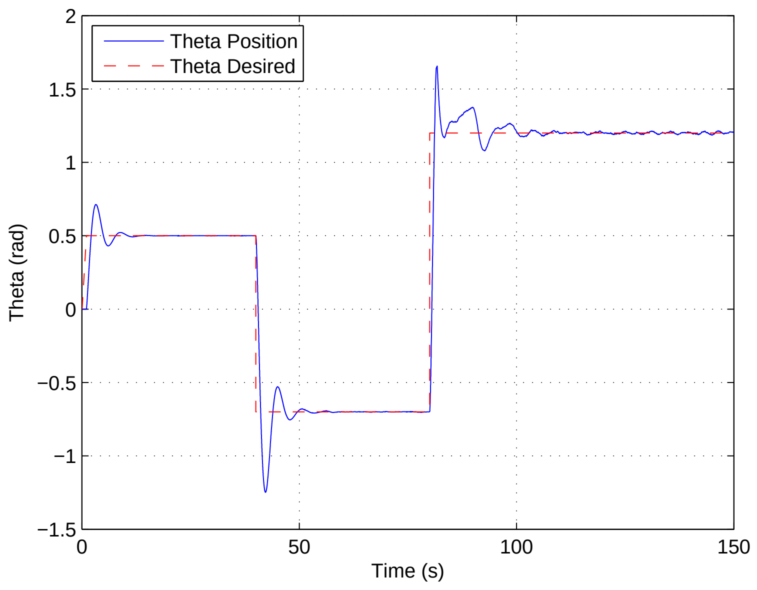

Figure 6.6: Results of the PID controller with varying θ

[1]The desired positions and timing of these two graphs are identical, but due to the increased speed of response with the MATLAB simulated PID controller, the discontinuities when a wraparound function is applied makes analysis difficult. For this reason, the absolute positions are shown for the ϕ plane in Figure 6.7 with the wrapped positions shown in Figure 6.5.

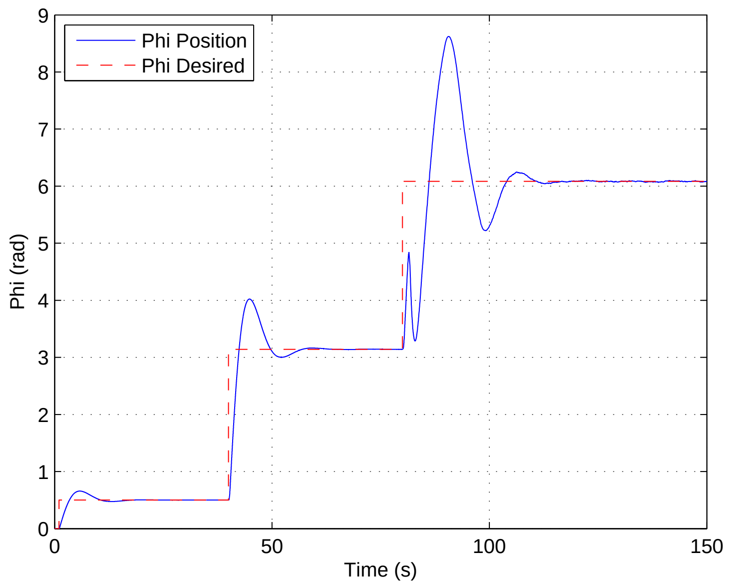

Figure 6.7: Results of the PID controller with varying ϕ

It should be noted when viewing these results that the nature of the realtime simulation has an impact on the performance of the controller; an impact that the MATLAB simulation does not produce. In the case of the MATLAB simulation, the effect of the computational load is bypassed and all calculations are performed before the integration routine performs its next step. This means that the results will not differ if the simulations were carried out on computers with different performance capabilities. In the case of the multi-agent PID framework there are two additional factors that will effect performance: all calculations must be done in a timely fashion as the plant simulation will not wait for controller calculations to be completed; and the system is not only having to calculate the control strategy, but also simulate the plant dynamics and produce multiple visualisations. If a controller cannot perform fast enough to control the fastest dynamics of the system plant, it can induce oscillations which will often cause a system failure.

Figure 6.8, shows that the *forces* generated by the fans are equal and opposite as required when keeping the TRMS stationary in both the θ and ϕ planes. When this is compared to Figure 6.9, the inputs are opposite, but not equal. This is due to the discrepancy in fan effectiveness between the forward and reverse directions.

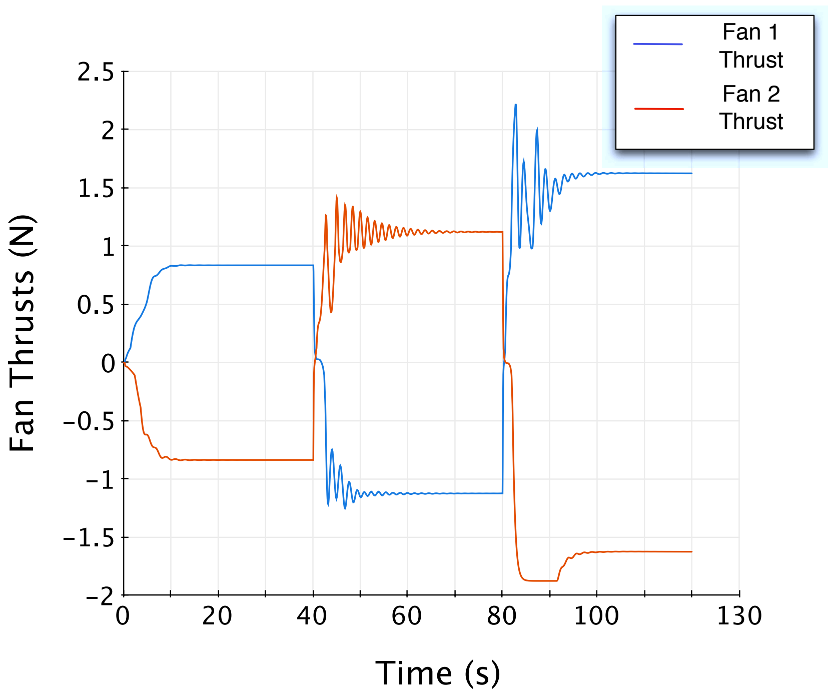

Figure 6.8: Force from both propellers for the TRMS with changing desired position

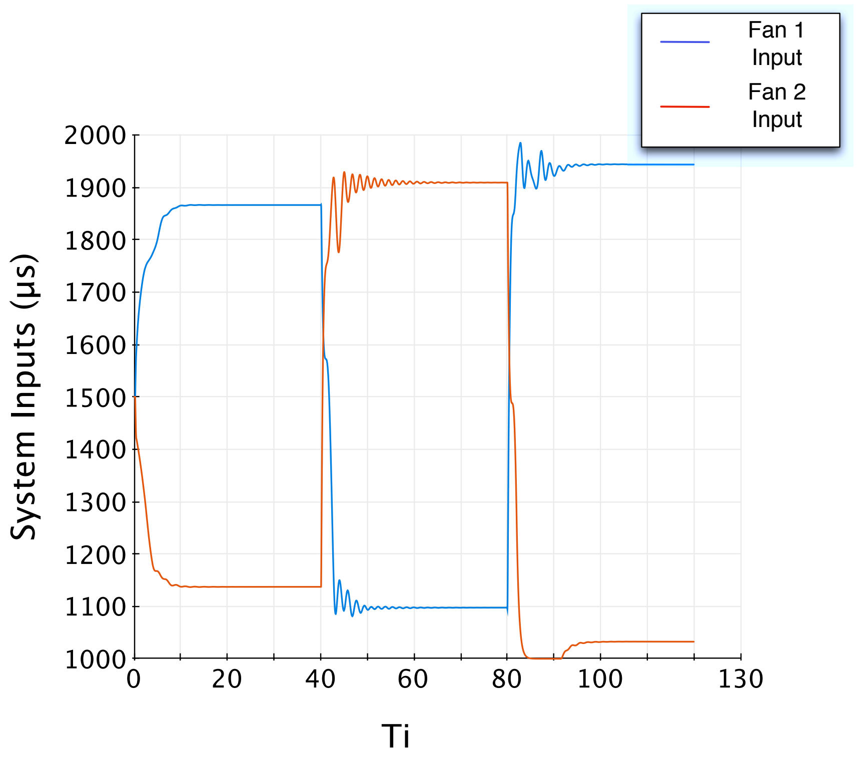

Figure 6.9: Inputs for both propellers of the TRMS with changing desired position

6.7 Framework Overview

When analysing this framework there are several issues to be considered. The main question this framework raises is, what immediate advantage does it have over a standard PID controller? In its current form, the answer is none. Whilst the *problem* has been distributed, apart from redundancy in the manager agents, every other agent is essential to the controller. It has multiple points of failure so in no way can in be thought of as a distributed framework for a future autonomous system.

Nevertheless, it does produce a valid control structure for the TRMS, and demonstrates that it is feasible to use a JADE based multi-agent system for a controller, which, given the required levels of interaction between agents, is substantive. Concerns over the communication and operation speed are manageable.

6.8 Conclusions

This chapter has illustrated the implementation of an agent-based PID controller which performs well given both its complexity, and the dynamic nature of the plant. It demonstrates that it is feasible to develop a more sophisticated multi-agent control system capable of learning and even self-organisation. Whilst there would certainly be merit in the continuation of this multi-agent PID approach, the plant specific nature of the controller does not serve as a good enough platform for further development of a plant independent multi-agent control scheme. As such, with the lessons learnt from this development, a new strategy is employed as detailed in the following chapter.

6.9 Chapter Bibliography

Canonical Ltd. (2009), *Ubuntu Version 10.04.3*, London, UK.

Oracle Corporation (2011), *Java 3D Version 1.5.2*, Redwood Shores, CA.

Chapter 7

A Distributed Learning Multi-Agent Framework

Contents

As previously discussed, it is the rigidity in the structure of the multi-agent PID controller is what makes it so susceptible to single points of failure. Therefore, a new strategy was employed to develop a distributed-learning, multi-agent controller for the TRMS. The guiding principle behind the design was that neither the agents nor the designer have access to information about the nature of the errors. Learning is based solely on the magnitude and sign of the errors presented. Whilst this might sound limiting from a control perspective, it grants huge freedom when trying to design for emergence; the standard engineering techniques must be dismissed. No longer can the engineer attempt to solve the control problem at hand; something that is so intuitive and hard not to do. Instead, attention must focus on the development of a suitable *learning* environment for agents.

Using this concept, the errors can be split up into error sets and assigned to **error agents** tasked with reducing their associated error. This notion is similar to that of fuzzy control except that with fuzzy control systems the sets are allocated with appropriate actions, based on the designer's knowledge of the system. Our idea also addresses the notion of collective intelligence; the agents only have limited spheres of influence.

7.1 Multi-Agent Controller Structure

To ensure that there is less susceptibility to single-point failures in the framework, the design steered away from imposed structures that determined the flow of information. Consequently, it is much less cluttered than the example described in Chapter 6. Only two types of agent populate the system: **Error Agents** and **Controller Agents**. Figure 7.1 shows how simple the organisation of the framework is, with no discernible hierarchy whatsoever.

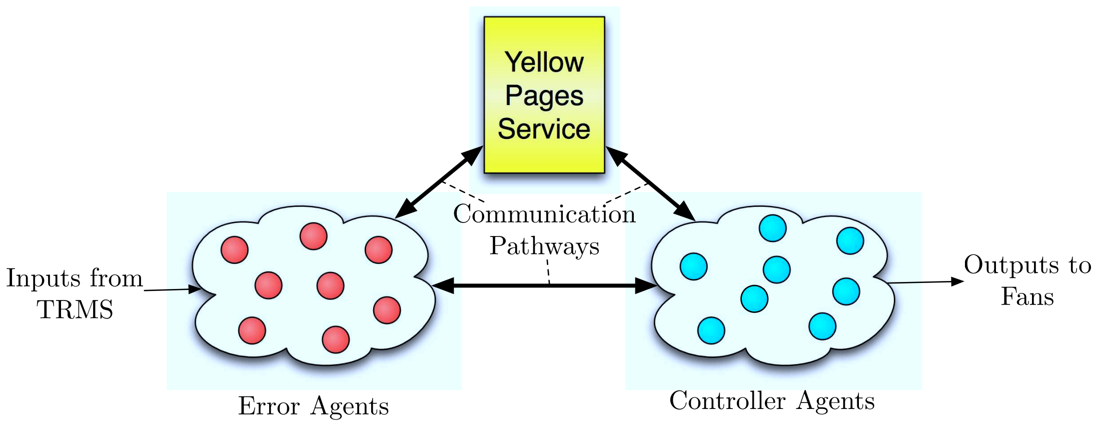

Figure 7.1: Framework structure for the multi-agent reinforcement learning controller

7.1.1 Flow of Interaction

The interaction flow between the agents is almost as simple as the structure and will be described here. Figure 7.2 depicts the **controller agents** publishing their services to the yellow pages service.

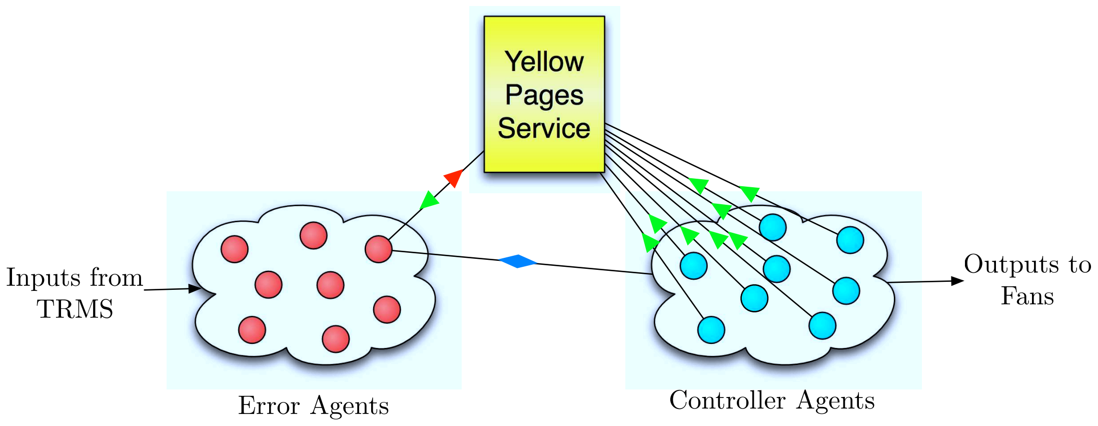

Figure 7.2: Controller agents publishing their services to the yellow pages service

Once all of the **controller agents** have published their services, an **error agent** searches for agents that can provide an *error reduction* service. Since all of the **controller agents** offer that service, the **error agent** invokes the contract-net protocol with all of the **controller agents** to see which one can offer *error reduction* at the best price. The details of both the **error agents** and the **controller agents** are discussed in Sections 7.1.2 and 7.1.3 respectively. The notion of how **controller agents** decide on an appropriate price is described in

Section 7.4.1.

Once a **controller agent** has been selected by the **error agent** to perform the *error reduction* task, the **controller agent** adjusts the fan speeds of the TRMS, as proposed by an RL algorithm, and informs the **error agent** that an action has been performed. Details of the use of RL algorithms by the **controller agent** are discussed in Section 7.1.3. After a specified waiting time, the **error agent** analyses the error to see if the action has been successful and informs the **controller agent** of the outcome. The **controller agent** updates its RL algorithm based on the **error agent** response. Using this method, all **controller agents** have the capacity to learn appropriate actions for all **error agents**, if *good* actions are discernable and they are given sufficient time and opportunity to converge.

7.1.2 Error Agents

An **error agent** receives information about the system and its desired position/velocity set-points from the simulation and generates an error term. This is very similar to standard negative feedback control. However, in this case, the **error agents** can combine errors in a variety of ways. The current framework has 8 **error agents**, each with a different error or a combination of errors. These are as follows:

- ThetaErrorAgent - generates an error signal based on the θ position.
- ThetaDotErrorAgent - generates an error signal based on the velocity in θ.
- ThetaIntegralErrorAgent - generates an error signal based on the integral of the θ error present on the system. Where there is an integral, there must be initial conditions, defined from a designated start time of operation.
- PhiErrorAgent - generates an error signal based on the ϕ position.
- PhiDotErrorAgent - generates an error signal based on the velocity in ϕ.
- PhiIntegralErrorAgent - generates an error signal based on the integral ϕ error present on the system. Where there is an integral, there must be initial conditions, defined from a designated start time of operation.
- ThetaCombinedErrorAgent - generates an error signal based on the combination of the θ position and the velocity in θ.

- PhiCombinedErrorAgent - generates an error signal based on the combination of the ϕ position and the velocity in ϕ.

These **error agents** were chosen because the **single error agents** are easy to implement and test. The **combined error agents** are a device for introducing some damping into the system.

Inbuilt Damping for Combined Error Agents

For the **error agents** that combine position and velocity errors, there is an implicit damping effect through proportional plus derivative action.

If the ThetaCombinedErrorAgent is used as an example, its error term is made up of the error in θ position and θ velocity. If the desired position is set at $\frac{\pi}{2}$ and the desired velocity at 0 rad s^{-1}. θ_d, θ_a, $\dot{\theta}_d$ and $\dot{\theta}_a$ are defined as the desired position, the actual position, the desired velocity and the actual velocity in θ respectively. The error terms are defined as θ_e and $\dot{\theta}_e$ respectively and θ_{ce} is used describe the combined error. The use of a superscript $+$, $-$ or 0 will denote whether the error is positive, negative or zero so $\dot{\theta}_e^+$ would represent a positive error in $\dot{\theta}$. The error terms are derived from

$$\theta_e = \theta_d - \theta_a \tag{7.1}$$

$$\dot{\theta}_e = \dot{\theta}_d - \dot{\theta}_a \tag{7.2}$$

Figure 7.3 below depicts a simplified version of the TRMS showing the θ_a, θ_d and θ_e error terms.

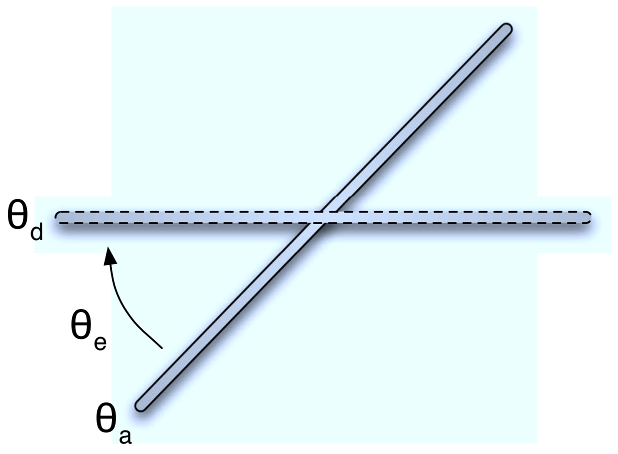

Figure 7.3: An example of desired and actual positions of the system

In this example it is assumed that the system is stationary and at θ_a whilst its desired state is stationary at θ_d. This situation is represented by the following equation:

$$\theta_e^+ + \dot{\theta}_e^0 = \theta_{ce}^+ \tag{7.3}$$

If the system starts to move in a positive direction (i.e. towards θ_d) then $\dot{\theta}_e$ will become negative as shown below:

$$\dot{\theta}_d^0 - \dot{\theta}_a^+ = \dot{\theta}_e^- \tag{7.4}$$

This reduces θ_{ce} until the point where θ_e^+ and $\dot{\theta}_e^-$ are equal. At this point the equation becomes:

$$\theta_e^+ + \dot{\theta}_e^- = \theta_{ce}^0 \tag{7.5}$$

As the difference between θ_d and θ_a decreases, the speed at which $\dot{\theta}_a$ needs to travel to balance the equation to zero also decreases resulting in a variable damping based on distance.

Clearly this is not the main damping control mechanism. This will come from the rate feedback $(\dot{\theta}_d - \dot{\theta}_a)$ **controller agent** action. The **combined error agent** action could be considered to be equivalent to a feed-forward term in that way that it is presented. The ratio of position to rater error action is fixed by the combination process.

7.1.3 Controller Agents

A **controller agent** has the capability to change the speed of the fans and therefore provides the control action. To do this it must first negotiate a contract with an **error agent** using the contract-net protocol described in Section 5.5. Each **controller agent** has an RL algorithm object associated with the **error agent** for which it is attempting to reduce the error. The RL algorithm chosen was the Linear Reward Inaction Algorithm (LIRA) as described in Section 4.1.3. The LRIA was chosen because of its computational simplicity and negligible memory usage.

Consolidating Fan Actions

As with the multi-agent PID framework, it was not practical for each **controller agent** to have direct access to the fan speed. As an alternative to this, a structure was created to allow for multiple **controller agents** to have an effect on the fans at the same time. An example of the structure is shown below in Table 7.1.

Error Agent	**Fan 1 Desired**	**Fan 2 Desired**	**Is Active**
ThetaError	0	0	False
ThetaDotError	1525	1475	True
PhiError	2512	2512	True
PhiDotError	0	0	False
ThetaIntegratedError	0	0	False
PhiIntegratedError	994	994	True
ThetaCombinedError	0	0	False
PhiCombinedError	1274	1274	True
Total	1576.25	1563.75	True

Table 7.1: The structure of the fan control system with example values

Here, the desired input value to the fan is calculated by summing the values of the desired columns and dividing by the number of active fans. Using this technique, all **controller agents** can affect any of the desired inputs in the fan control structure. However, they will only update the desired inputs of the **error agent** for which they are performing an action.

It should also be noted that since the PWM input range for the fans on the TRMS is between

1000μs and 2000μs the total input must be between these two values. However, the desired input values for each **error agent** are not limited. This allows for a large error submitted by one **error agent** to be effective. For example, if the *ThetaIntegratedError* agent had an error that was steadily increasing, due to integral action, and all other **error agents** had zero, or close to zero error, the suggested controller action for the *ThetaIntegratedError* agent might be outside the limits of the fan inputs. However, due to the effect of all the other values in the fan control structure, this action would be attenuated. If the associated desired input values for each **error agent** was limited to within the input range, the *ThetaIntegratedError* agent might not be able to generate enough thrust to reach its desired position.

RL Algorithm Utilisation

When a **controller agent** is selected to perform an action, it first determines if it has controlled the instigating **error agent** in the past. It does this by scanning a list of RL algorithm objects to see if the AID of the **error agent** is already associated with any of them. If so, the **controller agent** then applies the action suggestion from the associated RL algorithm object. Otherwise, it adds a new RL algorithm to its list and uses that. A depiction of this is given at Figure 7.4.

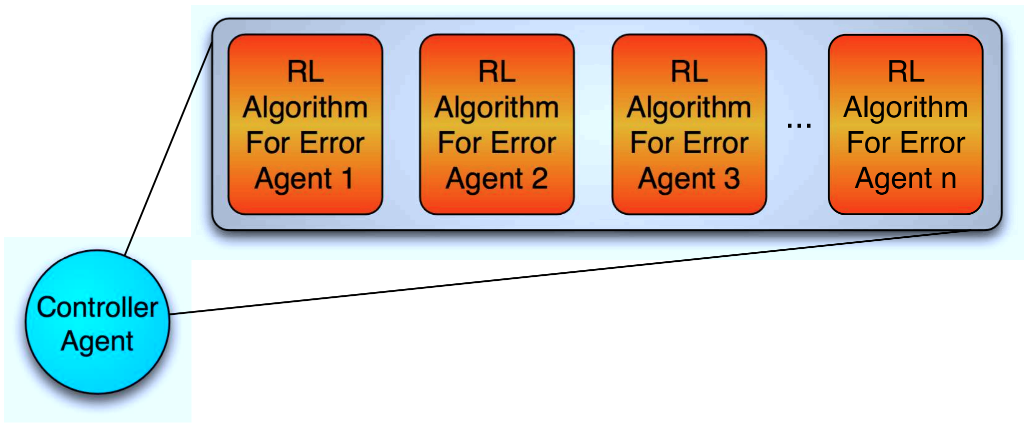
RL Algorithm For Error Agent 1
RL Algorithm For Error Agent 2
RL Algorithm For Error Agent 3
...
RL Algorithm For Error Agent n
Controller Agent

Figure 7.4: A **controller agent** with a list of RL algorithms, each associated with an **error agent**

Using this approach, a **controller agent** can dynamically add a new RL algorithm object whenever a new agent calls for an error reduction. This functionality would be potentially useful for an extension to the framework where **error agents** are dynamically assigned to the control system as in Stothert & Macleod (1997).

7.2 RL Algorithm Implementation

The LRIA requires that actions are discretised, or characterised, otherwise probabilities could not be assigned. In the proposed system, whilst it is possible to break down the suggested fan inputs into combinations of actions, these actions are not entirely discrete because error terms and scaling factors must be employed. The following section will describe this in more detail.

7.2.1 Possible LRIA Actions

In the LRIA implementation there are 4 possible categories of action that can be selected. These are shown in Table 7.2:

Action	**Fan 1 Action**	**Fan 2 Action**
1	-1	-1
2	-1	$+1$
3	$+1$	-1
4	$+1$	$+1$

Table 7.2: The four actions available to the LRIA

The numerical values in the **Fan 1 Action** and **Fan 2 Action** columns are denoted $F1_a$ and $F2_a$ respectively. They correspond to the *direction* in which the fan propeller will spin. Once the LRIA has made an action selection, the desired fan values, as seen in Table 7.1, for example, are updated using the following formulae:

$$\begin{aligned} F1_d &\Leftarrow \beta e F1_a + K_i \\ F2_d &\Leftarrow \beta e F2_a + K_i \end{aligned} \tag{7.6}$$

Where β is a scaling factor, e is the value of the error term, K_i is the input constant and $F1_d$ and $F2_d$ are the desired fan values for Fan 1 and Fan 2. In the case of the TRMS $K_i = 1500$ as an input to the fans of $1000\mu s$ yields a maximum fan speed in the negative direction and an input of $2000\mu s$ yields a maximum fan speed in the positive direction. The value of the scaling factor β converts the error value into a suggested input value for the fans. For example, a scaling factor of 200 using Action 4 would suggest a PWM fan input of 1700μs for both Fan 1 and Fan 2. In the current implementation, β is set heuristically. In future implementations, this parameter could be the subject of a suitable learning strategy.

7.3 Determining Correct Actions

The dynamic nature of the TRMS means that determining whether an action has been successful or not is not always straightforward. This has dramatic implications for RL as, without suitable action rewards, convergence towards appropriate strategies becomes difficult, if not impossible. This section deals with potential methods for overcoming this problem.

7.3.1 Error Reduction

The simplest method for establishing whether an action is good or bad is to take a snapshot of the error before the action has been performed and compare it to the error at some predetermined time after the action. If the error has been reduced then it may be assumed that the action was a good one and the opposite is true if it has been increased. An example of this is shown in Figure 7.5.

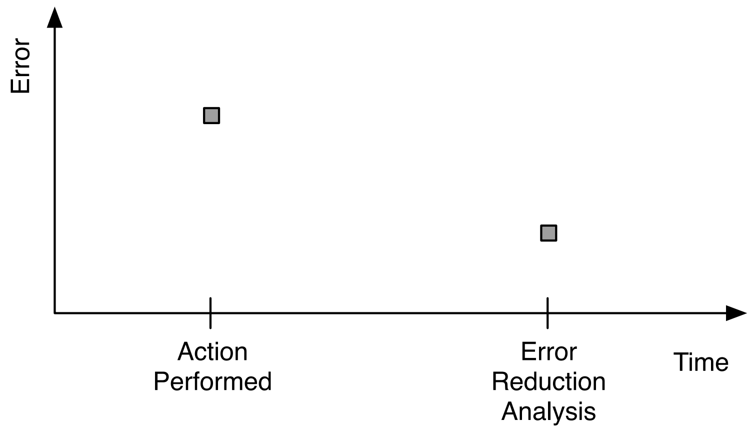

Figure 7.5: Example of a good action using the Error Reduction method.

However, as Figure 7.6 shows, simply using error reduction as a measure of action success can lead to erroneous conclusions.

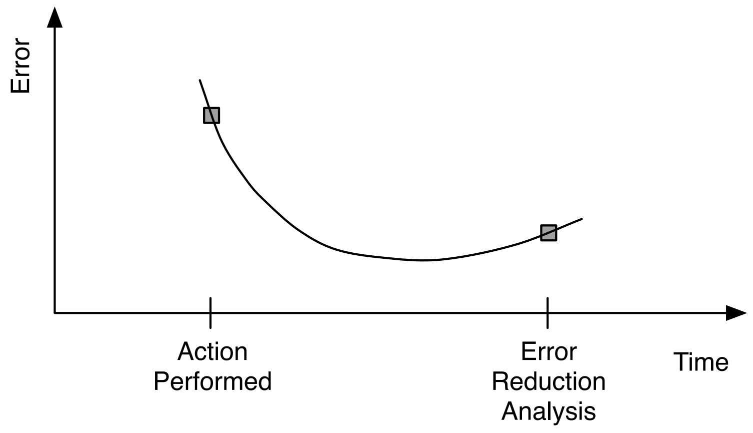

Figure 7.6: Example showing a reduction in error using a bad action.

It can be seen that in this case, the error reduction is an artifact of residual angular momentum at the initial sample point. The system was trending towards a reduced error when the action was performed and it actually had a negative effect. From this example it can be concluded that the error reduction method will work if the appropriate angular momentum is zero. Otherwise, it may induce an incorrect result.

7.3.2 Gradient Based Techniques

A more reliable technique would be to look at the gradient change over the course of the action effect. Two examples follow:

Total Gradient Comparison Technique

This technique eliminates the problem of angular momentum affecting the measurement by analysing the difference between the initial gradient and the total gradient. The calculation this is given by

$$\left(\left(\frac{e_1 - e_0}{t_1 - t_0}\right) - \left(\frac{e_N - e_0}{t_N - t_0}\right)\right)$$

where N represents the number of samples taken. An illustration of this technique is depicted at Figure 7.7.

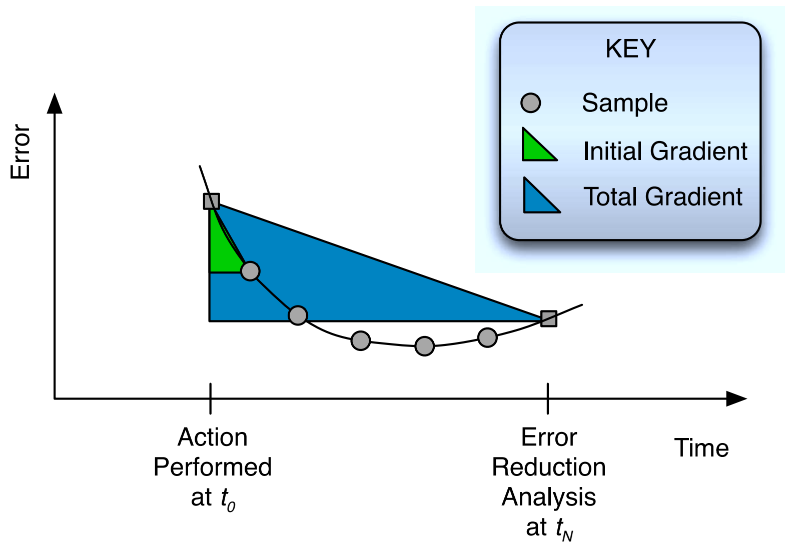

Figure 7.7: Examining the effectiveness of an action based on the total gradient

In this example, the final gradient is larger than the initial gradient indicating a *bad* action.

Final Gradient Comparison Technique

This is very similar to the total gradient comparison technique. However, in this case, the comparison is made between the initial gradient and the final gradient. The final gradient is defined as the gradient between the penultimate and final samples. The calculation is given by

$$\left(\left(\frac{e_1 - e_0}{t_1 - t_0}\right) - \left(\frac{e_N - e_{N-1}}{t_N - t_{N-1}}\right)\right)$$

an illustration of this technique is shown at Figure 7.8.

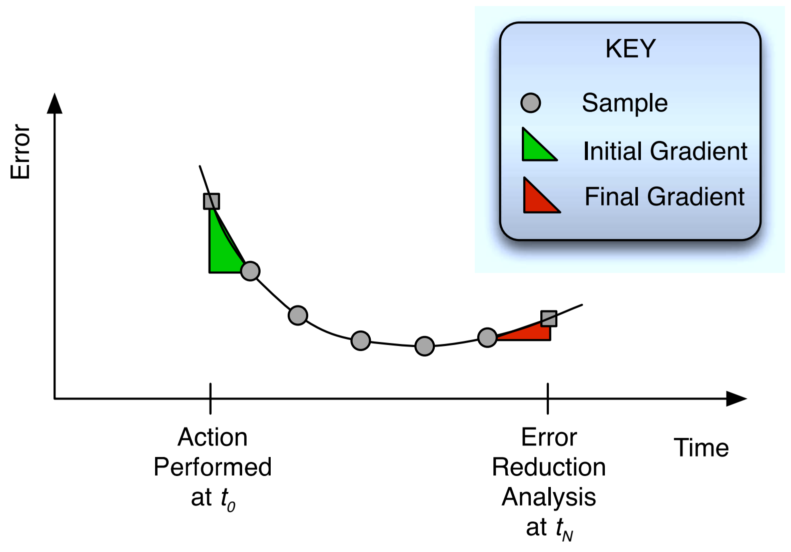

Figure 7.8: Examining the effectiveness of an action based on the final gradient

These two methods will suggest that the action has been successful if the initial gradient is greater than either the total or final gradients for positive initial errors. The opposite is the case for negative errors. For an error reduction to have taken place, the initial gradient will be less than the total or final gradients.

7.3.3 The Iterative Action Score

The iterative action score is another gradient based technique. However, it assigns a score for the action, based on the performance of the action over the entirety of the sampling period. It operates upon a sequence of contiguous gradient comparisons. The formula for calculating the iterative action score, Λ, is given in Equation 7.7.

$$\Lambda = \sum_{i=2}^{N} \left(\left(\frac{e_i - e_{i-1}}{t_i - t_{i-1}} \right) - \left(\frac{e_{i-1} - e_{i-2}}{t_{i-1} - t_{i-2}} \right) \right) \tag{7.7}$$

Here, e_i represents the current sample of the error and t_i represents the current time sample. The calculation for Λ can only begin once the third sample has been made as at least two gradients are required. This method for checking the score ensures that all samples are evaluated. Using the iterative action score technique is quicker, in terms of agent interaction, than the previous gradient methods, as the computation is done at the time of sampling rather than in a subsequent calculation. Figure 7.9 depicts an example of the iterative action score in use.

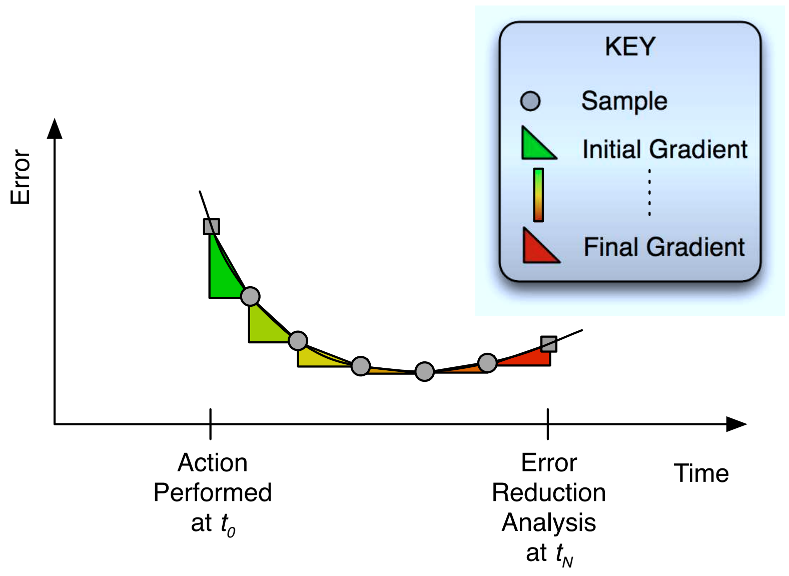

Figure 7.9: Examining the effectiveness of an action based on the iterative action score for an inappropriate action

Due to its superiority, both in terms of computational overhead and information captured, the iterative action score was implemented in the framework.

7.4 The Learning Phase

To ensure that the **controller agents** can learn appropriate actions for each **error agent**, the learning phase is sequenced so that only one **error agent** is active at any one time.

7.4.1 Ensuring Learning Coverage

When the **controller agents** send their proposals, they add a utility cost to the proposal so that the **error agent** can determine the best price. This allows for socio-economic analogies to be implemented.

In this instance, the proposed cost is a function of the highest action probability in the associated LRIA object. If one agent's LRIA has an action with a higher probability of selection than another's, the **error agent** will not contract it to reduce the error because cheaper offers will be available. This can be thought of as analogous to the agent charging a higher price for an action to be completed, based on how confident it is that it can complete the action successfully. Those with a higher confidence of completing the task successfully will charge a higher price. The effect of this is that **controller agents** converge toward an action at the same rate because the least confident agent will always be selected.

7.4.2 Zeroing the TRMS

Since the testing of the multi-agent framework is done in simulation, it is possible to zero the TRMS after each evaluation so that the starting point for each action is the same. This is known as episodic learning and has two main advantages over continuous or online learning. Firstly, it clears the residual fan forces and system velocities that affect the ability of the RL algorithm's to determine the action outcome. Secondly, it allows for statistical analysis techniques to be implemented using the values from the iterative action score, since the values can be correlated.

This notion of episodic learning can be crucial when using RL techniques, especially for control problems, otherwise certain systems can fall into an unrecoverable state. This is certainly true for an inverted pendulum, as described in Section 2.7, where letting the pendulum fall could be ruinous for the learning procedure. Whilst there are a number of papers describing RL for an inverted pendulum (Anderson (1989), Doya (2000), Kobori et al. (2002)[1]) the measure of success is always how few episodes it took to *learn* a control scheme.

Over the course of this research, it was postulated that it might be feasible the multi-agent system to learn a ruleset for controlling the TRMS in an online fashion. One reason for this thinking was that the TRMS will eventually return to its equilibrium position if the inputs are zeroed.[2], so there is no unrecoverable state. It was also felt that the iterative action score algorithm could provide enough detail to the learning procedure to select *good* actions regardless of the TRMS state. However, it was found that there were still difficulties with determining useful actions in this situation, even when using the iterative action score method. It was concluded that this was because previous learning actions drive the system states widely across the state space, not just in terms of the movement of the TRMS, but also in terms of the residual forces being presented by the fans at the start of the next learning trial. Attempts were made to use scores from the gradient techniques, detailed in Section

[1]This work provides details on RL for *swing-up* of the inverted pendulum, so letting the pendulum fall is potentially less ruinous.

[2]The term *equilibrium position* here means that there are zero velocities in both rotational planes and also a zero θ position.

7.3.2, to alter the LRIAs update factor α. This proved difficult because reference to a very *good* action is required before a scaling factor can be established for determining an action's value. Attempts were made to use values of the iterative action score to provide a scaling factor using statistical analysis techniques. However, once again, due to the effect of the previous actions the values were uncorrelated, so this kind of analysis was unviable. Because of these difficulties, it was decided that online learning of this system was not achievable in its current form, and so a more consistent, episodic method for learning the system was employed.

7.4.3 Scaling the Update Rate

One issue that was encountered during the development of the framework was that of multiple correct actions skewing the learning process. In the majority of cases, only one fan action produces a reduction in the error. However, with positive errors in the ϕ plane, there are three actions that can do this. This is, once again, due to the inequality in the forces generated by the fans when they are in the forward and reverse directions. A positive moment in the ϕ plane is generated when both fans are turned on in the forward direction. Additionally, equal and opposite fan speeds for the two fans will yield a positive moment, albeit smaller. Figures 7.10, 7.11 and 7.12 show the response in ϕ to actions 2, 3 and 4 (c.f. Table 7.2).

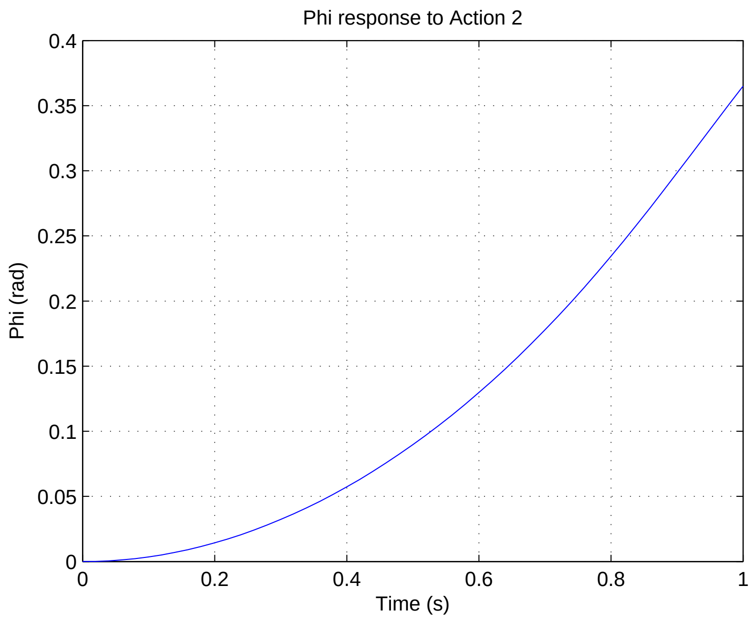

Figure 7.10: The ϕ response to Action 2 being applied to the TRMS

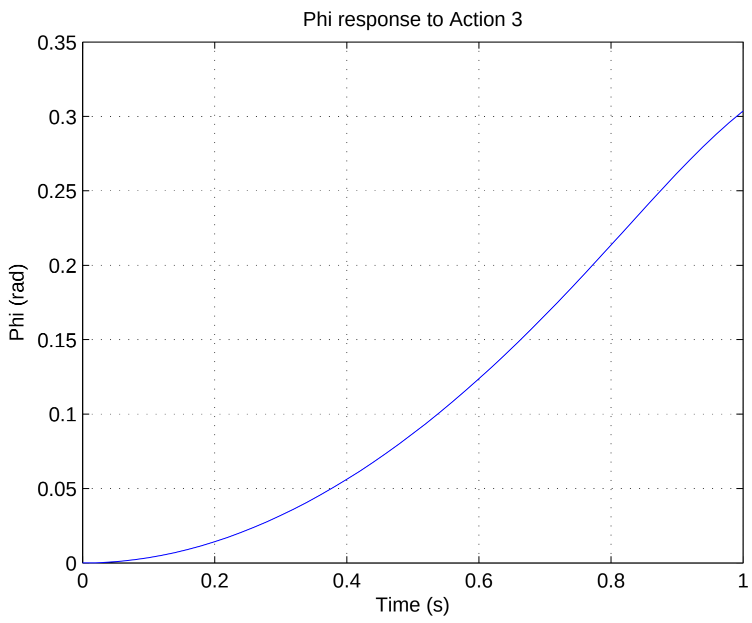

Figure 7.11: The ϕ response to Action 3 being applied to the TRMS

As expected, the response in ϕ is very similar for actions 2 and 3 as one fan is producing slightly more force than the other. The slight discrepancy in response can be attributed to the θ angle produced by the two actions being opposite and hence the $F_{\theta1}$ and $F_{\theta2}$ forces producing contradictory additional force in the ϕ plane.

The ϕ response to action 4 is similar in shape to the other responses but shows a displacement in the ϕ plane that is close to ten times that of the previous two actions.

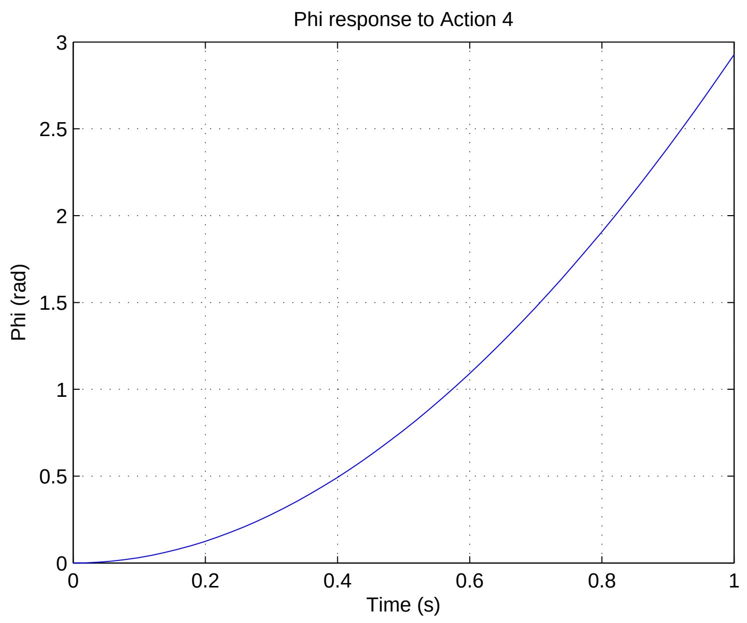

Figure 7.12: The ϕ response to Action 4 being applied to the TRMS

This was an issue for the LRIA, as actions are not deemed to be better or worse than others, just *good* or *bad* in terms of learning appropriate actions. All three actions are deemed to have equal *effectiveness*. For this reason, the update rate α, which increases the probability of good actions being selected, was altered to be a function of the iterative action score rather than simply a constant, as used in a standard LRIA implementation.

This was implemented by adding a parameter to the error message ontology that allowed for action scores to be communicated between **error agents** and **controller agents**. Every time an **error agent** informs a **controller agent** about the efficacy of the action it has performed, it includes the iterative action score of that action and of the most effective action it has encountered. This way, instead of using a constant, such as, say, $\alpha = 0.2$, the update rate can be scaled against the best action as follows:

$$\alpha = \frac{0.2\Lambda_{current}}{\Lambda_{best}} \tag{7.8}$$

where Λ denotes the iterative action score. By doing this, whilst the likelihood of inappropriate actions being selected and judged to be *good* is unchanged, the probability of convergence towards them is greatly reduced in favour of actions that dominate them.

7.4.4 Determining Appropriate Fan Action Gains

Whilst there are different computational methods for determining appropriate gains for PID controllers, most notably those of Ziegler & Nichols (1993), all rely on knowledge of the error terms. This is understandable. It would inadvisable, to say the least, to actively disregard plant information when designing a control system, if that information is available.

A number of approaches were investigated for determining appropriate gains, including analysis of the error terms by the **controller agents** to attempt to ascertain the error type (e.g. proportional, integral or differential). It was felt, however, that this violated the principles of the framework design, i.e. that **controller agents** should learn to reduce an error regardless of its type. Another approach that was considered involved reducing the gains for control suggestions that exceeded a certain value as it was thought that perhaps this might reduce the effect of runaway integral terms. This was rejected firstly because it required hard coding of parameters and also because the scheme required different limits for both the ϕ and θ **error agents**. The matter remains an open problem for future consideration.

7.5 Framework Overview

The inner workings of the framework have been described previously in this chapter. However, this section aims to provide a broader overview of the framework.

7.5.1 Real-Time TRMS Simulation

It was decided early on in the research that the TRMS simulation should run in real-time. This decision was made primarily to provide an appropriate time-frame for the multi-agent system to execute and, therefore, control the system. It was also felt that this decision would make it easier to transfer the framework to control a real TRMS in future work.

The real-time clock was implemented by using a simple integration routine that used a variable integration period to calculate the TRMS position/velocity from the acceleration values provided in Equations 3.13 and 3.14 of Chapter 3. The integration period varies due to variations in the processer load when executing the TRMS simulation loop, i.e. it sometimes takes longer to simulate one iteration of the loop than others. To keep the simulation running in real time, this had to be accounted for. The integration period is also used to calculate spin-up time of the fans. This real-time simulation was not only useful for the implementation of the multi-agent framework, but also made analysis of the TRMS from the Java 3D model and LiveGraph outputs much more straightforward.

7.5.2 Complete System Hierarchy

Since the complete system comprises a synchronous plant simulation and GUIs, combined with an asynchronous multi-agent system controller, correct interaction between these elements is key to the functionality of the system as a whole. Details of the GUIs will be discussed in Section 7.5.3. However, Figure 7.13 provides a representation of the system hierarchy.

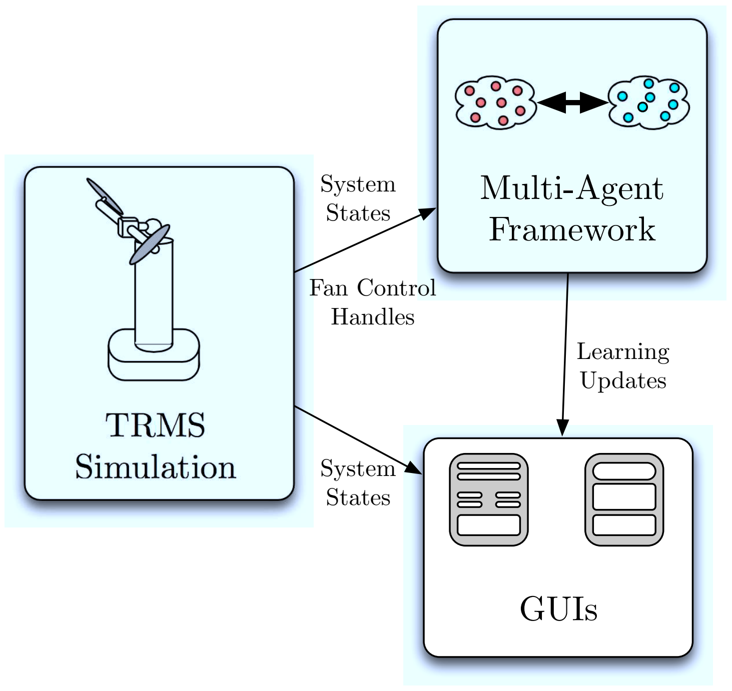

Figure 7.13: A representation of the complete system hierarchy

Whilst it might seem strange that there is no information flow from the multi-agent framework to the TRMS, this is actually a byproduct of the inner workings of JADE. It is not possible, due to JADE's implementation, to pass objects out of a JADE based multi-agent system and into a Java program. However, this problem can be alleviated by passing action listeners to the agents, which provides a method for altering fan speeds that are mimicked in the TRMS Java simulation. The use of an action listener to perform fan actions also alleviates any synchronisation difficulties between the TRMS and the multi-agent framework, as the action listener in the TRMS periodically checks to see if an action has been performed. This means that the framework can update the fan speeds at any time and it will get picked up by the TRMS at the next cycle.

7.5.3 Monitoring the System Progress

The nature of the system hierarchy and the intricacies of JADE make monitoring and debugging the system extremely challenging. The JADE debugging tools, outlined in Section 5.11.3, are useful for determining whether agents act as they should, but most debugging must take place from a global perspective, i.e. is the entire system working as expected? Since the system relies on the TRMS simulation and the multi-agent framework running concurrently, but separately, standard debugging techniques are impractical. It is not possible to step through each line of code to determine whether it functions correctly because there is such a huge level of reliance on calculations that are occurring elsewhere. For this reason, a number of GUIs were implemented using Java's *swing* library to oversee the system operation and, in conjunction with large numbers of system print statements, debug the code.

One of the two main GUIs that were implemented is the **Controller Probability** GUI, which oversees the learning phase, as shown in Figure 7.14.

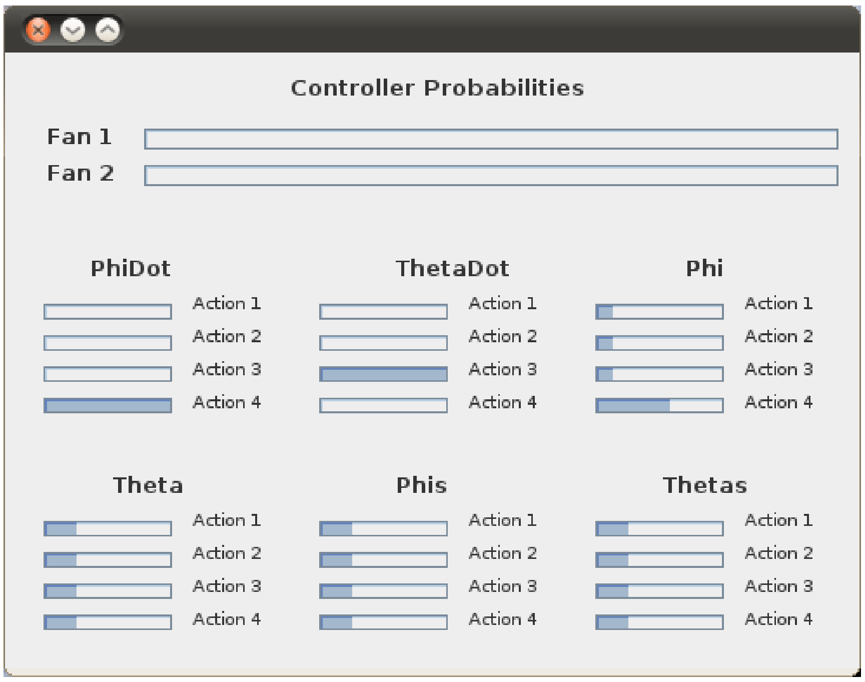

Figure 7.14: GUI for determining the magnitude of fan actions and overall fan inputs

This shows how the learning process is progressing by averaging the action probabilities for all

controller agent's LRIAs, associated with each **error agent**. In the state shown, the GUI demonstrates that all **controller agents** have converged on a solution for the *PhiDotError-Agent* and the *ThetaDotErrorAgent* and are currently learning appropriate actions for the *PhiErrorAgent*. Since this GUI relies on information from the JADE based multi-agent element of the system, synchronising the updates must be performed with care. Fortunately, the *swing* library possesses a method called *invokeLater* specifically designed for such situations. It allows for the GUI update to be placed in a queue until the specified GUI is ready to process it, thereby bypassing any synchronisation issues.

The **Fan Action** GUI (Figure 7.15) shows the value of each element in the fan control structure, and also the total output from the controller to the fan input.

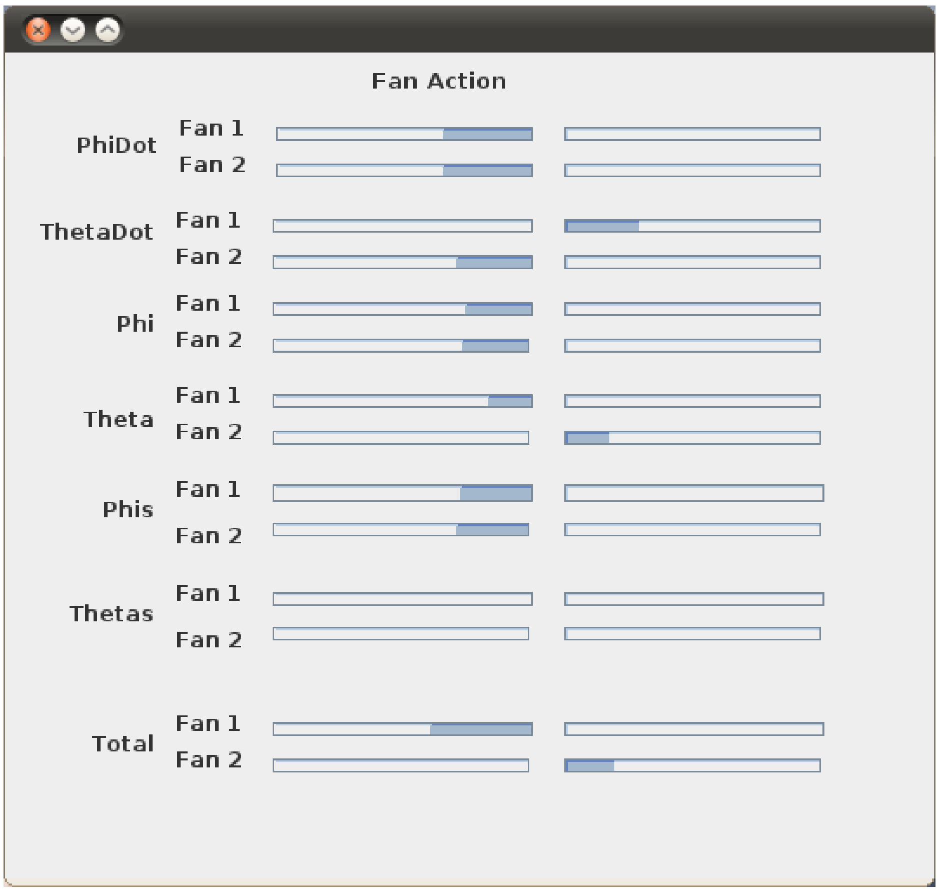

Figure 7.15: GUI for determining how the **controller agents** learning is progressing

This example is taken from when the framework is in the control phase, so all agents are active. The **Fan Action** GUI is particularly useful, firstly, for determining whether all **error agents** are active, and secondly, for determining whether the error produced by an individual agent is dominating the control scheme.

These two GUIs provided invaluable insight into the framework throughout the development process. Without them, debugging would have been extremely difficult and time consuming.

7.5.4 Agent Computational Flow

Whilst the importance of the learning phase for the controller to *learn* a control strategy has been highlighted, there is actually very little difference between the learning phase and the control phase in terms of computational flow. Figure 7.16 depicts the computational flow for both an **error agents** and a **controller agents**.

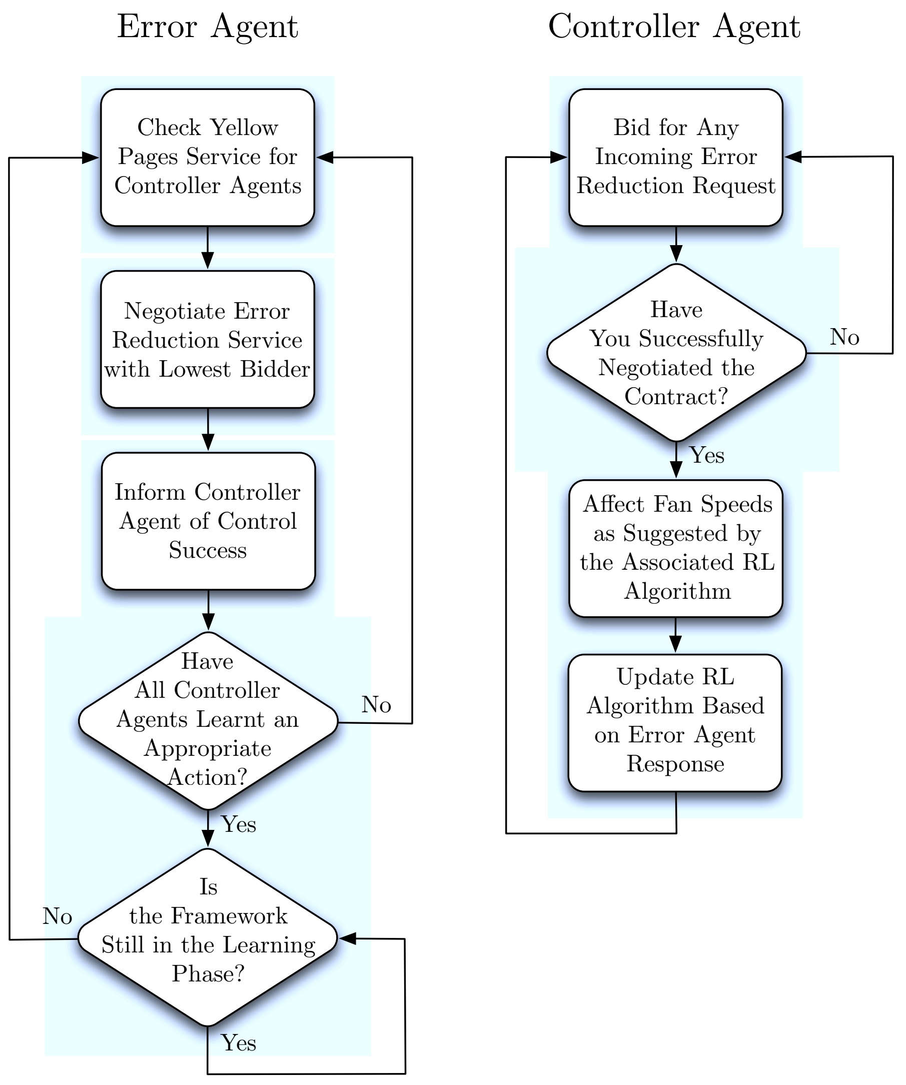
Error Agent
Controller Agent
Check Yellow Pages Service for Controller Agents
Negotiate Error Reduction Service with Lowest Bidder
Inform Controller Agent of Control Success
Have All Controller Agents Learnt an Appropriate Action?
No
Yes
Is the Framework Still in the Learning Phase?
No
Yes
Bid for Any Incoming Error Reduction Request
Have You Successfully Negotiated the Contract?
No
Yes
Affect Fan Speeds as Suggested by the Associated RL Algorithm
Update RL Algorithm Based on Error Agent Response

Figure 7.16: **error agent** and **controller agent** computational flow

As can be seen, there is only one minor alteration in the computational flow from the learning phase to the control phase and that is in the **error agent**. All **error agents**, except the **error agent** under examination, are suspended and do not function when in the learning phase. The additional conditional box at the end of the **error agent** computational flow ensures that **error agents** become dormant once the **controller agents** have all learnt an appropriate action for it. When the control phase begins, all **error agents** are then activated and utilise the same strategy as they did when in the learning phase.

7.5.5 Summary

As noted in Section 5.12.1, the field of MARL is primarily focussed on the development of algorithms that use *static* problems, often games, as a testbed. There is some research into dynamic games where agents affect the environment where others are learning. However, in the majority of cases, a lot of attention is placed on convergence rather than problem solving. With the development of this framework, no emphasis is placed on either proof or speed of convergence. The primary focus has been to produce a structure that can function within the proposed constraints. A number of difficulties were encountered in the design process, some of which were alleviated. One, the learning of an appropriate β scaling factor, is a matter for future research.

It was presumed, as is true with PID controllers, that if there was no parameter tuning of β, that the control system would be completely unstable. However, as is shown in Section 7.6.2, with one minor alteration to which **error agents** are present in the system, stability can be achieved.

7.6 Results

This section details the results of both the learning and control stages of the multi-agent control scheme.

7.6.1 Learning the Appropriate Actions

For the framework to be able to control for the TRMS, it must first *learn* appropriate state action pairings. Figures 7.17 and 7.18 depict the the learning progress of the LRIA in each **controller agent**. The graphs show the average probability of action selection between 8 **controller agents**, learning for each of the **error agents**, averaged over 10 runs of the framework. Since the probabilities in the LRIA are only updated when an action is deemed successful, the graphs do not include the unsuccessful actions.

As can be seen in Figures 7.17 and 7.18, **controller agents** attempting to reduce an error in θ, converge to the use of Action 3, with **controller agents** attempting to reduce an error in ϕ, converge to the use of Action 4. This is consistent with the expected action set. However, there is a notable difference between the two planes; the action probability graphs for ϕ errors apparently show all four action probabilities, whilst the graphs for θ errors appear to only show Actions 3 and 4. The apparent differences between these graphs are actually a byproduct of the number of perceived correct actions in each plane, as described in Section 7.4.3. For errors in θ, Action 3 is the only action that will yield a *good* iterative action score. Since probabilities are only updated when an action is deemed to be *good*, all actions, other than Action 3, will decrease in probability at the same rate. On the graph this is represented as a single Action 4 line, as this is the last action to be plotted. Whilst Actions 1 and 2 display identical learning behaviour, they are simply obscured by Action 4's plot.

In the case of the errors in ϕ, there are three actions that yield positive results. However, their probabilities do not get increased at the same rate as Action 4 because of the scaling factor α, also discussed in Section 7.4.3. It can be seen that the probability of selection of Action 1 is consistently lowest, as it is the only action that will not produce a correct action score under any circumstance, and will therefore never receive a selection probability increase.

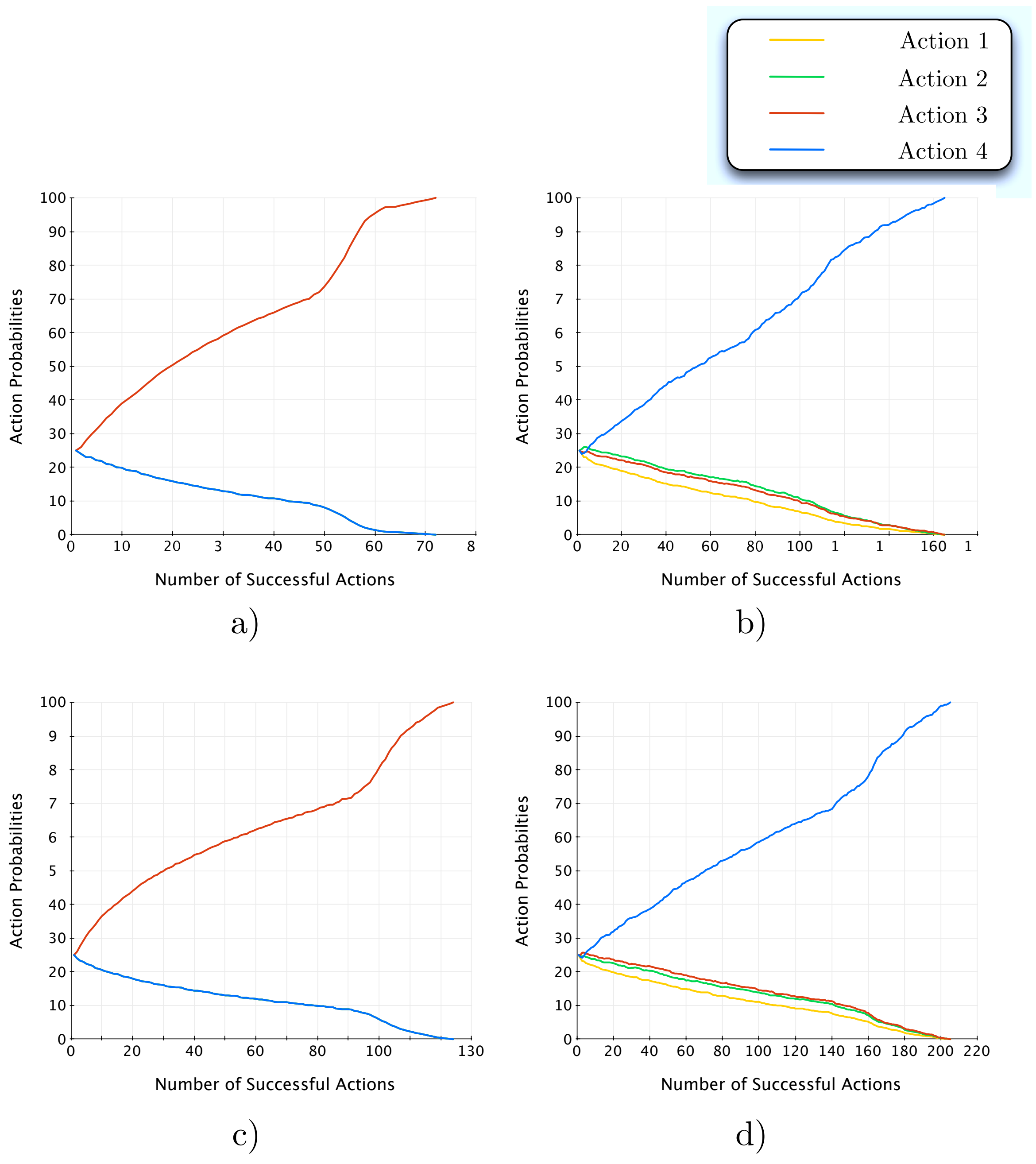

Figure 7.17: Learning probabilities of a) Theta, b) Phi, c) ThetaDot and d) PhiDot **error agents**

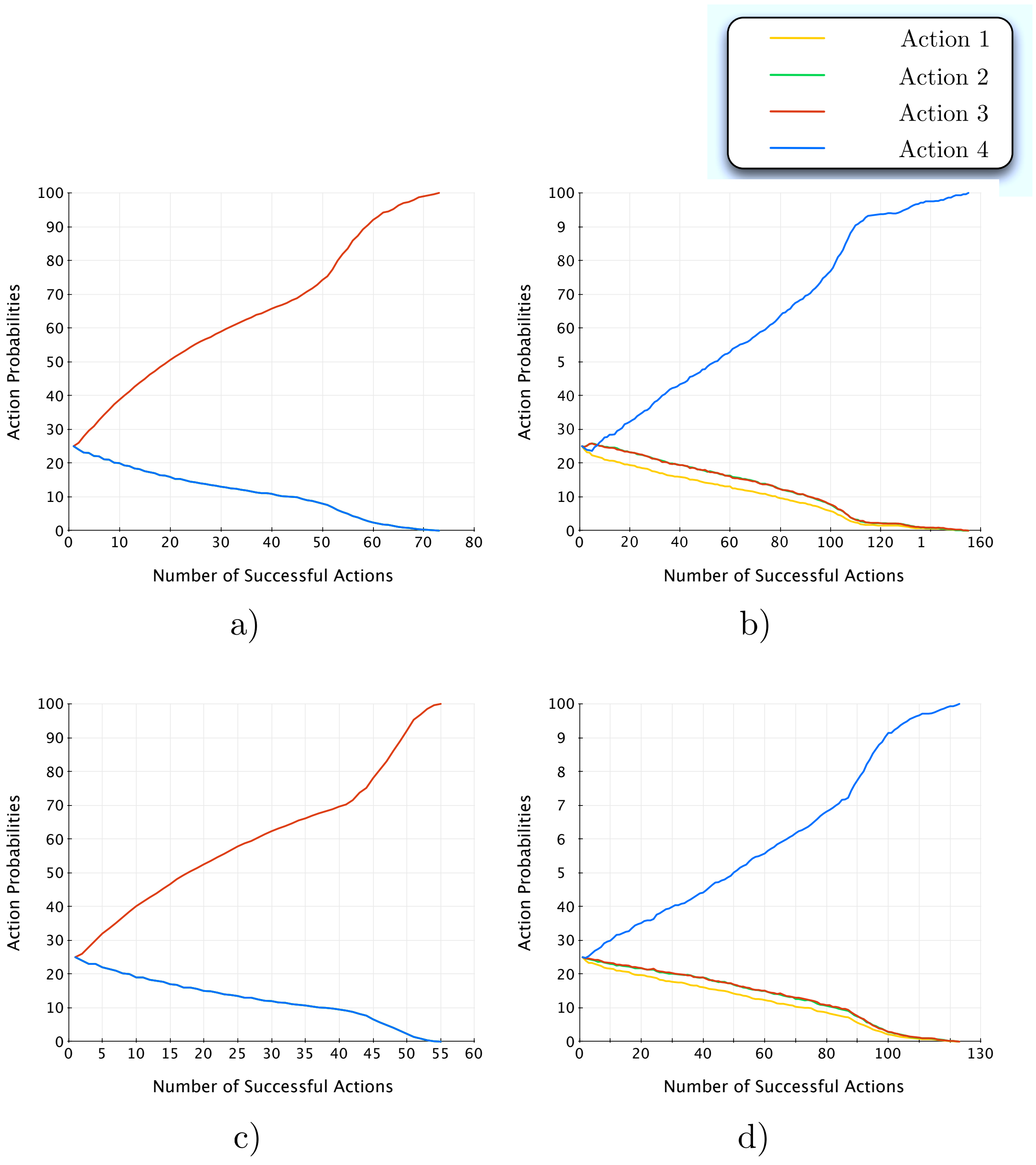

Figure 7.18: Learning probabilities of a) ThetaCombination, b) PhiCombination, c) ThetaIntegral and d) PhiIntegral **error agents**

In the current configuration with eight **error agents** and eight **controller agents**, it takes in the order of 900 seconds for all **controller agents** to converge on an action for all **error agents**. There is a slight variation between runs of around 20 seconds due to the probabilistic nature of the LRIA affecting the learning process. Detailed measurements of these variations are not provided here, as the focus of the research was on control rather than convergence.

The results of the learning processed are discussed in more detail in Section 7.6.3.

7.6.2 Controlling the TRMS

With the appropriate actions *learnt*, a number of runs were carried out to test the frameworks ability to control the TRMS. In each of these runs, the scaling factor β, that each LRIA uses to suggest fan input values, was set to a blanket value for all **error agents**. However, in Table 6.1 of Chapter 6, which shows the gain values for the PID controller, it can be noted that larger gains are necessary for the θ plane than the ϕ plane. It was found that, because of this incongruity between required force in both planes, the controller in its current form was unstable. It was determined that the cause of the majority of the instability lay in the error terms produced by the *PhiIntegralErrorAgent* which dominated the controller's suggested outputs. For this reason, the *PhiIntegralErrorAgent* was removed from the framework for the remainder of the testing. It was felt that this was a valid compromise, as integral action is not required to keep the TRMS in a stationary non-zero position in ϕ, whereas it is in θ. However, the consequence of this alteration is a finite position error in ϕ.

Figures 7.19 and 7.20 show the results of the controller for various desired stationary positions in both θ and ϕ with the scaling factor β for each LRIA set to 150. For plots of the runs with different β scaling factors, see Appendix A.

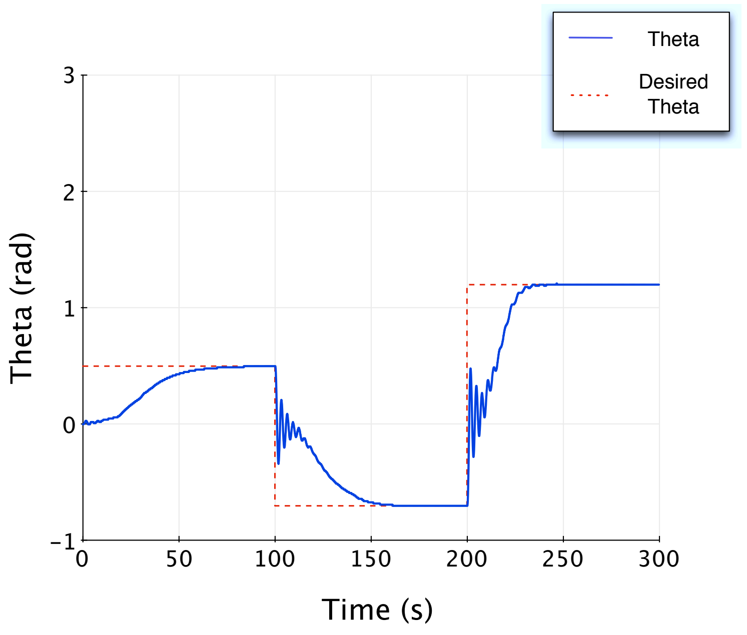

Figure 7.19: Framework control of θ with the scaling factor at 150

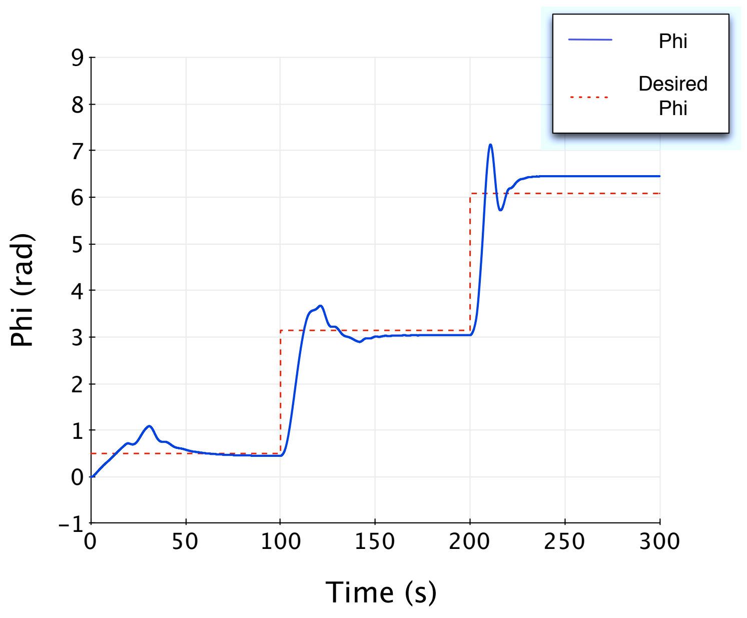

Figure 7.20: Framework control of ϕ with the scaling factor at 150

Figure 7.21 shows the fan thrusts of this run.

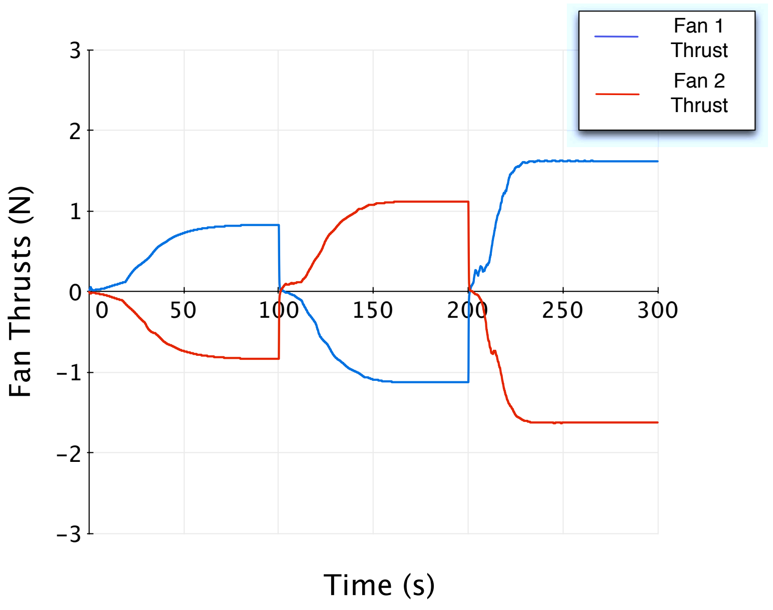

Figure 7.21: Fan thrusts of framework control with the scaling factor at 150

7.6.3 Discussion of Results

The learning phase of the framework shows the LRIA's ability to converge on the expected solution for the defined action sets. For action learning in the ϕ plane, there are examples (particularly Figure 7.18b) where the appropriate action probability decreases at the start in favour of other actions. This is because, for early action selection, the value of Λ_{best} might be a result of a suboptimal action due to the random selection procedure of the LRIA. However, once the optimal action has been selected for the first time, the value of Λ_{best} increases making the α probability update factor much smaller for suboptimal actions. At this point the optimal action gains traction and dominates the learning procedure.

Whilst speed of convergence was never the focus of this research, there are a number of avenues that could be explored to reduce the convergence time. An obvious starting point would be to investigate how the number of **controller agents** populating the framework affects operation. With fewer agents, the learning period would undoubtedly be shorter, but this would certainly have implications for control, as fewer agents would be available to concurrently alter fan speeds. Another obvious component to investigate would be the LRIA update rate α. If this was increased, updates would be more effective, but the learning process could be compromised as suboptimal actions would have more chance of convergence.

When analysing the performance of the framework in the control phase, it can be seen that it is slower to settle than both the PID controller presented in Section 3.5.2 and the multi-agent PID controller described in Chapter 6. There is also a discernible steady state error in the ϕ plane, as expected, due to the lack of integral action. Nevertheless, it is apparent that, despite the lack of a β scaling factor tuning process, the framework *can* learn to control the TRMS moderately well. Both θ and ϕ reach a steady state value, and there are no signs of instability. The $\%OS$ is smaller in the ϕ plane and non-existent in θ.

This is the first time that a distributed, multi-agent reinforcement learning based system has been used to control such a dynamic non-linear system as the TRMS. The imposed restrictions on the learning process, such as the formalisation of error terms to be homogeneous, make these results compelling. This is especially true given the blanket, system wide β value that scales all actions.

7.6.4 Summary

This chapter has provided details of the development and implementation of a multi-agent, RL based, emergent control structure. It has detailed the challenges involved in developing such a system and the methods that were employed to diminish their impact. It has also presented and evaluated the results when the framework is applied to a dynamic, non-linear system such as the TRMS.

Chapter 8 will analyse these results further and draw conclusions on the impact of the work. It will also highlight the contributions that have been made and discuss the potential for further work.

7.7 Chapter Bibliography

Anderson, C. W. (1989), 'Learning to Control an Inverted Pendulum using Neural Networks', *Control Systems Magazine* **9**(3), 31–37.

Doya, K. (2000), 'Reinforcement learning in continuous time and space', *Neural Computation* **12**(1), 219–245.

Kobori, N., Suzuki, K., Hartono, P. & Hashimoto, S. (2002), Learning to Control a Joint Driven Double Inverted Pendulum using Nested Actor/Critic Algorithm , *in* 'Proceedings of the 9th International Conference on Neural Information Processing', pp. 2610–2614.

Stothert, A. & Macleod, I. M. (1997), 'Using intelligent agent templates for dynamic structuring of distributed computer control systems', *Engineering Applications of Artificial Intelligence* **10**(4), 335–343.

Ziegler, J. G. & Nichols, N. B. (1993), 'Optimum Settings for Automatic Controllers', *Journal of Dynamic Systems Measurement and Control* **115**(2B), 220.

Chapter 8

Conclusions and Further Work

Contents

This thesis has provided an overview of some of the attractive properties of distributed artificial intelligence for control engineering problems, whilst providing insight into why the design of such systems can be challenging. It also details the development of a multi-agent reinforcement learning framework to try and alleviate some of the difficulties inherent when designing for emergence.

One fundamental principle that this framework adopts, is

> *no agent should have information about the system it is applied to, simply a disposition towards error reduction regardless of form.*

This principle was formulated from certain notions found within biological evolution. The evolutionary process, at a chemical level, has no concept if something will be advantageous or detrimental; a single celled organism can only propagate its genetic code if it can survive long enough to do so. The processing of a combination of inputs to achieve certain goals has developed through many evolutionary iterations; not by any design process, but because each iteration was either advantageous to survival or neutral. This framework uses RL as this iteration process with a view to developing emergent strategies for controlling the TRMS.

The TRMS was chosen as a plant not just because of its dynamic, non-linear nature, but also its potential for distributability. It was felt that its two rotational degrees of freedom, although cross coupled to some extent, provide suitable operational distinction for the learning process.

This chapter draws conclusions on the work described in previous chapters and discusses the potential for further work.

8.1 Conclusions

As was discussed in Section 1.2, there are inherent difficulties that are associated with the design of emergent systems. This is due to emergence being a byproduct of interaction rather than discernible and distinct sequential computation. For this reason, the majority of distributed artificial intelligent systems that display emergent properties use very simple agents enacting very simple rules.

MARL offers a number of appealing characteristics for the design of emergent systems as, if implemented creatively, it is the agents' interaction that can be *learnt* instead of individual strategies. The majority of past research into MARL is focussed on optimisation of convergence over standard RL techniques. As such, most examples use static games as a testbed, with the occasional foray into dynamic games and optimisation problems.

The work described in this thesis takes large strides towards addressing these issues through the development of a framework where much of the plant information has been hidden. By doing this, the difficulties of designing emergence are replaced by ones of representation of system information. However, as is shown by the results in Section 7.6.2, it is possible for multiple agents to *learn* an emergent control strategy when the system information is represented in a very simplistic fashion, such as the value of the error term in an indeterminate plane. This allows for a generic machine learning scheme to be adopted so that the framework need not be as plant specific as is necessary for the majority of control engineering strategies.

The results of the framework controller do indeed show that MARL can produce working results for more dynamic problems when episodic action learning is utilised. In the case of the TRMS, the residual effects of previous actions made online analysis of action correctness problematic, in its current form, without a system reset. However, this is in keeping with standard practices where RL is used for control engineering, and there is certainly scope to investigate the possibility of performing online RL for systems that are not so affected by residual forces.

When analysing the system in terms of adaptivity, whilst the framework does not display true autonomy, it does exhibit a degree of robustness as **error agents** can be duplicated or combined to offer redundancy. **Controller agents** also offer robustness as they all offer the same functionality, namely error reduction. In other words, the system degrades gracefully.

Whilst the control scheme the framework produces would not be able to compete with one that had be designed using standard control engineering techniques, it is not the performance of the controller that is under consideration. The fact that control is achieved at all, given the imposed restrictions, is striking.

Although still in its infancy, this approach has demonstrated potential for producing emergent systems that could be adapted to a variety of complex problems. Its power lies in its simplicity of design, which grants the system developer the ability to assign extremely simple learning goals to the agents. This diverges from standard RL or MARL techniques where the goals of the system have to be well defined, and hence understandable, for learning to take place. This framework provides potential for much less well defined goals to be feasible, since the problem complexity is handled by the learning procedure, not the system developer.

Chapter 1 made the following hypothesis statement:

> A multi-agent approach is *viable* for the development of a distributed, self-organising, self-learning control scheme for a dynamic, non-linear plant.

When this statement is considered in the context of this work, the results (shown in Figures 7.19 and 7.20 of Chapter 7) clearly show that control of the TRMS **can** be achieved using such an approach. Though the transient response is suboptimal, and the steady state error in the ϕ plane is non-zero, the system settles to an equilibrium state for each of its set-points despite the proximity of some to the edge of the control envelope. Therefore, the viability of the framework to control the TRMS has been demonstrated, and the hypothesis must be deemed well founded.

There is a lot of work to be done before this framework can be formalised into a fully fledged design methodology. However, it is felt that the results shown by this work demonstrates that there is certainly merit in continuing to explore the ideas proposed. What now exists is a solid foundation for future development of a generic multi-agent control framework where none existed before.

8.2 Contributions

This section highlights and discusses the contributions that the work described in this thesis has made.

8.2.1 Introduction of a New MARL Technique

The main contribution that this work has produced is the introduction of a new form of distributed reinforcement learning, that requires minimal definition of goals by the system developer. In the majority of cases, MARL acts to parallelize standard RL techniques to converge on a solution more quickly. Other forms of MARL provide optimisation of distributed problems such as route planning by way of localised optimisation. Whilst these are both valid research fields, they do not easily translate to further the field of multi-agent control of dynamic systems.

Using the proposed MARL techniques provided by the framework, various aspects of the system are learnt, and then combined to produce a coherent strategy. The virtue of this method is that it provides a potential increase of the search space, beyond what can be envisaged by the system designer.

As stated in Section 7.6.3, this is the first instance of a self-organising, self-learning, multi-agent control structure being used on a fast, dynamic, non-linear system such as the TRMS. It has large potential for the development of emergence in numerous problem domains, with some of the possible extensions discussed in Section 8.3.

8.2.2 Cross-Coupled TRMS Characterisation

Examples of TRMS experimentation reported in the literature consistently utilise the 'helicopter configuration' in which there is virtually no cross coupling between the two rotational degrees of freedom. In this work, a cross-coupled configuration has been employed. Furthermore, a detailed analysis of the non-linear nature of the TRMS dynamics has been performed. This helps to illustrate the magnitude of the control problem that the self-organising control, or indeed any other scheme has to address. With the TRMS is in this configuration, control becomes much less straightforward as fan actions must be balanced against each other. Nevertheless, if the fans are orthogonal, all positions within the envelope of operation can be achieved by some combination of fan thrust.

Chapter 3 provides a set of equations that can be used to simulate the TRMS using simple integration routines in its cross-coupled configuration and, with minor alterations, in its helicopter configuration. It also provides a detailed analysis of the TRMS in the cross-coupled configuration to demonstrate the magnitude of the control problem at hand.

8.2.3 Development of a Multi-Agent PID Controller

Whilst Stothert & Macleod (1997) did produce a multi-agent PID controller, their work focussed more on system recovery when agents failed rather than the controller itself. The agents were deployed to the system from agent templates and the system comprised an agent for each of the proportional, integral and differential error term components. There was no real interaction between the agents other than in the combination of the control outputs.

The novelty of the work presented here is its focus on agent action and interaction to achieve successful PID control. The multi-agent PID controller described in Chapter 6 of this thesis displays a much larger degree of interaction between agents. It uses the Contract-net protocol to introduce bartering within the controller so that the best schemes may be selected. Whilst it does not produce a particularly efficient control scheme, it demonstrates that the use of JADE the development of a multi-agent controller is viable regardless of the specific implementation. As such, this work acted as a stepping stone for the development of the self-organising multi-agent framework.

8.3 Recommendations for Further Work

The framework is currently in a form where learning and control can be achieved, but there are a number of aspects that could be investigated to improve its capabilities. There are also new problem domains that could be explored to demonstrate functionality. This section describes recommendations that provide direction for further work.

8.3.1 Learning the β Scaling Factor

It can be seen, in the example runs at Appendix A, that alteration of the β scaling factor has a dramatic effect on the framework's ability to control the TRMS. In these runs, the value for β is global, i.e. every agent must use the same value. There is little doubt that optimisation of this scaling factor for each individual error agent would improve the controller response to a desired position point. It would also allow for the inclusion of a *PhiIntegralErrorAgent* which would eliminate the steady state error in ϕ. However, a method for achieving this whilst adhering to the principles of the framework has not yet been formulated.

A potential avenue for investigation is a secondary learning phase, which could be implemented in different ways. One method would be to investigate the effect of control of a single **error agent** on the others. This would provide a new set of information to the agents, without supplying them with system specific details. Another potential secondary learning phase implementation could remove any **error agents** that integrate the error signal and make integral action a byproduct of a persistent non-zero error. In this way, the integral action would not be generated by an increasing error term (as in standard PID control) but by integration of the controller output itself. Using this method, refining the β value would be more straightforward, as the integral action is linked to a particular error, rather than being a separate error term. One final suggestion for a secondary learning phase uses the notion of global error. If there was a way to provide the **error agents** with an idea of how the system is functioning as a whole, then a secondary learning phase could be implemented using this information. It is difficult to predict how this could be implemented without single points of failure. Nevertheless, it is certainly worthy of investigation.

8.3.2 Investigation into Other RL Techniques

The LRIA provides a simple and effective RL algorithm for action selection. However, the required discretisation of action choices means that any increase in the actions space hinders convergence dramatically. In addition to this, having multiple action choices that produce *good* results is problematic, especially if there is little difference between the action scores. Q-learning offers potential solutions to these problems as actions can be discretised more subtlety so that convergence is based on the best action rather than the most popular. The former has implications for error combination, which is discussed in Section 8.3.3. One potential pitfall of using more complex RL solutions over the LRIA is the added complexity in computation and the memory allocation that is required. Both would have to be considered carefully.

Transfer learning is another technique that offers some interesting possibilities given the distributed nature of the framework and the potential for information sharing. Transfer learning is a blanket term for aiding the learning process based on some information that is already known, be it from a previous run, or from a different, but similar, context. Torrey & Shavlik (2009) describe an example of transfer learning in humans who can "recognize and apply relevant knowledge from previous learning experiences when we encounter new tasks." The use of the iterative action score to weight the learning scaling factor α could be thought of as a form of transfer learning as the information about Λ_{best} is shared around all of the controller agents.

Using some of these ideas, there is potential for a group of observer agents to be implemented which compile information about the learning phase and advise the controller agents on how best to combine or scale actions. It could also be used to draw information about how errors are related to one another.

8.3.3 Error Combination

Another interesting avenue of research afforded by the framework is that of error combination. If multiple agents are learning to reduce errors, where errors have no specific units, then there is no reason why the errors they learn to reduce can not be combinations of multiple errors. There would need to be safeguards against combining errors that were completely contradictory, but with reinforcement learning guiding the error reduction process, a global understanding of how errors are reduced is potentially unnecessary.

One option for such a scheme would be to populate the framework entirely with **error agents** whose inputs are made up of combinations of system inputs. Care would have to be taken that all of the system inputs were represented in the framework. Assuming that the **control agents** have the ability to alter the inputs in a variety of ways, and the combination of errors is not contradictory, there is potential for an RL scheme to be able to control them.

8.3.4 Exploring New Problem Domains

The TRMS is an example of a control problem that can be distributable, but there is scope for using this framework as a design methodology for systems that are more naturally distributed. One potential problem domain that this technique could be applied to is that of distributed communications networks that use resource allocation to maximise throughput of information. There are a number of parameters each node can optimise, such as transmit power, bandwidth, transmit time, among others. These parameters could be thought of as errors to minimise, with error combination also possible.

8.4 Chapter Bibliography

Stothert, A. & Macleod, I. M. (1997), 'Using intelligent agent templates for dynamic structuring of distributed computer control systems', *Engineering Applications of Artificial Intelligence* **10**(4), 335–343.

Torrey, L. & Shavlik, J. (2009), Transfer Learning, *in* 'Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods and Techniques', Information Science Reference, p. 22.

Appendix A

Framework Runs with Alternative Scaling Factors

The following graphs show alternative runs where the global β scaling factor is altered.

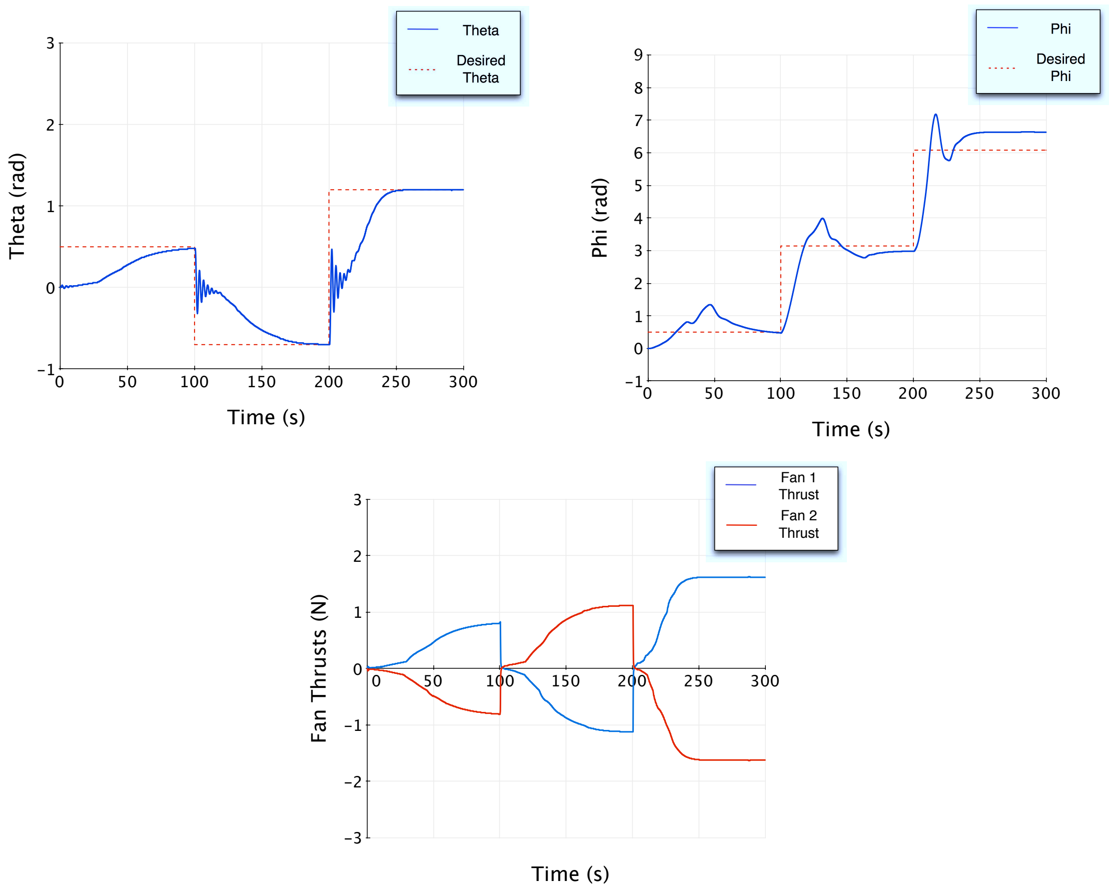

Figure A.1: Framework control of θ and ϕ with the scaling factor at 100

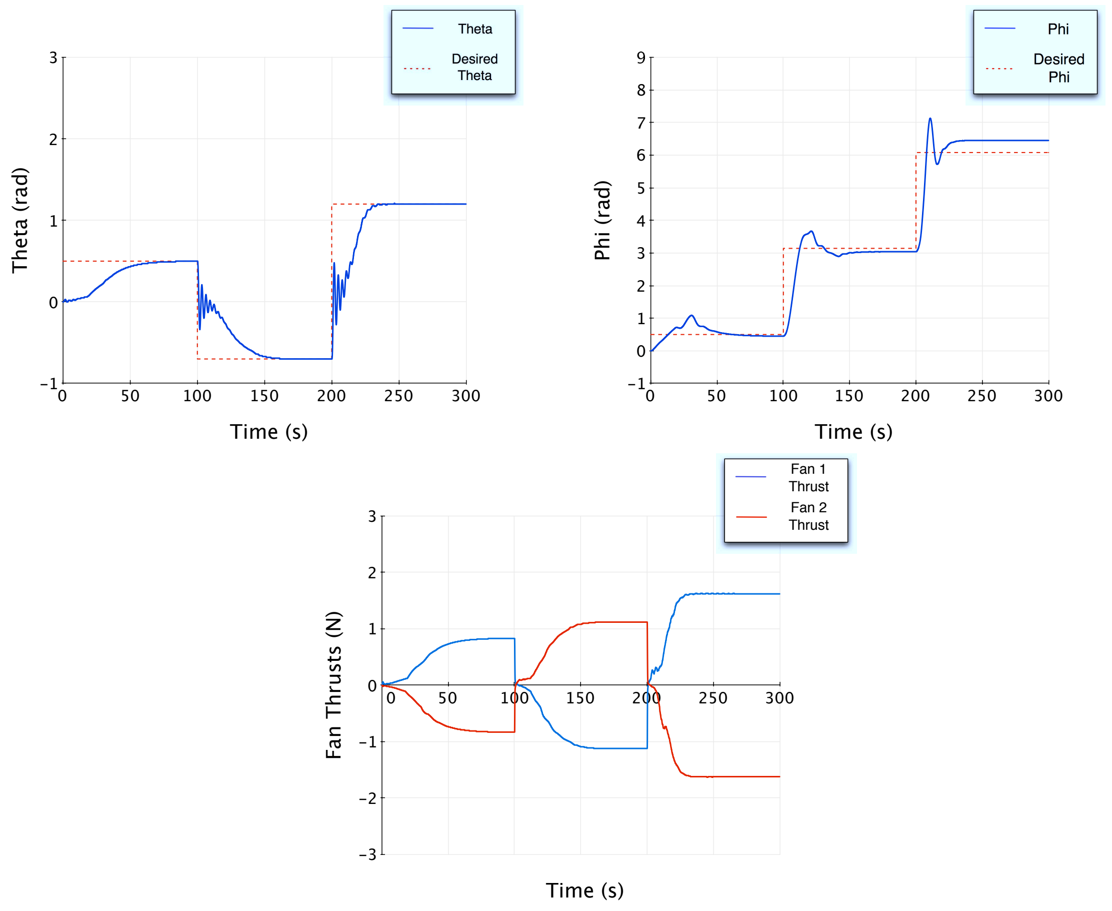

Figure A.2: Framework control of θ and ϕ with the scaling factor at 150

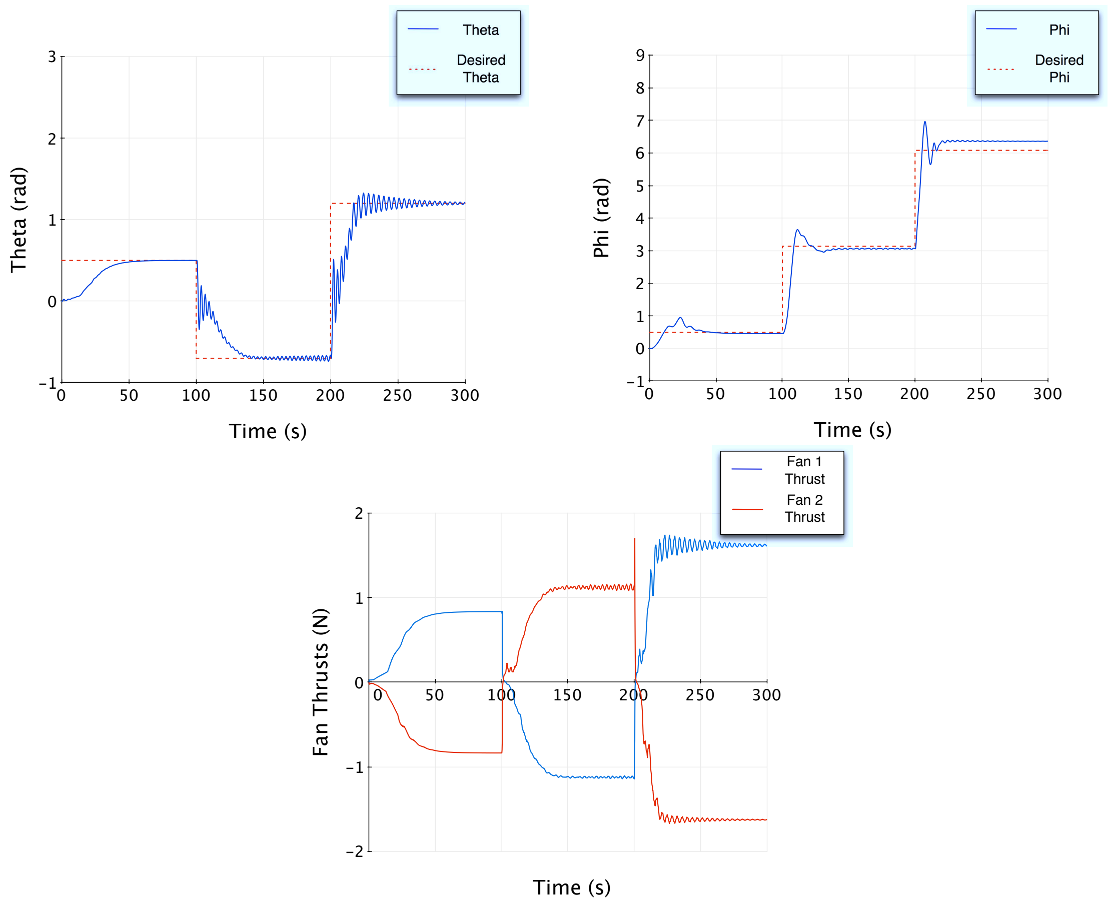

Figure A.3: Framework control of θ and ϕ with the scaling factor at 200

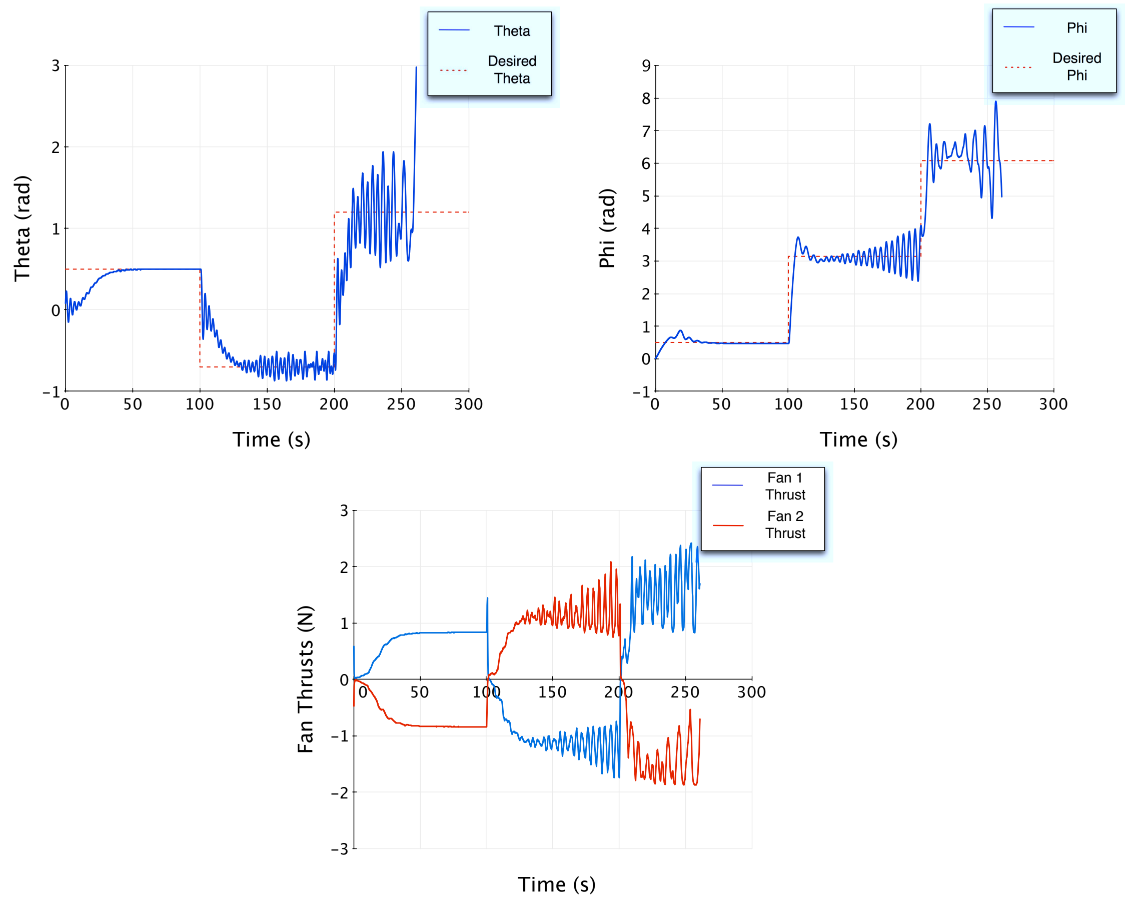

Figure A.4: Framework control of θ and ϕ with the scaling factor at 250

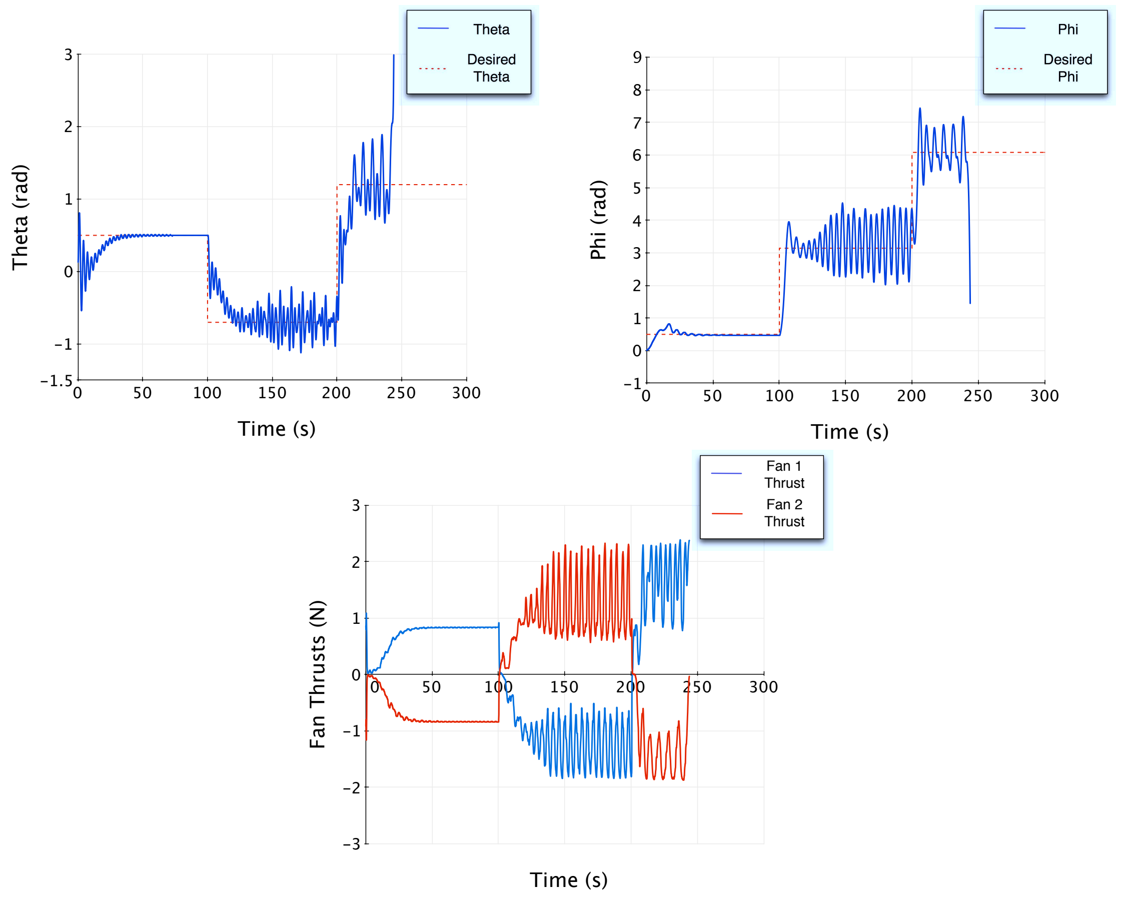

Figure A.5: Framework control of θ and ϕ with the scaling factor at 300

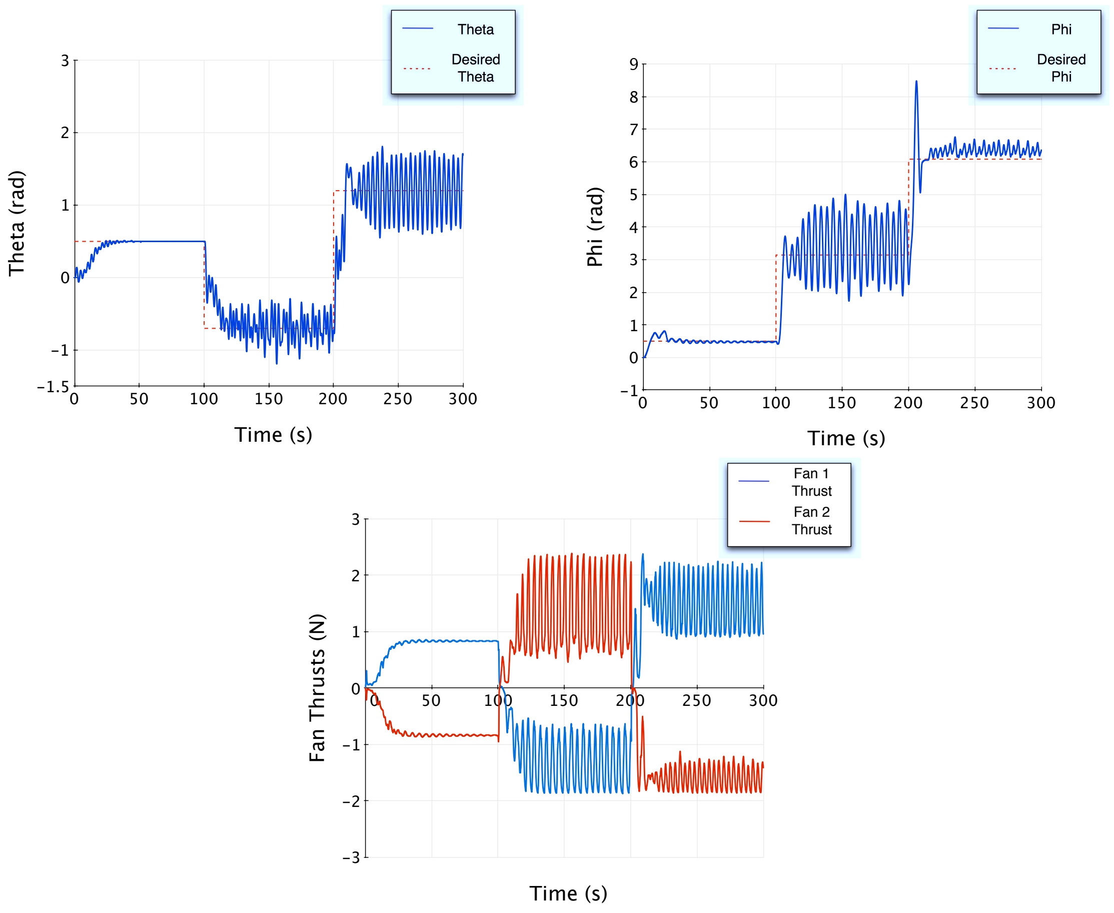

Figure A.6: Framework control of θ and ϕ with the scaling factor at 350

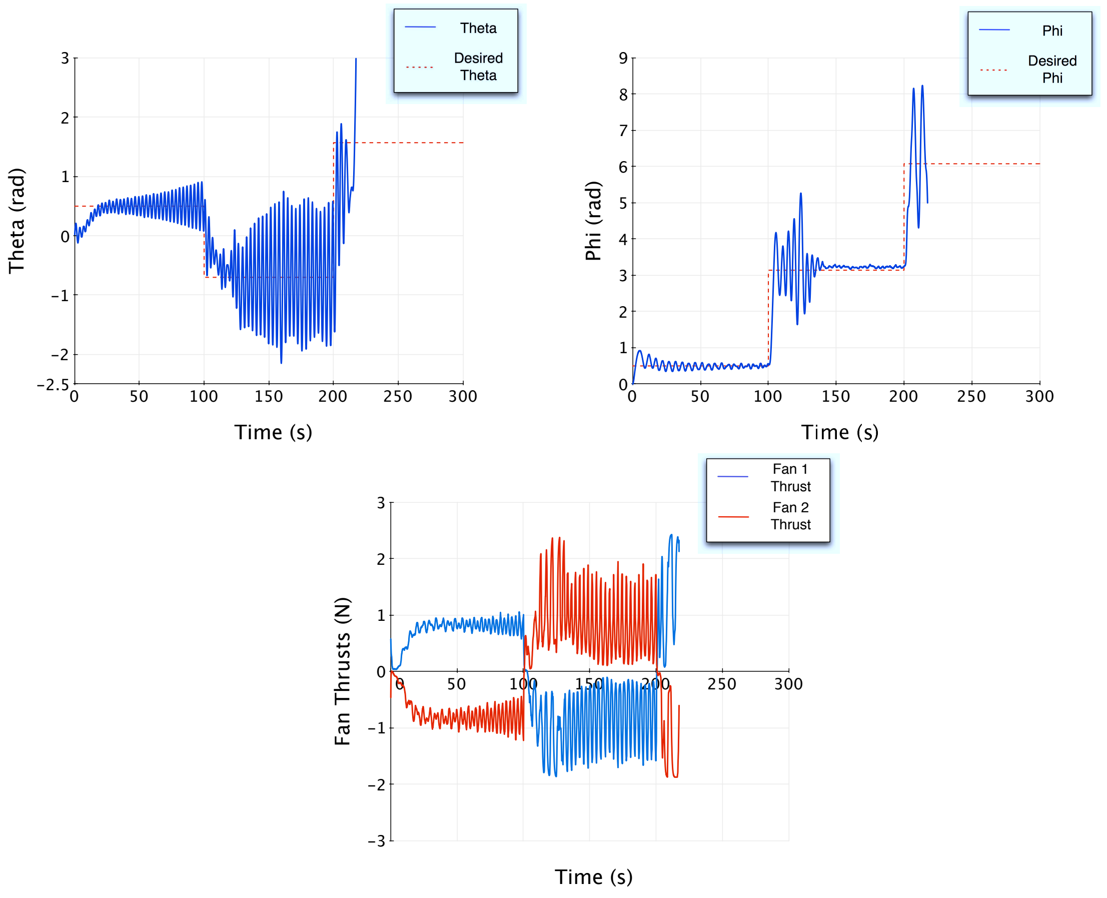

Figure A.7: Framework control of θ and ϕ with the scaling factor at 400

Appendix B

Fuzzy Membership Functions for the TRMS Fuzzy Controller

The following shows the fuzzy rule sets used in the fuzzy controller described in Section 3.5.3 of Chapter 3.

TRMS θ angle error									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VLP	HP	MP	SP	Zero	SN	MN	LN	VLN

Table B.1: Fuzzy membership rule function for θ angular position error of the TRMS with a weighting of 0.8

TRMS ϕ angle error									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP

Table B.2: Fuzzy membership rule function for ϕ angular position error of the TRMS with a weighting of 1.0

TRMS θ angle error integral									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VLP	HP	MP	SP	Zero	SN	MN	LN	VLN

Table B.3: Fuzzy membership rule function for the integral of the θ angular position error of the TRMS with a weighting of 1.0

TRMS ϕ angle error integral									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP

Table B.4: Fuzzy membership rule function for the integral of the ϕ angular position error of the TRMS with a weighting of 0.1

TRMS θ angle differential									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VLP	HP	MP	SP	Zero	SN	MN	LN	VLN

Table B.5: Fuzzy membership rule function for the θ angular velocity error of the TRMS with a weighting of 1.0

TRMS ϕ angle differential									
Membership Function	VLN	LN	MN	SN	Zero	SP	MN	LP	VLP
Fan 1 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP
Fan 2 Rule	VHN	HN	MN	SN	Zero	SP	MP	LP	VLP

Table B.6: Fuzzy membership rule function for the integral of the ϕ angular velocity error of the TRMS with a weighting of 0.8

Nomenclature

α	The probability update factor for the LRIA
β	The scaling factor for suggested fan input values to the TRMS
Λ	The value for the iterative action score after an action has been performed
$\mathcal{L}$	The Lagrangian as defined as the potential energy subtracted from the kinetic energy $\mathcal{T} - \mathcal{U}$ in a system
$\mathcal{T}$	The kinetic energy in a system
$\mathcal{U}$	The potential energy in a system
F_1	The thrust produced by Fan 1 of the TRMS
F_2	The thrust produced by Fan 2 of the TRMS
$F_{\phi 1}$	The thrust produced by Fan 1 in the ϕ plane of the TRMS
$F_{\phi 2}$	The thrust produced by Fan 2 in the ϕ plane of the TRMS
$F_{\theta 1}$	The thrust produced by Fan 1 in the θ plane of the TRMS
$F_{\theta 2}$	The thrust produced by Fan 2 in the θ plane of the TRMS
L_1	The distance between the pivot point in θ and Fan 1 on the TRMS
L_2	The distance between the pivot point in θ and Fan 2 on the TRMS
L_3	The distance between the pivot point in θ and the pendulum bob on the TRMS
L_4	The distance between the pivot point in θ and the the pivot point in ϕ on the TRMS
M_1	The mass of Fan 1 on the TRMS
M_2	The mass of Fan 2 on the TRMS

M_3	The mass of the pendulum bob on the TRMS
t	Time

Glossary of Terms

AMS	Agent Management System — Responsible for the management of all agents in a Foundation for Intelligent Physical Agents (FIPA) compliant multi-agent system.
AOP	Agent Oriented Programming — An extension to Object Oriented Programming (OOP) Agent Oriented Programming (AOP) is a structure for developing multi-agent systems.
AP	Agent Platform — A framework to house any agents controlled by a single Agent Management System (AMS). Multi-agent systems can span multiple machines each of which would have its own Agent Platform (AP).
CFP	Call For Proposals — A message sent out to a group of agents from an initiator agent to start the contract net protocol.
FIPA	Foundation for Intelligent Physical Agents — A committee that produces standards for the design and implementation of multi-agent systems.
LRIA	Linear Reward Inaction Algorithm — A probabilistic reinforcement learning algorithm that increases the probability of an action being selected if it yields a positive result and decreases the probability of all other actions.
MARL	Multi-Agent Reinforcement Learning — The field of research dedicated to distributing reinforcement learning techniques using multi-agent methodologies.
OOP	Object Oriented Programming — A programming paradigm where programs are built up through interaction between object which contain their own information, methods and properties.

PID	Proportional plus Integral plus Derivative — A simple yet effective control scheme used across a wide range of control engineering problems that feeds different proportions of the direct, integral and differential of the error signal.
PSO	Particle Swarm Optimisation — A distributed parameter space searching algorithm using swarms of simple software entities that explore and evaluate the fitness of the landscape at their current location.
PWM	Pulse Width Modulation — A modulation scheme for varying the input to an inertial electronic components by varying the width of pulses at the input. The width of the pulses determines the average power that is used to drive the component.
RL	Reinforcement Learning — A field of machine learning research devoted to an agent learning actions to employ for given states.
SCADA	Supervisory Control and Data Acquisition — A system used for gathering input data and controlling the output to the plant. It is often used in industrial process control.
TD	Temporal Difference — A form of reinforcement learning algorithm that estimates long term rewards rather than simply immediate rewards.
TRMS	Twin Rotor MIMO System — A pendulum system with two rotational degrees of freedom controlled by two fans that are tilted to produce thrust in both the θ and ϕ planes.
TSP	Travelling Salesman Problem — A commonly used problem for algorithm testing which, given a set of nodes and the distances between them, demands the shortest route where all nodes are visited at least once.

Bibliography

Abbott, R. (2006), ‘Emergence explained: Abstractions: Getting epiphenomena to do real work’, *Complexity* **12**(1), 13–26.

Anderson, C. W. (1989), ‘Learning to Control an Inverted Pendulum using Neural Networks’, *Control Systems Magazine* **9**(3), 31–37.

Antsaklis, P. J. (1994), Defining intelligent control, *in* ‘Report of the Task Force on Intelligent Control, P.J Antsaklis, Chair’, John Wiley & Sons, Inc.

Avižienis, A. (1975), Fault-tolerance and fault-intolerance, *in* ‘Proceedings of the International Conference on Reliable Software’, ACM.

Bailey-McEwan, M. (1991), ‘Use of the chiller computer program with conventional water chilling installations on south african gold mines’, *Journal of the Mine Ventilation Society of South Africa* **44**(1), 2–21.

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983), ‘Neuronlike adaptive elements that can solve difficult learning control problems’, *IEEE Transactions on Systems, Man, and Cybernetics* **13**(5), 834–846.

Bellifemine, F. L., Caire, G. & Greenwood, D. (2007), *Developing Multi-agent Systems with JADE (Wiley Series in Agent Technology)*, 1st edn, Wiley-Blackwell.

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press.

Bond, A. H. & Gasser, L. (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kauffman Publishers.

Bower, G. H. & Hilgard, E. R. (1980), *Theories of Learning (The Century psychology series)*, Pearson Education.

Bratman, M. E. (1987), *Intention, Plans, and Practical Reasoning*, Harvard University Press.

Bratman, M. E. (1990), What is intention?, *in* 'Intentions In Communication', MIT Press, pp. 15–32.

Bratman, M. E., Israel, D. J. & Pollack, M. E. (1988), 'Plans and resource-bounded practical reasoning', *Computational Intelligence* **4**(3), 349–355.

Brooks, R. A. (1986), 'A robust layered control system for a mobile robot', *IEEE Journal of Robotics and Automation* **2**(1), 14–23.

Brooks, R. A. (1991), 'Intelligence Without Representation', *Artificial Intelligence* **47**(1-3), 139–159.

Busoniu, L., Babuska, R. & De Schutter, B. (2008), 'A comprehensive survey of multiagent reinforcement learning', *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews* **38**(2), 156–172.

Canonical Ltd. (2009), *Ubuntu Version 10.04.3*, London, UK.

Colorni, A., Dorigo, M. & Maniezzo, V. (1991), Distributed optimization by ant colonies, *in* 'European Conference on Artificial Life', Elsevier.

De Wolf, T. & Holvoet, T. (2003), Towards autonomic computing: Agent-based modelling, dynamical systems analysis, and decentralised control, *in* 'IEEE International Conference on Industrial Informatics', IEEE.

Dorf, R. C. & Bishop, R. H. (2008), *Modern Control Systems*, 12th edn, Pearson Prentice Hall India.

Dorigo, M. & Gambardella, L. M. (1997), 'Ant colony system: A cooperative learning approach to the traveling salesman problem', *IEEE Transactions on Evolutionary Computation* **1**(1), 53–66.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996), 'The Ant System: Optimization by a Colony of Cooperating Agents', *IEEE Transactions on Systems Man and Cybernetics Part B Cybernetics* **26**(1), 29–41.

Doya, K. (2000), 'Reinforcement learning in continuous time and space', *Neural Computation* **12**(1), 219–245.

Driankov, D., Hellendoorn, H. & Reinfrank, M. (1993), *An Introduction to Fuzzy Control*, Springer-Verlag.

Eberhart, R. & Shi, Y. (2001), Particle swarm optimization: developments, applications and resources, *in* ‘Proceedings of the 2001 Congress on Evolutionary Computation’, IEEE.

Facebook (2004), ‘Facebook’.
URL: *http://www.facebook.com*

Feedback Instruments Ltd. (2002), *Twin Rotor Mimo System: Advanced Teaching Manual 1*, Crowborough, UK.

Ferguson, I. A. (1992), TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents, PhD thesis, Clare Hall, University of Cambridge, UK.

FIPA (2002*a*), ‘FIPA ACL Message Structure Specification’.
URL: *http://www.fipa.org/specs/fipa00070/*

FIPA (2002*b*), ‘FIPA Agent Management Specification’.
URL: *http://www.fipa.org/specs/fipa00023/*

FIPA (2002*c*), ‘FIPA Agent Message Transport Service Specification’.
URL: *http://www.fipa.org/specs/fipa00084/*

FIPA (2002*d*), ‘FIPA Contract Net Interaction Protocol Specification’.
URL: *http://www.fipa.org/specs/fipa00029/*

Fodor, J. (1997), ‘Special Sciences: Still Autonomous after All these Years’, *Noûs* **31**(s11), 149–163.

Friedman, M. (2006), Afterward, *in* ‘I, Pencil: My Family Tree as told to Leonard E. Read’, Foundation for Economic Education, p. 18.

Gaing, Z.-L. (2004), ‘A Particle Swarm Optimization Approach for Optimum Design of PID Controller in AVR System’, *IEEE Transactions on Energy Conversion* **19**(2), 384–391.

Gribble, S. (2001), Robustness in Complex Systems, *in* ‘Proceedings Eighth Workshop on Hot Topics in Operating Systems’, IEEE Computer Society.

Gross, S. D., Stephan, V., Debes, K., Gross, H. m., Wintrich, F. & Wintrich, H. (2000), A Reinforcement Learning based Neural Multi-AgentSystem for Control of a Combustion Process, *in* ‘IEEE-INNS-ENNS International Joint Conference of Neural Networks’, IEEE Computer Society, pp. 217–222.

Hu, X. & Eberhart, R. (2002), Multiobjective optimization using dynamic neighborhood particle swarm optimization, *in* 'Proceedings of the 2002 Congress on Evolutionary Computation CEC02', IEEE Computer Society, pp. 1677–1681.

Iglesias, C. A., Garijo, M. & Centeno-González, J. (1998), A Survey of Agent-Oriented Methodologies, *in* 'Proceedings of the 5th International Workshop on Intelligent Agents', Springer-Verlag, pp. 317–330.

Jennings, N. R. (2000), 'On Agent-Based Software Engineering', *Artificial Intelligence* **117**(2), 277–296.

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), 'Reinforcement Learning: A Survey', *Journal of Artificial Intelligence Research* **4**(1), 237–285.

Kennedy, J. & Eberhart, R. (1995), Particle Swarm Optimization, *in* 'IEEE International Conference on Neural Networks', IEEE, pp. 1942–1948.

Kobori, N., Suzuki, K., Hartono, P. & Hashimoto, S. (2002), Learning to Control a Joint Driven Double Inverted Pendulum using Nested Actor/Critic Algorithm , *in* 'Proceedings of the 9th International Conference on Neural Information Processing', pp. 2610–2614.

Liu, C.-s., Chen, L.-r., Li, B.-z., Chen, S.-k. & Zeng, Z.-s. (2006), Improvement of the Twin Rotor MIMO System Tracking and Transient Response Using Fuzzy Control Technology, *in* '1st IEEE Conference on Industrial Electronics and Applications', IEEE, pp. 1–6.

MacLeod, I. & Stothert, A. (1998), 'Distributed Intelligent Control for a Mine Refrigeration System', *IEEE Control Systems Magazine* **18**(2), 31–38.

Magana, M. E. & Holzapfel, F. (1998), 'Fuzzy-logic control of an inverted pendulum with vision feedback', *IEEE Transactions on Education* **41**(2), 165–170.

Mathworks Inc. (2011*a*), *MATLAB Version 7.12.0*, Natick, MA.

Mathworks Inc. (2011*b*), *Simulink Version 7.7*, Natick, MA.

McCarthy, J. (1979), Ascribing mental qualities to machines, *in* 'In Philosophical Perspectives in Artificial Intelligence', Humanities Press.

Mendham, P. D. (2006), Multi-Agent Control for the Skylon Spaceplane, PhD thesis, University of York.

Mendham, P. D. & Clarke, T. (2003*a*), Dependable intelligent control through the use of multiple intelligent agents, *in* 'Proceedings of the 16th International Conference on Systems Engineering, ICSE2003', IEEE, pp. 478–483.

Mendham, P. D. & Clarke, T. (2003*b*), Growing dependability using a multi-agent approach to fault tolerance, *in* 'Proceedings of the 54th Congress of the International Astronautical Federation', IEEE.

Mendham, P. D. & Clarke, T. (2005*a*), MACSim: A simulink-enabled environment for multiagent simulation, *in* 'Proceedings of the 16th IFAC World Congress', IEEE.

Mendham, P. D. & Clarke, T. (2005*b*), On acheiving true autonomy: Using multi-agent control for assto spaceplane, *in* 'Proceedings of Data Systems in Aerospace', IEEE.

Mendham, P. D., Pomfret, A. & Clarke, T. (2004), Dependable dynamic control using distributed intelligent agents, *in* 'Proceedings of the 55th Congress of the International Astronautical Federation', IEEE.

Meyer, B. (1997), *Object-Oriented Software Construction*, 2nd edn, Prentice Hall.

MIT Center for Collective Intelligence (2006), 'MIT Center for Collective Intelligence'.
URL: *http://cci.mit.edu/index.html*

Müller, J. P., Pischel, M. & Thiel, M. (1995), Modelling Reactive Behaviour in Vertically Layered Agent Architectures, *in* 'Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents', Springer-Verlag, pp. 261–276.

Nise, N. S. (2010), *Control Systems Engineering*, 6th edn, John Wiley & Sons.

Ogata, K. (2001), *Modern Control Engineering*, 2nd edn, Prentice Hall.

Oracle Corporation (2011), *Java 3D Version 1.5.2*, Redwood Shores, CA.

Parrott, D. (2006), 'Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model using Speciation', *IEEE Transactions on Evolutionary Computation* **10**(4), 440–458.

Pay, M. (2008), Kill the Wildebeest, Master's thesis, University of York.

Poli, R. (2008), 'Analysis of the Publications on the Applications of Particle Swarm Optimisation', *Journal of Artificial Evolution and Applications* **2008**(2), 1–11.

Price, B. & Boutilier, C. (2003), 'Accelerating Reinforcement Learning through Implicit Imitation', *Journal of Artificial Intelligence Research* **19**(1), 569–629.

Puterman, M. L. & Shin, M. (1978), 'Modified Policy Iteration Algorithms for Discounted Markov Decision Problems', *Management Science* **24**(11), 1127–1137.

Rahideh, A. & Shaheed, M. (2006), Hybrid Fuzzy-PID-based Control of a Twin Rotor MIMO System, *in* '32nd Annual Conference on IEEE Industrial Electronics', IEEE, pp. 48–53.

Rao, A. S. & Georgeff, M. P. (1995), BDI Agents: From Theory to Practice, *in* 'Proceedings of the first international conference on multiagent systems ICMAS95', pp. 312–319.

Reynolds, C. W. (1987), 'Flocks, Herds and Schools: A distributed behavioral model', *SIGGRAPH Computer Graphics* **21**(4), 25–34.

Searle, J. (1969), *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press.

Shoham, Y. (1993), 'Agent-oriented programming', *Artificial Intelligence* **60**(1), 51–92.

Smith, A. (1776), *An Inquiry into the Nature and Causes of the Wealth of Nations*, Liberty Fund.

Steels, L. (1990), Cooperation between distributed agents through self-organisation, *in* 'IEEE International Workshop on Intelligent Robots and Systems', pp. 8–14.

Stepney, S., Polack, F. & Turner, H. (2006), Engineering emergence, *in* '11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06)', pp. 89–97.

Stothert, A. & Macleod, I. M. (1997), 'Using intelligent agent templates for dynamic structuring of distributed computer control systems', *Engineering Applications of Artificial Intelligence* **10**(4), 335–343.

Sutton, R. S. (1988), 'Learning to Predict by the Methods of Temporal Differences', *Machine Learning* **3**(1), 9–44.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, The MIT Press.

Tan, M. (1993), Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, *in* 'Proceedings of the tenth international conference on machine learning', pp. 330–337.

Torrey, L. & Shavlik, J. (2009), Transfer Learning, *in* 'Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods and Techniques', Information Science Reference, p. 22.

Twitter (2006), 'Twitter'.
URL: *http://www.twitter.com*

Ul Islam, B., Ahmed, N., Bhatti, D. & Khan, S. (2003), Controller design using fuzzy logic for a twin rotor MIMO system, *in* '7th International Multi Topic Conference', pp. 264–268.

van der Walt, J. & Whillier, A. (1978), 'Considerations in the Design of Integrated Systems for Distributing Refrigeration in Deep Mines', *Journal of the Mine Ventilation Society of South Africa* **31**(12), 217–243.

Veres, S. M. & Luo, J. (2004), A Class of BDI Agent Architectures for Autonomous Control, *in* '43rd IEEE Conference on Decision and Control', pp. 4746–4751 Vol.5.

Voos, H. (1999*a*), Market-based Algorithms for Optimal Decentralized Control of Complex Dynamic Systems, *in* 'Proceedings of the 38th IEEE Conference on Decision and Control', pp. 3295–3296.

Voos, H. (1999*b*), Market-based control of complex dynamic systems, *in* 'International Symposium on Intelligent Control Intelligent Systems and Semiotics', pp. 284–289.

Voos, H. & Litz, L. (2000), Market-based optimal control: a general introduction, *in* 'Proceedings of the 2000 American Control Conference', pp. 3398–3402 vol.5.

Watkins, C. J. C. H. (1989), Learning from Delayed Rewards, PhD thesis, Kings College, Cambridge, UK.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Technical Note: Q-Learning', *Machine Learning* **8**(3), 279–292.

Wen, P. & Lu, T.-W. (2008), 'Decoupling Control of a Twin Rotor MIMO System using Robust Deadbeat Control Technique', *IET Control Theory & Applications* **2**(11), 999 – 1007.

Wiering, M. (2000), Multi-Agent Reinforcement Learning for Traffic Light Control, *in* 'Proceedings of the 25th International Conference on Machine learning', pp. 1151–1158.

Wikipedia Foundation (2001), ‘Wikipedia’.
URL: *http://www.wikipedia.org*

Wolfram, S. (2002), *A New Kind of Science*, Wolfram Media Inc.

Wooldridge, M. (2009), *An Introduction to MultiAgent Systems*, 2nd edn, John Wiley & Sons.

Young, S. C. & Krishnapuram, R. (1997), ‘A robust approach to image enhancement based on fuzzy logic’, *IEEE Transactions on Image Processing* **6**(6), 805–825.

Zadeh, L. A. (1965), ‘Fuzzy sets’, *Information and Control* **8**(3), 338–353.

Ziegler, J. G. & Nichols, N. B. (1993), ‘Optimum Settings for Automatic Controllers’, *Journal of Dynamic Systems Measurement and Control* **115**(2B), 220.