University of Sheffield

Can Control Hierarchies be Developed and Optimised Progressively?

Benjamin James Hawker

Supervisor: Roger K. Moore

A report submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

in the

Department of Computer Science

May 17, 2021

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

Name: _____

Signature: _____

Date: _____

"It has become customary, in contemporary cognitive science, to blame Descartes for almost everything" [Shanahan, 2010]

Abstract

Hierarchical structures are used in robots to achieve effective results in control problems. Hierarchical structures are found in a wide array of applications of AI and robotics, making them a key aspect of control. Even though they hold an integral part in control, such structures are typically produced heuristically, resulting in inconsistent performance. This means that effective control tasks or controllers perform poorly due to the hierarchy being badly defined, limiting what controllers can do. Complex control problems that require adaptive behaviour or autonomy remain challenging for control theorists, with complex problem domains making the heuristic process of producing complex hierarchies harder.

It is evident that the heuristic process must have some form of procedure that could be turned into a methodology. By formalising or automating this process, control hierarchies can be produced with consistently effective results without relying on the heuristic production of a control engineer which can easily fail. This thesis proposes an algorithmic approach (inspired by Perceptual Control Theory) known as Dependency-Oriented Structure Architect (DOSA). DOSA produces heirarchies automatically using real world experience and the inputs the system has access to. This thesis shows that DOSA consistently reproduces effective hierarchies that exist in the literature, when billions of possible hierarchies were available.

Furthermore, this thesis investigates the value of using hierarchies in general and their benefits in control problems. The computational complexity of hierarchies is compared, showing that while hierarchies do not have a computational advantage, the parameter optimisation procedure is aided greatly by hierarchical parameter optimisation. The thesis then proceeds to study th hierarchical optimisation of parameters and how hierarchies allow this process to be performed more consistently for better results, concluding that hierarchical parameter optimisation produces more consistent controllers that also transfer better to an unseen problem domain. Parameter optimisation is a challenge that also limits otherwise effective controllers and limits the use of larger structures in control.

The research described in this thesis formalises the process of generating hierarchical controllers as well as hierarchically optimising them, providing a comprehensive methodology to automate the production of robust controllers for complex problems.

Lianne, you were always leaps and bounds ahead of me. You made goals and dreams happen that I couldn't have found in myself the strength and will to do. Intelligence and compassion of your stature rarely meet and blessed I was to have you in my life.

Sleep tight, Subalary.

Contents

List of Acronyms

AI Artificial Intelligence

DOSA Dependency-Oriented Structure Architect

PID Proportional-Integral-Derivative

RCS Real-time Control System

RMA Reference Model Architecture

STRIPS Stanford Research Institute Problem Solver

BG Basal Ganglia

MRF Medial Reticular Formation

DPMs Distributed Processing Modules

TD Temporal Difference

DNN Deep Neural Network

HAMs Hierarchical Abstract Machines

ABM Adaptive Behaviour Module

DACS Distributed Adaptive Control System

DAC Distributed Adaptive Control

MBBN Mind Brain Body Nexus

GW Global Workspace

LIDA Learning Intelligent Distribution Agent

PCT Perceptual Control Theory

HPCT Hierarchical Perceptual Control Theory

LCS Living Control Systems

1 Introduction

A Control System enacts control over some variable in their environment. This environment could be a virtual environment, or the real world which the robot changes with actuators. By controlling the environment they work in, they aim to achieve a desired outcome for the variable they control, which is known as the controlled variable. Control systems are common in many applications of robotics, where robotic agents must control the world around them to achieve goals.

Control systems can be broken down into separate control processes, where their interactions between each other perform some control task greater than the individual control any one subsystem processes. Control hierarchies are an arrangement of control subsystems where control subsystems delegate to one-another as a means of controlling their objectives. Hierarchies are used extensively in programming, AI and control theory, to allow complex problems to be broken down into manageable concepts, domains and functions.

The introduction of hierarchies to control has produced great improvements in capability and robustness, including deep learning [Graves et al., 2009] to older developments such as the Subsumption Architecture [Brooks, 1986]. Hierarchies play an important role in Control Theory, enabling learning and control to be spread over smaller parts of a larger structure. This supports maintaining a larger structure as it's clear what each individual function does. The application of hierarchies, however, is inconsistent.

Control hierarchies have been used since control theory and AI emerged, with structures being considered a way of managing more complex problems. Robots were produced with hierarchical controllers, boasting powerful self-regulation and autonomy, such as SHAKEY the robot [Cocosco, 1998]. However, efforts to generalise the robots to new environments failed. Additionally, as hierarchies grew, they became increasingly difficult to balance and maintain. Subsumption Architecture is a notable example where the hierarchy was difficult to implement after the first few levels [Brooks, 1986]. It remains an open question why this occurred and what was lacking in the proposed architecture.

A contrasting example is the rise of deep learning, which has seen recent successes in strategy games [Mnih et al., 2013], robotics and locomotion [Levine et al., 2015] and grasping objects [Lenz et al., 2015]. The hierarchies here are clearly robust, but exactly what the hierarchy is or how it is performing its task is unknown. Large amounts of training on data produce hierarchies embedded in layers of neural networks, but the 'black box' approach yields a lack of understanding on both the successful and unsuccessful results. Clear pitfalls in the choices of deep learning agents have been identified [Nguyen et al., 2014], but the lack of an understanding of the hierarchy or system means that conclusions cannot be drawn on the failures of these systems.

Biological systems have inspired new structures to be used in control theory, such as models imitating the structure of the Cerebellum or Basal Ganglia. This includes deep learning which originates from studies of groups of neurons, but also the Actor-Critic variant of Reinforcement Learning [Joel et al., 2002]. Furthermore, the fields of cognitive science and psychology detail approaches to control that focus on the perceptions of the agent. PCT, devised by William Powers, details how complex hierarchies can perform complicated control tasks and high level cognitive function [Powers, 1974]. It is clear that many resources detailing hierarchies and how they occur in natural agents have not been fully utilised in control theory.

This poses a wider question on what makes a hierarchy and what the value of a hierarchy is in the place of control. Hierarchies have been shown to produce good results

when built correctly, but the literature does not indicate what a well-built hierarchy is. Implementations of hierarchies are common and often include details of their implementation, but the semantic understanding of why the hierarchy was built as it was is lacking in many descriptions of the implementation. This information is clearly critical to the effective building of a hierarchy and unlocking strong and robust results, but remains unstudied as to how to produce a hierarchy.

1.1 Research Objectives

The emergence of hierarchical control structures fails to be the focus of research despite being a core element of Control Theory. Researchers and control engineers spend extensive hours constructing hierarchies for Artificial Intelligence (AI)s or control structures, with no formal process existing which a control engineer can follow. Allowing agents to learn these hierarchies automatically would give robust and consistent results while also consuming less manual time and effort of a control expert. This also allows the opportunity to perform comparative studies, evaluating the benefits of using hierarchies compared with other approaches. This justifies an approach common to control engineers while also enabling it with automation.

This Thesis explores the following questions:

- What is a hierarchy?
- Can the process of deriving a hierarchy be automated?
- Can the process of deriving control parameters in a hierarchy be automated?
- What are the benefits to using a hierarchical approach?

The main contributions of this Thesis are: a methodology for automatically deriving hierarchies (DOSA), hierarchical parameter optimisation is effective, and that hierarchical parameter optimisation produces better results on unseen situations in robots.

1.2 Structure of the Thesis

Section 2 covers an overview of Classic Control Systems, detailing existing hierarchical approaches including their strengths and weaknesses. This also includes fundamentals of Control Theory.

Section 3 covers control systems which are inspired by biological agents, covering control systems from neurology and cognitive neuroscience (Subsections 3.1 to 3.5) and control systems inspired by psychology (Subsections 3.6 to 3.9).

Section 4 outlines the principles of a hierarchical relationship after examining the literature and uses this to produce DOSA, a methodology for automatically deriving hierarchies. DOSA is defined algorithmically in Subsection 3.4, which is an expanded version of a paper published as proceedings from UKRAS20 [Hawker and Moore, 2020]. Subsection 3.6 demonstrates DOSA on an existing hierarchy in the literature to show the similarity of DOSA's derived hierarchy the the effective manually derived hierarchy.

Section 5 conducts experiments to evaluate the effectiveness of hierarchies and hierarchical parameter optimisation, as used in Step 3 of DOSA. Section 5.1 provides a comparison of computational complexity of hierarchical and non-hierarchical approaches, showing no significant difference. Section 5.2 outlines how a curriculum for hierarchical

learning can be produced from a hierarchy DOSA generates. This is then used in Section 5.3, which concludes that parameter optimisation using hierarchies provides more consistent results. These results were published in TAROS 2020 [Hawker and K Moore, 2020]. Section 5.4 extends this work, showing that hierarchical parameter optimisation also provides parameters that produce better control on an unseen test set after training, which is currently under review.

Section 6 outlines the contributions of this thesis, revisiting the research objectives mentioned in Section 1.1. Section 7 concludes five directions for future work and concludes the thesis, providing final remarks.

2 Classic Control Systems

2.1 Control Theory

Control Theory encompasses the many engineering approaches used to act on and control the world. A control system is manufactured to command and direct a specific real world problem. This Section outlines the fundamentals of Control, which are used extensively in controllers for robots.

Within Control Theory, any incoming signal to the control system is considered an input. The output of a system is the response obtained from the control system which is sent to devices that can interact with the real world, enacting some change.

Furthermore, an important distinction in control systems is whether they are open-loop or closed-loop:

> *"An **open-loop** control system is one in which the control action is independent of the output.*
> *A **closed-loop** control system is one in which the control action is somehow dependent on the output"*
> *— Schaum's Outlines; Feedback and Control Systems [DiStefano et al., 1967][p. 3]*

A typical example of an open-loop control system would be a toaster, whereas a closed-loop control system would be a thermostat. A toaster is open-loop because the output has no effect on the input to the system. The input is the dial indicating how long the toaster needs to be active and the output is the heat applied to the conductors which heat the toast. No amount of heat applied changes where the dial is, meaning there is no connection between them. However, a thermostat's controller directly affects future input signals. The inputs to the system are the actual and desired temperature of the room. The system output is activating heating elements around the room, such as radiators. The radiators influence the temperature of the room, meaning the output of the system has a direct effect on the input.

2.1.1 Classic Control Theory

Classic Control Theory is a subset of Control Theory that concerns solutions to linear time-invariant systems. The introduction of feedback as a concept in Control Theory originated in classic Control Theory. The classic negative feedback closed loop controller is shown in Figure 1.

A typical negative feedback loop within classic Control Theory has several properties. The control system has two stimuli, the **reference (r)** and the **input (i)**. The input signal represents some external reading about the world and the reference signal is a representation of the desired value for the input signal. So, if the input matches the reference, the plant has achieved its goal.

The reference and input are processed at a **summing point**, which outputs the algebraic sum of all inputs. Some inputs are marked with a '-' sign, meaning the input signal is subtracted from the reference signal. This can also be denoted with a circle with a cross in the centre.

The **controller** is some function which takes the error e and returns the output signal u to the plant. It can be as simple as algebraic functions, or a more complex

transform. The controller may have parameters which change the transform it performs on e to produce u. A common transform applied after u is received and before it is sent to an actuator is a **gain**. A gain is a parameter which u is multiplied by to scale it appropriately for the actuator it is being sent to. The **Plant** is the device which has some real world interaction in order to control the external world and receives u after it has been multiplied by a gain, if applicable. The disturbance, d, marks the unexpected changes to the effect of the output by the outside world.

Finally, there is the **feedback element**, a device which takes a reading from the external world to provide an input for the system. In a closed loop system, this input is directly affected by the output to the plant, hence the looped connection in the diagram. The feedback element produces the input signal for the control system, which completes the circuit.

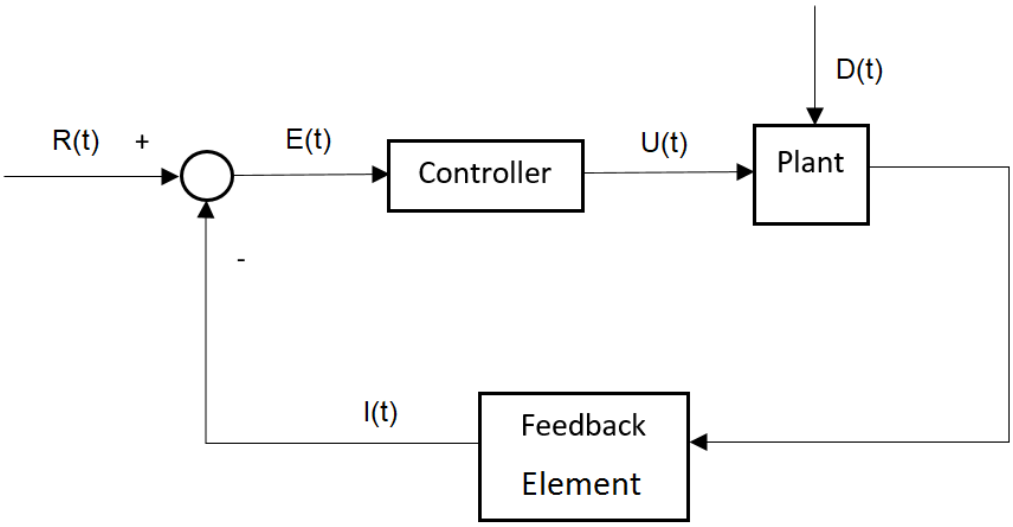

Figure 1: *A Classic Control Theory controller. The feedback element provides some signal measuring the real world, marked as input I(t). The value of i at time t is subtracted from the reference r at time t, giving the error value E(t). This is processed by a control function to produce an output signal U(t). This output signal is sent to the plant, enacting some change on the world. The goal is to have i match r at each time step, by choosing an appropriate signal U(t) to send. However, disturbances in the world represented as signal D(t) can affect the effectiveness of control of i.*

Closed-loop systems have greater accuracy and robustness than open-loop systems. However, closed-loop systems suffer instability as a result not calculating what a future error signal may be [DiStefano et al., 1967]. This can result in oscillations or steady-state error, where the input remains a close but constant distance from the reference. Even with these trade offs, simple closed-loop systems are still employed today to solve control problems. Although they cannot optimally solve non-linear problems with certainty, they have shown better competency towards non-linear problems than open-loop systems [DiStefano et al., 1967]. The simplicity of closed-loop systems combined with their general effectiveness make them a commonly used controller in Control Theory.

2.1.2 PID Controllers

Proportional-Integral-Derivative (PID) Controllers are a widely employed control system within classic Control Theory. The proportional component calculates an output signal based on the current error. The integral element integrates the error over time, allowing the controller to account for error that has persisted for a long time, known as Steady-State Error. The derivative component uses the gradient of the error to compensate for going past the reference point, which is known as overshoot. A PID controller is used in a variety of control solutions where a problem requires a computationally inexpensive but effective solution. The equation for a PID controller is shown in equation (1). If u(t) is the output for the controller at time t, then the equation is as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

Where e(t) is the error at time t and τ is the time constant, intuitively representing the step response between iterations. Equation 1 is used as the controller element in Figure 1 to form a negative feedback control system. PID controllers are used as single layer controllers [Shabani et al., 2013] or in a two layer hierarchical arrangement [Dash et al., 2015], known as 'Cascade Control'.

A common problem with PID controllers is known as 'Integral Wind-Up'. When an actuator is saturated with output signal from the control system, the integral term can end up disproportionately large due to poorly predicting the output needed to account for past error [Bohn and Atherton, 1995]. The calculation for the integral term does not take into account the maximum effect of the actuator and thus ends up anticipating a large error it believes to exist. The integral term then takes time to be countered by the proportional term, or oscillates continuously between too far above and below the reference point.

There are a variety of solutions to integral wind-up, known as anti wind-up techniques. These techniques modify or constrain the control function to avoid the winding up of the integral. Common solutions are to put upper and lower bounds on the integral value, have the integral value decay naturally over time which is known as a 'leaky integrator', or predicting when the control signal is inflated beyond the controllable range of the control variable and modifying the integral accordingly. When building a PID controller, it remains something that one must account for in the design process.

2.2 Hierarchical Control

Classic Control Theory and PID Controllers formed some of the core building blocks for control of robots. Complex approaches involving multiple controllers emerged from these techniques, allowing effective control of multiple tasks. Layered or hierarchically arranged controllers that each handled specific behaviours were conceptually appealing. An understanding of typical hierarchical approaches is undertaken in this Subsection, with the aim of identifying what is missing from hierarchies and preventing their application to more complex problems.

2.2.1 Conventional use and definition of 'Hierarchy'

The Oxford Dictionary defines 'Hierarchy' as:

> *"A system in which members of an organization or society are ranked according to relative status or authority. Alternative Definition: An arrangement or classification of things according to relative importance or inclusiveness."*
>
> — *Oxford Dictionary, Definitions for the word 'Hierarchy'*

This definition covers what a hierarchy is, but not in the context of control. Words such as 'authority', 'importance' and 'inclusiveness' do not have definitions in the context of Control Theory. Control nodes do delegate commands to lower nodes, which could be considered a hierarchy of authority. However, no nodes are any less important in a hierarchy as they are all required for the control hierarchy to function.

Corporate or organisational hierarchies follow the definition presented above, but do not inform about control hierarchies or their semantics. Organisations have hierarchies that follow many different forms, like control hierarchies. Hierarchies often start with CEO, who hires the immediate members necessary for function. As it succeeds, new people are added at various levels to handle the higher level functions desired of the corporation. It is important to conclude that this evolution of the corporate hierarchy emerges from the context of the organisation, but no indications on how this would emerge in a control problem.

Control hierarchies vary in practice as much as corporate structures do, meaning a consistent definition is hard to find. Many words are used to describe an individual entity within a hierarchical control system. Units, subsystems [Findeisen, 1984] and nodes [Albus et al., 1981] are all terms commonly used. Before understanding a control hierarchy, one must understand a control system, and then a control subsystem.

A control system is a device, tool or some implementation that controls some property about its world. It is unclear exactly what a control subsystem should be, as any number of individual parts of a control system could be considered to be enacting some aspect of control contributing to the whole system's output. Findeisen in his work 'The Essentials of Hierarchical Control' helps identify what a control subsystem is. Specifically, a subsystem has some clear sub-goal which helps the overall control system achieve the main goal(s) it wishes to achieve [Findeisen, 1984]. This is important for three reasons. First, this clarifies what a subsystem is, in that it must have some clear subgoal to be a control subsystem, *i.e.* it must control something which pertains to the full system's control objectives to be a control subsystem. Furthermore, this tells us about the relation between subsystems that are hierarchically arranged, in that they use one-another to complete their goals, where one assists another. Finally, the detail that any subsystem must control something means it must have what it requires to control said something. This would mean a control subsystem must have at the bare minimum an input (which it aims to control) and a reference (the desired value of said input) or some computational equivalent.

This presents a semantically richer definition of a control hierarchy. A control hierarchy can be present in a control system, where the control system is decomposable into a variety of control subsystems that are hierarchically arranged. These subsystems each have their own control goals that contribute to the greater control goal(s) of the control system. There are many terms for the same object or concept in control theory, particularly in hierarchical control. This Thesis defines the terms that are used throughout this Thesis:

- **Control Subsystem** A control subsystem within Control Theory is a distinct part of a control system. More exactly, a control subsystem is some control function which takes inputs and delivers outputs, with the aim of the input being

controlled by the output. A control system thus can be comprised of multiple control subsystems.

- **A Feedback Element** is a device which provides an input encoding something about the real world, such as a light sensor.
- **A Plant** is a device which takes output signals, processes the signal and performs some real world action.
- **An actuator** is a real world device that acts on the world. An example may be a motor. An actuator only covers the device itself, whereas a plant also covers the converting of the output signal to some real world effect.
- **An Input signal** is a signal provided to a control subsystem. This is a signal that the subsystem is aiming to control, by enacting on the real world. These are displayed entering a control subsystem on the left.
- **A Reference signal** is a signal that indicates the goal or target of a subsystem. These are displayed entering the control subsystem from above.
- **An Output signal** is the output of a control subsystem, delivered to plants or other control subsystems to enact change on the world and influence the input signal. These are displayed leaving the control subsystem on the right.
- **A Gain** is a floating point number which multiplies a single output of a control subsystem. This allows the output to be magnified or reduced based on the sensitivity of the place it would output to. This is a type of parameter commonly used in closed loop control.

It is evident that a hierarchy in the context of control is some collection of control subsystems that each have their own control task, where the interactions between them achieve a greater or more complex task. It does not however inform how a hierarchy is made, what inputs are selected, or what functions must be used to change the output of one subsystem to be a reference or input for another subsystem.

2.2.2 Hierarchies *vs* Layers

Layered Control and Hierarchical Control are both used to describe control systems with multiple control subsystems, but a consistent definition is difficult to find. Because both are structural systems with subsystems, it can be difficult to discern them apart. Mitchinson describes Layered Control and Hierarchical Control as:

> *"A layered control system is one in which there are multiple levels of control at which the sensing apparatus is interfaced with the motor system. It is distinguished from hierarchical control by the constraint that the architecture should exhibit dissociations, such that the lower levels still operate, and exhibit some sort of behavioral competence, in the absence (through damage or removal) of the higher layers but not vice versa."*
>
> — *B. Mitchinson, MiRo: A Robot "Mammal" with a Biomimetic Brain-Based Control System [Mitchinson and Prescott, 2016]*

This paper cites and is supported by Prescott [Prescott et al., 1999], with detailed analysis of how Subsumption Architecture differs from typical hierarchical control. The definition seems intuitive, indicating that layered control does not rely on that which is below it. However, this is not necessarily how it is used in practice. Others refer to what Prescott would describe as hierarchical control as layered control [Bianchi et al., 2011]. This is a common occurrence and is due to what layered means outside of a control context.

A more general definition of 'layered' is something having distinctly identifiable partitions which are arranged on top of one another. This definition has crossed over to conventional use in determining whether a control system is hierarchical, layered or sometimes both. Aicardi refers to an architecture in their studies as 'a two layered hierarchical architecture' [Aicardi et al., 1995], stating that it is both hierarchical and layered. A common use of a layer is a subsystem or group of subsystems that perform some specific type of control on the environment and have nothing beside them in the hierarchy.

The definition given by Prescott is behavioural and not related to the structure of the control system. Hierarchies and layers are structural entities, meaning the semantic information that makes it a hierarchy must be evident in the structure. The definition examines the behaviour of controllers without other parts of the system. This test could well identify a lot of hierarchies, but functioning worse without a lower subsystem is not necessarily a requirement. For example, a badly tuned hierarchy is still a hierarchy even if it doesn't function well. It is unclear if a dysfunctional hierarchy would thus fail the test. Furthermore, higher levels of a hierarchy may be able to mitigate issues or damage with lower subsystems and their actuators, or perform suitably without their lower subsystems on the behavioural test. If this is the case, a functioning hierarchy that can robustly adapt would be classified as not a hierarchy. Additionally, if a control system has both layered and hierarchical elements, it is not clear what this definition would define them as.

Definitions separating hierarchies and layers are lacking in the literature, with only one definition and that definition has been shown to be lacking. The definition from Prescott focuses on the behaviour of the control system to identify whether it is layered or hierarchical, and not the structure. Using behaviour that is symptomatic but not always a property of particular structure means classifications can be incorrect and can be subjective. A definition that analyses the underlying structure is required to discern layers from hierarchies consistently. Analysing hierarchical controllers in detail is required to enable a definition of a hierarchical control structure.

2.2.3 Hierarchical Control and Robotics

Key definitions surrounding hierarchies and structural controllers have been detailed. This Section details key hierarchical controllers to gain a better understanding of their contributions and limitations. This Section aims to find where control hierarchies are not being utilised to their capacity to inform better approaches to control.

Structural approaches to AI became common from the 1980s onwards, as computer processing power increased to allow distributed programs to be feasible and trainable. A notable example of a robot being controlled with this kind of framework is SHAKEY, that was programmed with logical conditions mediating behaviour processes. Initial work and construction of the robot was done between 1966 and 1972 using Stanford Research Institute Problem Solver (STRIPS) [Cocosco, 1998] and this work was extended

to improve on the system in later years [Shanahan and Shanahan, 1998, Shanahan, 2000, Shanahan and Witkowski, 2001]. While not necessarily a hierarchy, these were part of a movement towards distributed and structural control.

The Real-time Control System (RCS) architecture has an extensive hierarchy for controlling generalised problems in robotics [Albus, 1993, Albus and Rippey, 1994]. RCS explains how information is drawn from the environment and then provides a hierarchy. The generalised hierarchy indicates that sensations are at the bottom and high level planning at the top. The different hierarchical tiers depend on the time in which it takes to perform the task. Sensations change quickly, but high level tasks do not. This is an interesting theory, but doesn't provide the semantics of what is hierarchically above what. Many hierarchies in Cascade Control have the velocity and position of an agent in a hierarchy. However, these both change at the same rate and are hierarchically arranged in many hierarchies. As such, it is difficult to understand how an agent would be built using these hierarchies.

Work that extends RCS, called the Reference Model Architecture (RMA) instead identifies the tiers of the hierarchy by name [Albus et al., 2004]. Instead of being based on the rate of change, the layers are classified based on the task type, such as 'Servo', 'Primitive' and 'Task'. These tiers amount to a similar architecture to one which assigns tiers based on rates of change, thus providing no extra information on how hierarchies are generally built. These control architectures are limited in their application, so examining hierarchical architectures that have been used widely is advised to better understanding control hierarchies. Subsumption Architecture is a widely employed hierarchy and thus worth investigating.

Subsumption Architecture is a hierarchical control system with different levels embodying different behavioural aspects [Brooks, 1986]. The lower levels are subsumed by higher levels where necessary, with higher levels releasing control to lower levels when the context is appropriate. All levels receive the input but higher levels have the ability to suppress or inhibit the output of lower levels. The levels are thus added incrementally, trained as to when they should override the lower levels and take control, prioritising output.

The architecture has considerable strengths, shown by producing the best selling commercial robot to date, the Roomba [Brooks et al., 2015]. However, no publications exist showing Subsumption Architecture implementing all the layers shown in the original publication. Published work shows the complexities of managing multiple independent controllers, with increasing complexity and difficulty in elegantly combining their decisions [Rosenblatt and Payton, 1989]. When modules compete to act or control, the system which manages the competing desires to act must be robust and well-tuned. As more controllers are added, this becomes increasingly complex and can require re-tuning all the previous layers and their interactions, explaining why Subsumption Architecture can scale poorly. Various attempts have been made to modify or combine Subsumption Architecture with AIs [Togelius, 2004] [Amir and Maynard-Zhang, 2004] [Birnbaum and Collins, 1991] but with no substantial success and nothing which solves the original issue of scaling poorly.

It is worth discussing whether Subsumption Architecture is a hierarchical or layered system. Figures 3 and 4 are the original zeroth level implementation and the level zero and one implementations for Subsumption Architecture [Brooks, 1986] respectively. A circle with an I indicates an inhibitor node (which stops the output of a specific controller) and an S indicates a suppressor node (which stops the output and replace it with its own).

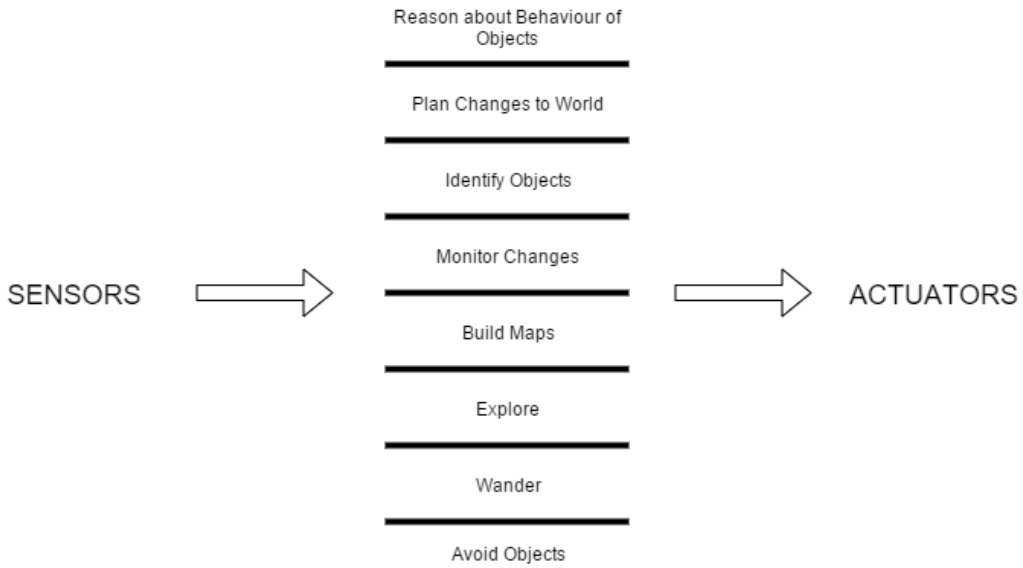

Figure 2: *A diagram showing the behaviours in Subsumption Architecture, reprinted with permission from [Brooks, 1986]. The behaviours were added to the system incrementally, with each layer being built on top of the last and optimised.*

There are no inhibitor nodes in the zeroth and first level, but the connection feeding output from the first level back to the zeroth level in Figure 4 has a suppressor node.

As seen in Figure 3, the first level introduces the Wander and Avoid nodes. The Wander node produces a heading which goes towards an unchecked direction. The Avoid node takes the force from FeelForce which indicates where obstacles are and the heading from Wander. The Avoid node compromises between the two, having the heading that explores move away from obstacles. This suppresses the Runaway command and replaces it with its own.

By Mitchinson and Prescott's definitions, the distinguishing feature of a hierarchy is that the lower level functions independently of the higher level, but not the other way around. If both work independently, it is layered. The lower level can be assumed to function fine without the higher level.

By simple observation, it is a hierarchy. The Avoid node as a finite state machine only chooses to suppress the runaway module when it requires acting. Therefore, the runaway module's behaviour when Avoid is not acting is instrumental to the overall function of Avoid, and, it is critical to the behaviour and would compromise Avoid if dysfunctional.

Further examination of the structure raises questions about whether it is a hierarchy. Feelforce provides the force of objects to Avoid and combines them to produce a heading which Avoids objects and wanders. This node contains the functional purpose of the runaway module, arguably making it obsolete. Whether the Avoid module subsumes runaway or replaces it depends on the control function. Given the inputs to the Avoid node can be processed to achieve Runaway's function, it is not necessarily dependent at all on Runaway and thus is layered.

Subsumption Architecture has demonstrated the issues of a behavioural definition for hierarchies. A more elegant definition that refers explicitly to the structure and environment would be more helpful. Furthermore, the implementation shown in Figure

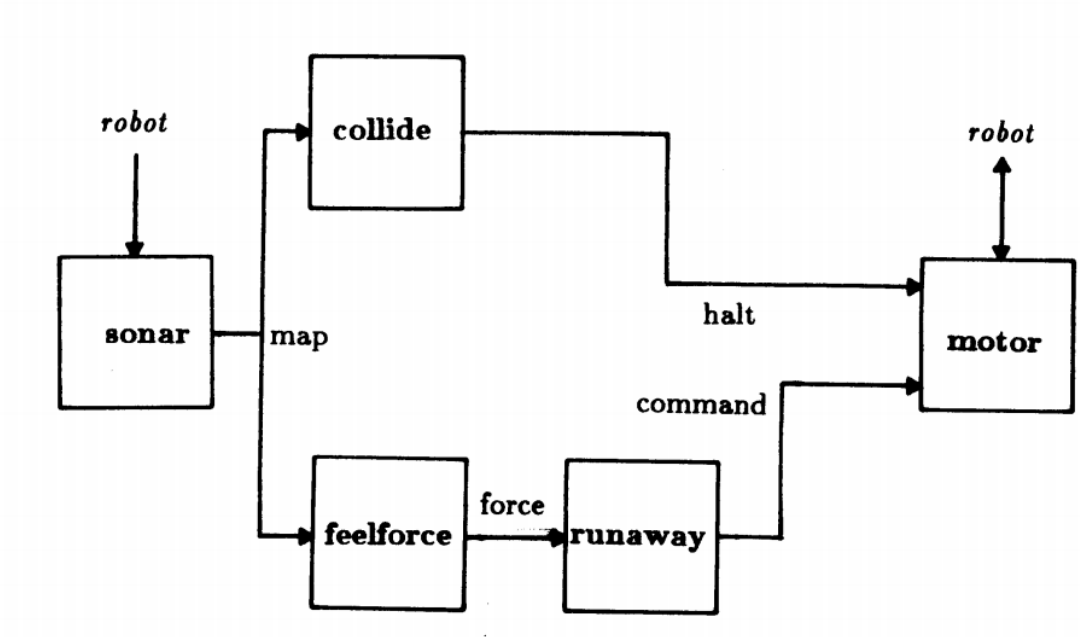

Figure 3: *A diagram showing the implementation of the zeroth level of Subsumption Architecture, reprinted from [Brooks, 1986] with permission.*

3 is clear, but does not inform how a Subsumption Architecture could be built for a different problem. It is not clear what the layers should be for a particular problem, what input signals are required, or how the subsystems interact. To allow a hierarchical controller to be applied to many problems, it must be clear how to apply it to a given problem.

When one considers the work extended from the original Subsumption Architecture presented by Brooks, the only modifications have been to change the contents of the levels or combine the Subsumption approach and principles with other AIs. One of the underlying criticisms is the separation of behaviour with regards to the levels, such that the behaviour by combining them is dependent on careful optimisation of the inhibitions and suppressions [Rosenblatt and Payton, 1989]. Because the behavioural layers do not cooperate and are swapped between, Subsumption Architecture relies critically on the well balanced suppressions and inhibitions. As behaviours are added, they can become incompatible with previous suppressions and inhibitions. A hierarchy should not require entirely re-optimising lower levels when higher levels are added.

Overall, Subsumption Architecture is a system built off of the elegance of layered and hierarchical behaviour. However, certain aspects remain unclear with both Subsumption Architecture and Cascade PID Control. How the subsystems for a hierarchy are decided and how the hierarchy is arranged remain unknown.

2.2.4 John Doyle's Hierarchy Methodology

John Doyle demonstrated a theory of hierarchies in many areas, such as physics, control and neurology [Doyle and Csete, 2011]. The methodology posits that hierarchies occur in the form of an hourglass or bow tie in shape. At a low level, an agent has a lot of possible sensations that can occur in the environment. However, only specific combinations of those might be useful or relevant, even less of those combined form specific movements

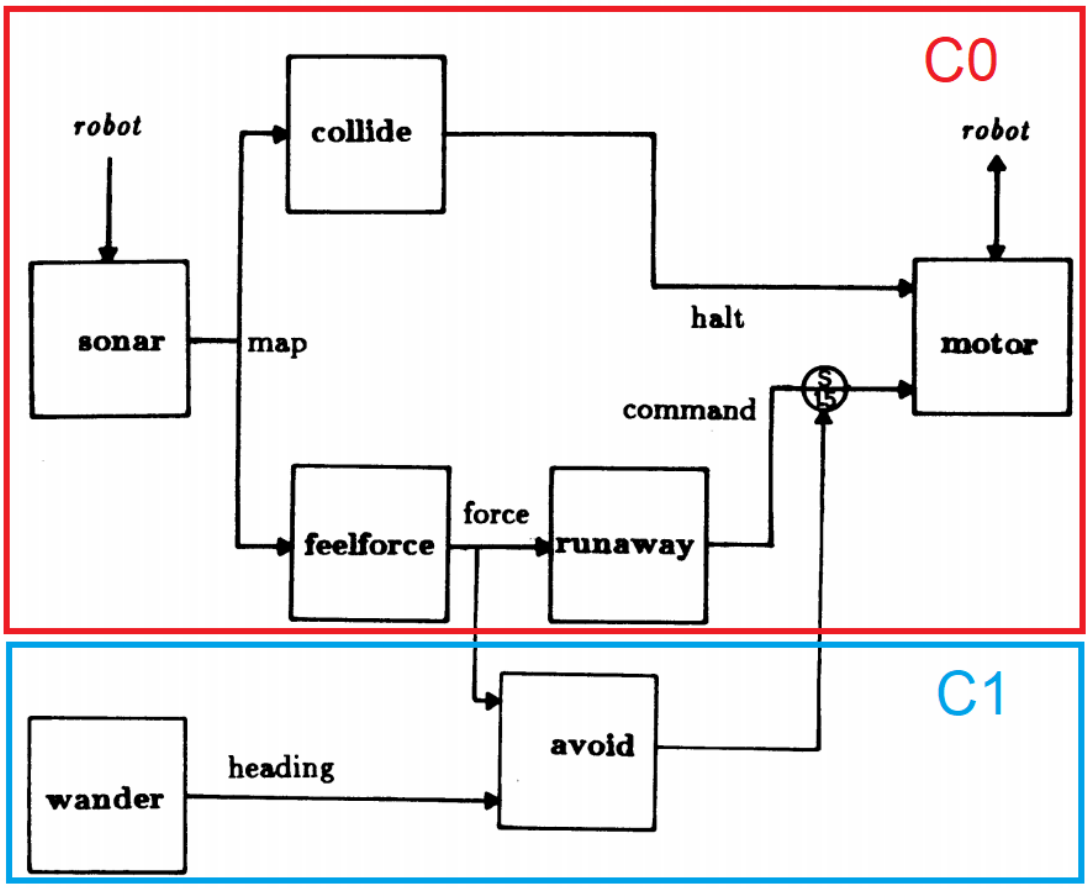

Figure 4: *A diagram showing the implementation of the zeroth and first levels of Subsumption Architecture, reprinted from [Brooks, 1986] with permission. The bottom two boxes in this diagram, labelled 'wander' and 'avoid' (contained within a blue box) are the first layer of Subsumption Architecture, whereas the remaining boxes (contained within the red box) are the zeroth layer. The suppressor node, which is the circle with the S and 15 in it, suppresses the output of the zeroth level. When the first level wants to assert control, the suppressor node divides the zeroth level's output by 15.*

that an agent might do. Eventually, this decreases to become a core set of conceptual skills an agent might interface with. For example, the conceptual action of grabbing an item can be done in many different ways. The context of the action matters, such as what the item is, where it is, why one is grabbing it, *etc.* But, the general concept remains the same.

Above the theorised bottleneck in John Doyle's Hierarchy, number of reasons these conceptual skills that might be invoked starts to widen. Concepts, ideas and motivations build on top of these skills to act on the world. There are many reasons to grab something, from simple survival to one's societal duty. Plus, there are many ways of grabbing the same object, let alone objects of different sizes and in different locations. Many examples are demonstrated by Doyle, but for the purposes of this Thesis, studies of a cognitive agent trying to control their environment takes focus.

Doyle theorises that the hourglass structure filters out irrelevant perceptual combina-

tions of sensations, allowing elegant processing of a large supply of sensory data. Living agents must be capable of this in order to identify key perceptions that enable an agent to survive. The response time of an action is how much of the hourglass hierarchy it must pass through and thus actions that invoke a lower level directly takes less time. So, direct instinctive responses to avoid something such as a burning pain won't be processed at a higher level.

An important note made by Doyle is regarding the overloaded use of the term 'Abstraction' for the separation between lower and higher nodes. The term used instead is 'Virtualisation', so the higher levels use a virtualised set of representations for the lower layers. In essence, a higher node has some interpretation of how the lower level behaves and how it is to be given tasks, but doesn't need a deep knowledge of how it works. So, this is an interface between the layers. This bears an obvious similarity between abstract classes and interfaces in programming.

Abstraction is the process of considering something independently of its associations or attributes. Certainly, multiple nodes in a hierarchy can operate independent of the associations and attributes of lower layers. Their communication does not necessarily require any knowledge of what lies beneath. However, *learning* how to use lower levels requires the effective function of lower levels. A virtualised interface requires some knowledge of what it is interfacing to. Importantly, this allows it to process incoming input to deliver output in a form that achieves the desired results. Perhaps this subtle terminology difference gives an important context that have failed other hierarchical structures in the past.

If human architects of control systems are providing the required connections and allowing the system to simply learn the method of communication, a lot of the work is done. The interface is already directed to the right place. But, finding the right place requires a greater understanding of the needs of the higher layer. Or, what is more likely the case, is lower levels can be built on by producing interfaces that produce useful behaviour from what is already known. Certainly, this is the first piece of information that provides conceptual clues as to the origin of hierarchies. Abstractions appearing out of nowhere must be replaced with some tool to learn virtualisations progressively, bottom to top.

This methodology explains the principles of what a connection between two subsystems should do and an overall architecture that should be followed, but does not inform on how a hierarchy emerges from a particular problem. Some understanding of the types of inputs that occur lower in the hierarchy is detailed, but this information is not enough to design and implement a hierarchy methodically.

2.3 Summary

Hierarchical control has proved effective in complex control problems. Application of hierarchies to problems despite this is inconsistent and sparse. This chapter reviewed hierarchies in the literature to examine why hierarchies are not being utilised. Analysis of existing hierarchies has found that an understanding of what a hierarchy is is lacking. This means that hierarchies are not being applied properly or consistently and no methodology exists to make sure a hierarchy is appropriately made. The next chapter reviews literature outside of control theory to shed light on control structures to enable a greater understanding of hierarchies.

3 Living Control Systems

Structural approaches (such as hierarchical or layered systems) became popular when a shift towards connectionism over symbolicism occurred. Connectionism was originally proposed by Hebb in 1949 [Hebb, 1949] but could not be utilised effectively until the 1980s where computational power increased to allow large structural AI. Connectionism supports the notion that cognition is the result of simple units connected together producing intelligent behaviour, rather than a single entity managing high level concepts. Thus, connectionism rejects the notion that there is a specific entity for high level concepts, but the management of them comes from emergent interactive behaviour of the connective system. Since connectionism became viable, comparisons began to be drawn [Horgan and Tienson, 1991][p. 382-416] and are still being debated [Dinsmore, 2014] as to which is the better approach to AI and cognitive systems. Connectionism boasts emergent intelligence without manually deciding symbols and concepts, but often requires extensive training to replace manual selection of symbols.

Connectionism hinted to the power of multiple smaller computational processes interacting together to produce intelligent behaviour. Structural approaches became popular within control, such as Subsumption Architecture [Brooks, 1986] and STRIPS [Shanahan and Shanahan, 1998, Shanahan, 2000, Shanahan and Witkowski, 2001]. These approaches do not detail what a hierarchical approach should be. However, studies of biology and psychology have developed beyond simple principles of Connectionism to a more fluent understanding of the mind and body.

Discoveries in biology and psychology have been used to inspire AI and control theory, allowing structural approaches to control to be built that are more complicated than traditional approaches, such as Subsumption Architecture or Cascade Control. This chapter examines studies that model and understand the neurological, biological or psychological structure of living agents to inform how control systems should be built. The aim of this chapter is to use this information to understand what makes a hierarchy.

3.1 Control Structures in the Body

3.1.1 The Basal Ganglia

The Basal Ganglia has received significant attention in neurological studies. It has been identified as a key area for cognitive processes, from low level skills such as attention [Teicher et al., 2000, Ravizza and Ivry, 2001], perception [Brown et al., 1997], habitual repetition [Knowlton et al., 1996] and learning [Packard and Knowlton, 2002] to higher cognitive skills such as language [Ullman et al., 1997, Teichmann et al., 2006, Prat et al., 2007], planning [Dagher et al., 2001, Anderson et al., 2005], problem solving [Anderson, 2005, Stocco and Anderson, 2008] and reasoning [Goel et al., 2000]. The structure of the Basal Ganglia is key to how it achieves cognitive prowess.

The Basal Ganglia (BG) has presented researchers with questions regarding to the overall structure of this part of the brain and how the biological structure produces the BG's cognitive benefits. While conventionally the BG was considered to be concerned with muscular control, it is shown that the BG relates to action selection [Redgrave et al., 1999]. Important research has originated justifying hierarchical control as the structure of the Medial Reticular Formation (MRF) [Humphries et al., 2007] which is another area which is argued to have an impact on action selection. Subsumption Architecture is mentioned as one of the plausible candidates for a matching architecture to the MRF.

Redgrave initially speculated that the BG acts as a centralised architecture with a similar structure to the Global Workspace [Redgrave et al., 1999]. However, the central node of this structure is still something of a 'black box', lacking any formal definition of how the central node produces decisions between the options. There have since been computational tests [Gurney et al., 2001] and tests on a robotic agent [Girard et al., 2003] which show capabilities of selecting actions for a given task. The tests did not extend to complex decisions.

The exact neurological structure of the BG remains contested. However, many models have been made attempting to recreate the behaviour of the BG. Action Selection is a key cognitive process, meaning the models of the BG aim to recreate this behaviour. These models are inspired by theories of the underlying structure.

Further work into models of the BG suggest a structure which contains more elements than that proposed by Redgrave, but with the same structure. Cisek proposed an architecture named the Affordance Computation Hypothesis which has actions competing for selection by the BG [Cisek, 2007]. This model draws comparison to Sequential Sampling Models, which has actions accumulate 'evidence' until they reach a threshold and are then invoked. Contrasting those models, Cisek's model encodes actions differently. One single population produces the encoding of various actions in Cisek's model whereas previous models have distinct populations to represent each action [Cisek, 2007]. Overall, Redgrave's model and the Sequential Sampling Model have been updated to isolate lower representations of parts of the architecture.

Houk examines action selection on living agents and combines existing knowledge of the BG to produce a model [Houk et al., 2007]. The focus is the encoding of actions through Distributed Processing Modules (DPMs). DPMs are modelled from the BG and related areas of the brain from neuroscientific studies. Importantly, the DPMs use negative feedback for self-regulation and are used to encode not just actions, but build plans and commands from the actions. Houk states that these DPMs could be an appropriate model for a lot of high level cognitive behaviour [Houk et al., 2007].

Further models of the BG use Reinforcement Learning. There are many models focused on Reinforcement Learning, mainly using Temporal Difference (TD) methods and the Actor-Critic method (see [Joel et al., 2002] for a comprehensive list of related models). Other Reinforcement Learning algorithms have been successfully applied [O'Reilly and Frank, 2006, Dominey et al., 1995] but TD is the most applied and the most successful [Stocco et al., 2010]. TD methods are an application of Reinforcement Learning where the agent aims to minimise the expected and actual values returned for an action, which is the temporal difference [Sutton, 1988]. Authors have stated that there is success with these models and that the features of Reinforcement Learning bear similarities to the functioning of biological aspects of the BG [Stocco et al., 2010]. Given the number of models and how these have recently been extended to parts beyond the BG [Haynes, 2014, Takahashi et al., 2008], this is a strong claim.

The Actor-Critic method hosts many BG models [Haynes, 2014, Stocco et al., 2010, Joel et al., 2002, Takahashi et al., 2008, Khamassi, 2005] with a variety of applications. However, the focus of each model is to use it to simulate a specific element of behaviour with no generalisability. This is to be expected as the Actor-Critic Model is a simulation of the behaviour of the BG rather than an architecture which mimics the structure.

Of the approaches presented, suitable behavioural simulations have been achieved as well as analysis of the overall structure. This has produced architectures with biological inspirations from the BG [Redgrave et al., 1999, Cisek, 2007, Houk et al., 2007]. Notable

conclusions which dictate further research are the use of hierarchical or modular structures to mimic the BG [Humphries et al., 2007] and the use of Reinforcement Learning as a model for the BG [Joel et al., 2002]. Hierarchical structures have successfully mimicked the behaviour of the BG [Haynes, 2014, Stocco et al., 2010, Joel et al., 2002, Takahashi et al., 2008, Khamassi, 2005].

The approaches that do not use RL have a central subsystem that selects between subsystems who wish to act. The central subsystem controls which other subsystems act, putting it at the top of a hierarchy and submitting references to them. The unusual aspect is that in some of the models compete for favour of the parent subsystem. This would mean structurally that the subsystems each feed input signals to the higher controller in order to inform them of their contextual relevance and their need to enact control. This hierarchical arrangement would suggest that a hierarchy emerges from contextual relevance to the problem, but how this is evaluated in a general hierarchy remains unclear.

The approaches that use RL rely on an Actor-Critic structure, which involves a higher subsystem (the critic) changing the control function of a lower subsystem (the actor) through feedback. The critic does not provide a reference control signal, but rather a motivator signal to adapt control behaviour. This is not a control hierarchy, but a learning hierarchy. While this is useful, it does not inform about where structures emerge in control hierarchies. However, it presents the possibility that principles of hierarchies could be used for learning a control hierarchy.

Overall, the only notable point for control hierarchies is the use of hierarchical selection through context of which subsystem would best function for the task at hand. But, no detail of what this context is or how an agent can define this context for any problem.

3.1.2 Cerebellar Controllers

The Cerebellum has been linked to adaptive filtering of a human's behaviour and is responsible for a repertoire of adaptive behaviours [Dean et al., 2009]. Recent anatomical studies have shown internal recurrent connections being attributed to associative conditioning [Gao, 2016]. The Cerebellum's structure and how this produces adaptive behaviour remains undetermined. A comprehensive coverage of the anatomical research can be found in 'The Perceptual Shaping of Anticipatory Actions' [Maffei et al., 2017]. The focus of this Section is not the structure of the Cerebellum, but rather controllers and models which replicate the behaviour of the Cerebellum and the structures of those controllers.

Cerebellar models being utilized for control problems have been theorised for over 40 years [Albus, 1975]. Cerebellar controllers have been produced that are based on supervised learning of basis functions [Maffei et al., 2017] and recurrent feedforward systems [Porrill et al., 2004]. All of these controllers are placed above a classic feedback controller, which is posited to mimic the brain stem [Porrill et al., 2004]. The latter uses internal recurrent connections, which has been shown in the Cerebellum in anatomical studies [Gao, 2016].

These controllers have produced anticipatory behaviour [Herreros and Verschure, 2013] but lack the competence of a functional cerebellum. The generalisability of the cerebellum is not produced by these models. A controller could not, for example, generalise to a novel signal based on previous signals. If the warning signal encoded some information which revealed details about the impact, multiple patterns of behaviour would be needed to solve this problem. As an example, assume a problem where an agent must learn when to close its eye to avoid an air puff entering the eye. On a warning signal of one,

the agent receives the puff after a second. On a warning signal of five, the agent would receive five seconds before the puff occurred. An intelligent agent would extrapolate from this that a warning signal of three would give them three seconds until the puff. The models previously mentioned are not able to achieve this, as they do not have a rich bed of behaviours to call on. So, the above can be considered one behavioural expression of the cerebellum.

However, studies in Echo State Networks worked towards solving this problem. Echo State Networks are a specifically formed deep neural network of three layers. The first is a layer for encapsulating input and then projecting this into the middle layer. The middle layer, commonly called the reservoir, aims to project the input into higher dimensional space in order to capture relevant features. There is internal feedback to the reservoir with sparse interconnectivity. Finally, the output layer aims to take the features from the reservoir that can be used to minimise error and formulate an output that minimises error. There is feedback from the output layer to the reservoir, such that the reservoir attempts to highlight features and behaviours that benefit the output layer. These two forms of feedback internal to the Echo State Network are posited to mimic the internal feedback of the cerebellum[Yamazaki and Tanaka, 2007, Gao, 2016].

Studies have concluded that Echo State Networks are able to encode temporal dynamics [Rössert et al., 2015] but whether this allows for temporal generalisation to unknown situations remains unknown. An example might be training the agent on a stimulus which has a magnitude proportional to time in which it is received. So, a magnitude of one might mean an impact in one second but a magnitude of five might mean an impact in a fifth of a second. The agent could be trained on discrete magnitudes and then tested on the full continuous spectrum of magnitudes, to see if it could generalise to a magnitude of 2.5 from having experienced signals with magnitudes of two and three. Comparing an Echo State Network controller with a supervised learning feedforward controller to show a significant difference in performance.

Cerebellar controllers are a specific control hierarchy where the upper subsystem handles future predictions of the input signal to compensate for the lower subsystem not doing this. The upper subsystem directly references the lower system, motivating it to act towards the goals of the higher subsystem. However, not all control hierarchies are built with the higher level performing anticipatory behaviour. The structure of the cerebellum nor these cerebellar controllers have a wide enough context to conclude how control hierarchies emerge.

Echo State Networks are hierarchical with an unusual self-referencing aspect. The networks are clearly hierarchical, with each performing a separate transform that is then delegated to lower networks for further processing. The combined processing of all the networks produces the control output, but cannot function properly without any particular network. The unusual property is that some networks send output to themselves, meaning they are hierarchically arranged above and below themselves. While this property seems bewildering, it is analogous to those networks retaining input from previous iterations for further processing. Echo State Networks and more generally Artificial Neural Networks are hierarchical and built regularly. More research on how these are built and utilised is warranted.

3.2 Artificial Neural Networks

The features of neurons and the brain have inspired theories, models and architectures, of which Artificial Neural Networks being the most applied. An Artificial Neural Network

consists of some number of neurons as computational nodes, with weighted connections between the nodes indicating how they impact one-another. The inputs to the network propagate through progressive matrices of neurons, with the weights between the matrices transforming the input in a complex non-linear fashion.

The original Artificial Neural Network model was produced in 1943 [McCulloch and Pitts, 1943]. McCulloch's work did not produce competently learned control, but with a learning algorithm, the Artificial Neural Network is able to perform well in complex tasks [Schmidhuber, 2014]. Three major learning algorithm archetypes are used with Artificial Neural Networks, namely Supervised, Unsupervised and Reinforcement. Supervised Learning algorithms train the model with knowledge of the correct answer. From the model seeing the correct answer and the inputs, the model adapts itself so the correct answer is achieved on that input. By applying this to enough examples, the model is assumed to be able to produce the high level and complicated rules that underlie the problem. Unsupervised Learning is where the model is left to train itself on the data rather than knowing the answer in advance, using statistical correlations to identify patterns or trends. Finally, Reinforcement Learning takes observations at regular intervals, detailing the value of a specific action. From the values of given actions, the agent forms a policy. A policy is a description of which action is best to take in which state to maximise reward. Importantly, Reinforcement Learning does not receive the correct answer in the same way supervised Learning does, but it receives a reward signal indicating how good the action undertaken was. This informs the action values and thus the policy.

The Perceptron was created [Rosenblatt, 1957] and used to allow an Artificial Neural Network to learn the solution to a linear classification problem with Supervised Learning, which is a widely applied algorithm even in today's work. The first Unsupervised Learning algorithms were used around 1970 [Grossberg, 1969, Kohonen, 1972].

Research on Artificial Neural Networks then receded. Criticisms of Artificial Neural Networks and their limitations caused the field to stagnate by the 1970s [Mynski and Papert, 1969]. The computational demand of Artificial Neural Networks meant the hardware at the time was insufficient, and a single level Artificial Neural Network can only solve linear problems: for example, it could not solve the 'exclusive or' problem. Increased computational capacity allowed multi-layered nets. Combined with techniques from Machine Learning, Artificial Neural Networks were able to solve complicated problems with noteworthy accuracy [Graves et al., 2009, Simonyan and Zisserman, 2014].

A Deep Neural Network (DNN) is multiple hierarchically arranged Artificial Neural Networks and has been applied to classify handwriting [Graves et al., 2009] and recognise objects [Simonyan and Zisserman, 2014] with state of the art results. Furthermore, 'superhuman' results were published on sign recognition for DNNs [Cireşan et al., 2012] (a full overview of achievements and progress can be found by Schmidhuber [Schmidhuber, 2014]). DNNs have been applied less to the field of robotics. Some work exists, including problems such as grasping an object [Lenz et al., 2015] where the system attempts to classify the best grasping location based on training.

Work on generalised robotic applications has been investigated [Levine et al., 2015]. Here, the robot is presented the task of screwing in a bottle cap or placing a coat hanger on a rack and tested on such to train the network. When the environment is changed, the robot handles the changes well, isolating the key elements of the task and ignoring non-relevant items. For example, the experimenter could move the coat rack half way through the robot's movement or place other coats on the rack with no hindrance to the

robot. The method used elements of reinforcement and supervised learning to allow it to generalise the input data [Levine et al., 2015].

DNNs can only learn behaviour that they are trained on, either explicitly or implicitly. So, the two solutions are to have a big enough data set or change the DNN to perform more complex learning to find generalisations in the training data. DNNs have been specialised mainly to classifying visual stimuli, which doesn't lend itself well to many of the challenges of a robot performing a task. DNNs structurally are built to classify, meaning it is challenging to produce them in a way where they learn to enact on the world. DNNs can be used to control the reference to a controller, delegating acting on the environment to the controller.

Comparing the image recognition capabilities from a DNN [Simonyan and Zisserman, 2014] and a paper published fooling the same DNN with random noise images or fake images [Nguyen et al., 2014] shows flaws in the DNN's approach to learn from the training data. The features often represented in the images that fool the DNN are what might be identified as a prototypical feature of the object in question, such as the red stripes on a white background for a baseball bat. DNNs use data to draw powerful classifications, but the training data does not contain the context a living agent would use to classify objects. Calling these results better than a human when they fail on such clear examples is contentious. In the words of Uexküll, their Umwelt is empty [Uexküll, 1934]. A DNN could not classify with ease what a human could sit on, for a DNN simply does not sit.

Artificial Neural Networks with appropriate learning algorithms are powerful classification engines, showing classification capabilities that have been identified as 'superhuman'. However, the applications to robotics are still emerging. The stimulus-response nature causes problems in real world applications and semantic inaccuracies on values of states. See the Atari game player as an example [Mnih et al., 2015]. The agent values mostly empty boards and also boards full of enemies highly. This is because the game state that has a lot of enemies happens just after the agent gets reward, but this does not match to the intuitive definition of the goal of the game. Small semantic differences are evident of an approach lacking embodiment.

Nicolas Heess has utilised Deep Learning on control problems, with current work showing strong results in continuous action control [Heess et al., 2015]. Heess discussed about the future work needed for Deep Learning with what he described as 'The Curriculum'. It was explained that the results of Deep Learning varied based on the order, time and depth of the layers in a Deep Neural Network. This is intuitively obvious, but has ramifications for the direction of future work in Deep Learning. The order that the agent learns specific skills in is critical and thus needs to be considered in future work. This posits that the development of a hierarchy of skills is key to the success of an agent. Big data achieves robust results with enough data, but it's clear that the right data at the right time allows quicker learning. The hierarchy is an encoding of the developmental order of the skills, but no detail of how a hierarchy could be derived is presented by Heess, nor how their Deep Learning structure was selected.

The hierarchies in Artificial Neural Networks are clearly defined, but how they emerge is not. Networks are manually selected and trained simultaneously on large amounts of data, providing no intuition on why a particular hierarchy solves the problem or what each part does. However, the work by Heess on training individual parts of the hierarchy in a curricular fashion is interesting. The individual networks build on each other to produce a complex task with generalisation, implying that a hierarchy is built from simple to complex skills, with simple skills perhaps requiring the previous ones. This is

clearer, but not a formal definition. Further work needs to be undertaken to formalise what a curriculum is formally and how this could produce a hierarchy.

3.3 Reinforcement Learning

Reinforcement learning gives an agent a reward signal for an action undertaken in their environment. The agent evaluates the reward signals from taking actions and records them, understanding patterns of actions that produce maximal reward. As the agent maximises reward, the agent is expected to generalise their behaviour to meet the desired goal optimally in a variety of configurations and environments [Sutton and Barto, 1998].

A policy is the decision node that decides what action the agent takes, depending on the situation. The policy could be viewed as a collection of stimulus-response rules, where the states and actions are finitely defined.

A reward function assigns the reward for a given state or state-action pair at a moment in time. This is not to be confused with a value function, which decides the value of a given state or state-action pair by considering all previous actions undertaken in this state. The value function acts as a record of previous attempts, or an overall value of the state. The policy can be adapted based on the value function records calculated, indicating which states are advantageous to end up in.

Already, some concerns for the roboticist are present. A requirement of the policy is that discrete actions and states are required that are finite so the value function can evaluate them appropriately. Additionally, Reinforcement Learning requires a lot of training time and data. However, approaches have been undertaken to improve Reinforcement Learning in challenging domains and time constraints. Apprenticeship learning has been used to circumnavigate problematic initial states, which gives the agent a tutor to demonstrate trajectories to learn a policy from. This technique is popular in aerial locomotion [Abbeel et al., 2007] since it is allows learning without first hand experience in flying. Dynamic Movement Primitives, a type of dynamical system, are used to efficiently represent high-dimensional control policies [Pastor et al., 2011]. This can be combined with reinforcement learning to produce superior results in learning control policies. Function approximation has been used in Reinforcement Learning to tackle the finite state and action restriction. Function approximation methods aim to remove the finite action and state representation and predict values directly from a function rather than looking up previous values. Function Approximation has been used with varying results and applications [Melo et al., 2008].

Furthermore, there is an issue with rewards being temporally restricted due to discrete stimulus-response models in Reinforcement Learning. The Credit Assignment problem is noted as one of the biggest problems to solve in Reinforcement Learning [Kaelbling et al., 1996, p. 251], which concerns the problem of a series of actions contributing to a reward but no clear indication as to which action caused the reward. For problems that have delayed reward, the agent should credit appropriately the actions that eventually lead to the reward, not the immediate action. If an agent is considering the action before, the agent never learns purposeful delayed behaviour which is common in a natural environment.

A more diverse reinforcement algorithm was produced, introducing the concepts of Quantum Collapse and Amplitude Amplification to produce a more comprehensive algorithm with a more robust representation of state values [Dong and Chen, 2006, Dong et al., 2012]. The algorithm has probabilistic states of the actions that can be undertaken, producing a well balanced exploratory and exploitative algorithm. The algorithm has

been applied to an autonomous exploring robot with a simple objective and achieved robust results. This does not counter all the aforementioned critiques, but does advance on some of them, such as the discrete action and state representation.

The current challenge of Reinforcement Learning is the curse of dimensionality. Wide arrays of sensory information result in an insurmountable state space, causing problems for searching in such a space. Like traditional approaches, to apply to real-world problems in a robust way, new techniques are needed. Therefore, it is important to see how hierarchical Reinforcement Learning, that is more demanding on training, has been achieved.

3.4 Hierarchical Reinforcement Learning

Reinforcement learning has been used in many hierarchical structures. Hierarchical Reinforcement Learning has shown advantages over Reinforcement Learning and is motivated by many disadvantages in Reinforcement Learning. For example, the curse of dimensionality is mentioned as something inhibited by Hierarchical Reinforcement Learning [Barto and Mahadevan, 2003]. As well, movements towards using hierarchies to improve learning speed and improve abstraction have been undertaken [Uther, 2002, Yamaguchi et al., 1996]. Hierarchically organised Reinforcement Learning where Reinforcement Learning agents call on each other to achieve their desired state yielded successful results [Takahashi and Asada, 1999, Takahashi and Asada, 2000].

Multiple Reinforcement Learning agents have been used to solve many problems, either as a hierarchy or as multiple independent agents acting cooperatively; biped robot control [Yamada et al., 1998], virtual exploration with multiple goals [Digney, 1998], virtual maze problems where the agent can only see one square ahead [Wiering and Schmidhuber, 1997] and RoboCup Soccer [Stone and Sutton, 2001] have all been competently achieved. The training time of hierarchies makes these complex challenges difficult for Reinforcement Learning agents. But, it is possible and realistic to train hierarchical Reinforcement Learning agents even with the curse of dimensionality.

One approach is to have delegated Reinforcement Learning agents each learning Subsections of the task. This is useful in cases where the full state isn't observable and relies on context, formally known as Partially-Observable Markov Decision Processes [Wiering and Schmidhuber, 1997]. Agents have been produced which learn their subtasks and when to delegate to another agent to achieve their goal [Wiering and Schmidhuber, 1997]. Some approaches learn basic tasks as experts then have a gating function select between them [Tham, 1995] or some which take elements of gating functions and delegation [Jonsson, 2006, Morimoto and Doya, 1998, Morimoto and Doya, 2001].

The MAXQ method for hierarchical Reinforcement Learning segments higher level problems through hierarchical delegation [Dietterich, 1998] [Choueiry and Walsh, 2000][p. 26-44]. The agent can learn complex discrete tasks with a designed hierarchy [Dietterich, 1998]. However, hierarchies in Reinforcement Learning produce a hierarchical credit assignment problem, that sub-agents account for penalties that are the responsibility of sub-agents in another tree [Dietterich, 1998]. As well, the hand-designed nature of the hierarchy presents a large amount of priors that the agent cannot modify if the task composition changes. This shows hierarchical Reinforcement Learning solving problems, but it does not define its own hierarchy.

The Options framework represents actions as choices at the states, but also policies to follow for a short period of time [Sutton and Barto, 1998]. These are known as options. The options are learnt independently and hand-selected. Motivation for said approach is

to allow higher level, temporally extended actions to be learned by one-step reinforcement learners [Precup et al., 1998a][p. 382-393]. Work has gone into allowing learning to be extended by off-policy learning [Precup et al., 2005] and combining Options with model-based Reinforcement Learning [Precup et al., 1998b] [Precup et al., 1998a][p. 382-393].

Intuitively, Options seems a lighter addition to Reinforcement Learning than MAXQ but effective. However, work is still being undertaken to have Options be self-defined by the framework rather than provided as a priori [Perkins and Precup, 1999]. The Options Framework, however, does not define their own hierarchy.

Hierarchical Abstract Machines (HAMs) are a tool for placing constraints on the Markov Decision Processes underlying a policy [Parr, 1998, Parr and Russell, 1998]. Unlike the other Multi-Reinforcement Learning agents, the focus of HAMs are that the hierarchy restrains the options of the lower levels to fit higher level abstractions that better model successful behaviour. For example, moving through a grid from the top right corner to the bottom left corner could easily be modelled by always going left or down. Immediately, the state space is much smaller and allows the algorithm to converge quicker. HAMs were shown to have quicker convergence than Reinforcement Learning alone [Parr and Russell, 1998] However, the requirement to provide the Abstract Machines as priors means the agent is not learning this behaviour.

Hierarchical Reinforcement Learning structures use subsystems to perform subtasks of the main task. The agent is able to learn which subsystems are useful to the task, strengthening their hierarchical connection or frequency of use. The hierarchies are conceptually similar to those in Artificial Neural Networks, having the hierarchy used for skills required to complete the above task. This is proving consistent, with the lower levels being required for the higher levels to function. It is not clear how this can be formally defined in order to build a suitable hierarchy. Reinforcement Learning has been used to learn hierarchical structures, which is worth investigating to better understand hierarchical structures and their origins.

3.5 Learning a Hierarchical Reinforcement Learning Structure

Digney has researched the use of Q-Learning to allow hierarchical structures of actions to build a control system. Initially, the research began with an augmented Reinforcement Learning control node being produced called an Adaptive Behaviour Module (ABM) [Digney and Gupta, 1993]. It is part of a scheme named the Distributed Adaptive Control System (DACS). The ABM node is able to solve the walking of a quadruped robot in simulation with future projections on using multiple self-organising nodes [Digney and Gupta, 1993]. A hierarchical solution to the same problem is shown two years later with over half a dozen ABMs managing the problem [Digney, 1995]. The agent is able to function with severe hindrances, such as limbs being disabled. The control system is self-reinforcing but not self-organising, as the structure was provided. The work has not been applied to a real robot.

Later, Digney makes a first attempt at a self-organising hierarchy [Digney, 1996]. The abstract places heavy criticisms on the use of real robot learning, implying that the work proved too difficult to implement on a real robot. Q-Learning nodes are generated from features. Features are defined as some new state of existence from the robot's perceptions. Each Q-node is attempting to learn how to acquire the desired feature as inputs. Each node may invoke the call of other nodes or primitive actions to do so.

This forms a delegation hierarchy with the actuators at the bottom. Additionally, some bottom-up learning is provided. When an interesting event occurs, a bottom-up response is generated from that node. This can be analogised to instinctual linking of basic actions to state configurations naively, guiding higher level nodes on where to delegate command signals so a specific result can be achieved.

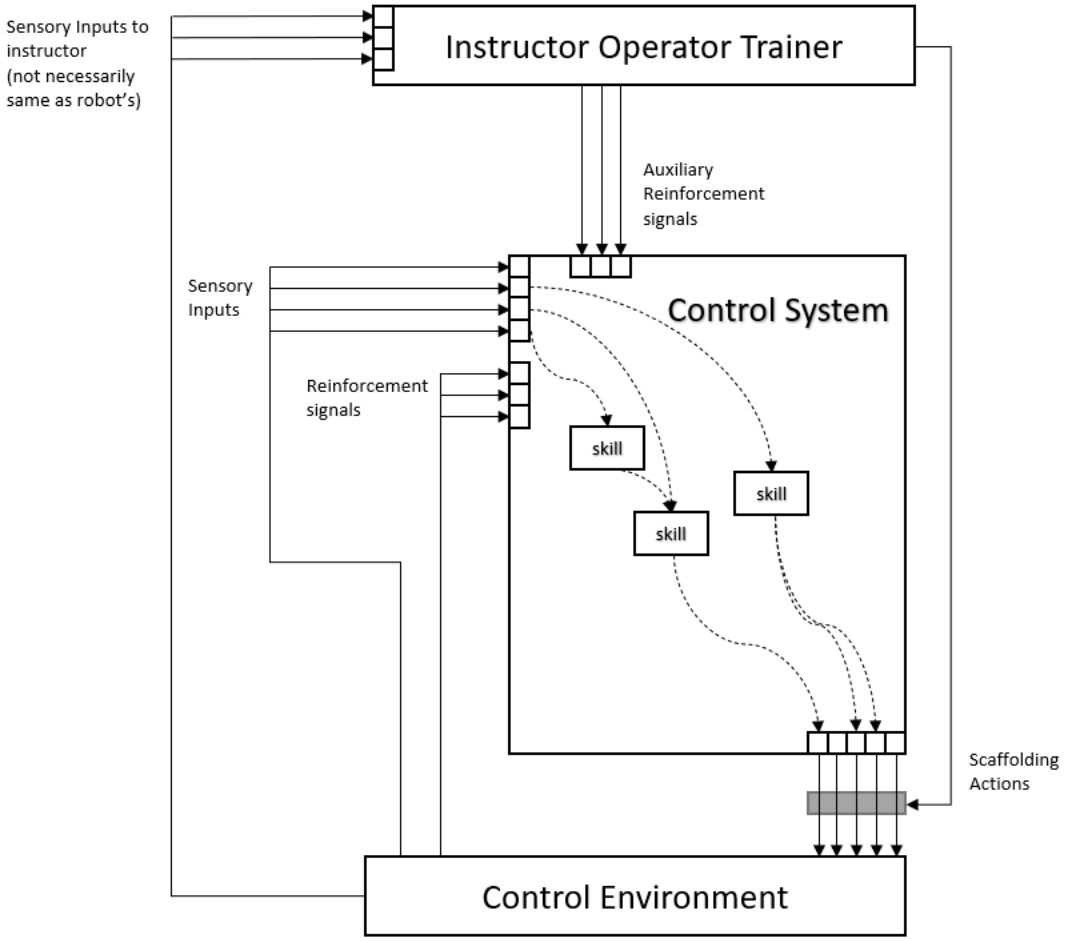

Figure 5: *A diagram showing the control system for learning a hierarchical system with Q-Learning, recreated from [Digney, 1996]. The control system contains multiple control subsystems that aim to enact some skill in the state space. The dotted lines represent possible connections, where the control system and subsystems experiment with various configurations which is critiqued on by reinforcement signals from the environment and the instructor. The instructor provides reinforcement signals and also provides scaffolding actions. Scaffolding actions override the choices of the controller, usually when the controller is going to make a poor choice in the early stages to guide it.*

The system is able to solve a simple simulated travelling problem of moving to one of two locations in a small grid [Digney, 1996]. All nodes were provided to the system for all state configurations, which make fifteen states for the different grid positions. The algorithm eventually abandoned most of these nodes and opted for the four most

important to solving the task which were the four corners (as both coloured squares were in the corners). The agent was able to hierarchically learn how to get to blue nodes and green nodes via delegation. Additionally, the algorithm possesses some elegance. The agent would be randomly placed in either world 1 or world 2. In world 2, the colours of the squares are swapped which means learning a strictly optimal policy is not possible. What the agent learnt to do was go to one square and if it was the wrong one then move to the other. This is an elegant solution which according to Digney was notable in Reinforcement Learning applications [Digney, 1996]. Importantly, the agent learns the hierarchical connections between existing nodes which extends on other hierarchical Reinforcement Learning applications.

Work extending the 1996 implementation of DACS demonstrates the control structure on a problem with several goals [Digney, 1998]. Additionally, there is a wall in the centre of the state space providing difficulty to reaching the goal spaces. Besides the state space complexity being increased, another notable change is in the acquisition of control nodes. To recall, control nodes represent features which is some state of the system. The node's job is to attain said state space using Reinforcement Learning. In this iteration, the states are not all pre-defined but selected when a state is found that has some meaningful effect on reward. This is defined as a state which occurs more regularly in high reward trajectories through the state space.

This is notable, as real world experience is now dictating elements of the hierarchy. The results showed success with multiple key locations producing controller nodes, including a hidden node [Digney, 1998]. However, the state space is simple and has foundation actions which are binary in nature. One can still question whether such a system is able to move into a continuous time environment and perform or successfully identify features. The hierarchy is defined through competence of enacting the higher subsystem's control objective, but this is only tested in an environment that requires sequential selection of nodes to activate. This does not transfer well to control problems, which often require continuous references and a hierarchy that isn't based on the current context of the agent. Furthermore, reinforcement learning is notably slow to learn. While the principles underpinning this work, there is no evidence to support this deriving hierarchies in complex control systems in a reasonable time.

Through studies of biologically inspired control systems, the agents have some clear hierarchies that have lower subsystems they require to complete their tasks. Some learning is undertaken to identify these, but the learning is in simple situations and is through exhaustive testing. It is clear that hierarchies are born from subsystems requiring the use of other subsystems, but an elegant method of finding this that could be methodologically undertaken is not present.

3.6 Structural Control in Psychology

An approach which has seen little attention with behavioural scientists or AI researchers is the perceive-act cycle and studies of perception. Inspirations to AI and control have largely originated from symbolic approaches or stimulus-response models.

Stimulus-response models are claimed to leave out a critical element of the cognitive experience, by separating the stimuli and response [Powers, 1974, p. 41-43]. Powers comments on the requirement of an agent to act continuously. Action-selection AI require explicit discrete definitions of actions, which is difficult to translate to a continuously responsive action. Powers therefore argues against this being a suitable model of building a robust controller. Put simply, programming an agent to behave continuously is required

to operate in a continuous world.

Stimulus-response models by nature are open loop. As a result, they struggle to deal with disturbances in a real-world setting without anticipating them first. Living agents have the capacity to account for future error and deal with current error, but how these are achieved remains up for debate. Powers posits that behaviour that seems anticipatory is achieved through controlling the right perceptions in the environment, rather than performing complex calculations to identify future error.

Various experiments note the accuracy of perceptual continuous behaviour in living organisms such as the spotlight task [Taylor and Birmingham, 1948]. A spotlight is on a target location and the agent must counter a random disturbance to the position of the spotlight. A comparable but more difficult task is the catching of a ball, where a living agent's approach is successful. The agent would not practically have time to calculate the location that the ball would land. If one were consulted as a test subject mid-test as to where the ball might fall, one would certainly not be able to provide an equation that would identify where it would fall. One could, however, state that one would need to head in a specific direction or that the ball will land in a general area. This supports the closed-loop methodology, indicating that simply countering disturbances is a psychologically accurate approach. Evidence has been produced contesting the stimulus-response view of the generation of behaviour and the benefits of a perceptual loop system [Powers, 1974, Grush, 2004].

Psychology has offered insight into behaviour of a cognitive agent that imbues robust behaviour, covering the continuity that should be desired in a cognitive architecture. Agent-based architectures that focus on the agent have lead to structurally adaptive agents that can handle changing environments [Dennis et al., 2014, Dennis et al., 2016], showing that agent-focused approaches that concern the perceptions of the agent merit investigation. Cognitive sciences have been applied to produce cognitive architectures which may contain cognitively-inspired hierarchies.

3.7 Cognitive Architectures; Theory

It is worth clarifying a common difference between hierarchical control as applied in cognitive systems and Control Theory. The distinctions are subtle but relevant to the principles of cognition, whereas an engineering application likely has little concern. The main distinction concerns the reference signals. An engineering solution sees this value as *external* to the system, whereas a cognitive solution requires this to be *internal* to it. An example of this in the case of a negative feedback loop would be the reference level of the specified value which the feedback loop has to maintain the input to.

A living agent such as oneself contains many 'references'. Such references might be how much milk one generally desire in their tea, how hard to pull the door when opening it or the exact muscular tensions required to pout sarcastically. These references may be *learnt* externally but are not manually retrieved from the external environment. The environment provides input signals which can be processed to produce a reference, but the reference value itself is produced internally.

Over the period of time that AI has been developed as a field, concepts of biology, philosophy, psychology and computing have slowly been introduced into AI. This effort has aimed to make AI more grounded in how cognitive agents function, particularly humans. A variety of long standing theories have been contested and these bring us to some important concepts that tend towards Hierarchical Control being a suitable model of cognition.

Dualism is the concept that the body and mind are to some degree separate, such that the mind is a non-physical entity, suggested by many prominent philosophers, including Aristotle [Hicks and Others, 1907], Plato [Duke et al., 1995] and Descartes [Descartes and Cottingham, 2013]. Consequently, this has formed a barrier on the analysis of the mind. Refuting the idea that the mind is something empirically understandable and bound by the physical laws which are assumed to restrain everything else caused a lack of practical and physical analysis of the mind. Wittgenstein contests the dualism argument to hold any weight on cognition with the Private Language argument, contesting the relevance of the separation of mind and body [Wittgenstein, 1967]. A summary of Wittgenstein's argument is that the separation of mind and thus some level of sensations and body is like the mind having a private language. This private language is one that only the user can understand but is untranslatable through bodily experiences. Thus it is a language that one cannot accurately convey and is private to the user. Wittgenstein concludes that it is not possible for such a language to exist, as it would require the user to not be able to understand their own language. Uncertainty was then shed on whether a separate entity to the body that has some significant effect can exist.

A large amount of philosophical and psychological study states the importance of perception and the interlinked nature of the environment and the agent. Powers shows the strength on perception, that the means of an agent's actions are to shape the agent's environment and the two are strongly interlinked [Powers, 1974]. Uexküll makes similar statements much earlier [Uexküll, 1934]. Furthermore, it is possible for a program to capably contain and simulate another program, but Uexküll contests the view that one cognitive agent can by principle simulate another. The Umwelt Theory he states means that one, as a human, could not be a bat and know what it is like to be one. The cues, triggers, perceptions and thoughts are all radically different [Uexküll, 1934]. Despite the underlying structural similarities that they may have, the individual functional parts can only go so far to define an agent. Most certainly, one cannot simulate what it is like to be a bat or a cat. One cannot even though they have a more computationally complex mind, because one is simply not a bat or a cat. Therefore comparing living agents to computer programs is shortsighted.

The final concern to be mentioned is the input to output nature of computers. Computers possess no concept of embodiment and no ties to their environment. Questions have been raised as to this approach being appropriate, in that presenting the interactive dynamic of the agent with the environment as an input to output model removes a large temporal dynamic in the agent's interaction. This has been alluded to in the previous statements regarding what Powers and Uexküll have stated. Input to output AI are still used commonly, even against the theories displaying them as unauthentic to living agents.

These theories show that the input to output nature of AI removes important dynamics of a living agent. Also, revelations in theory of the mind have been presented as a foundation for suitable cognitive systems. This advancement of how cognition works has been critical in psychological and philosophical attitudes towards intelligence and AI. Attempted implementations of such, known as cognitive architectures, began once these important revelations had been made. Leading cognitive architectures are largely hierarchically arranged [Verschure, 2012, Ramamurthy et al., 2006]. Examining cognitive architectures shall be undertaken to learn more about hierarchies in living control systems.

3.8 Cognitive Architectures; Implementations

Distributed Adaptive Control (DAC) is a hierarchical system of groups of neurons, structured to perform several different functions based on their level (contextual, adaptive, reactive) [Verschure et al., 2003]. DAC is an entirely non-symbolic architecture, relying on neuronal processing and representation of 'events' without further specifying any concepts. DAC has been applied to robotic tasks of a cognitive nature, such as social interaction [Metta et al., 2008] and foraging [Renno-Costa et al., 2011] with other applications discussed [Verschure, 2012].

DAC is built around managing all the concepts of the Mind Brain Body Nexus (MBBN), balancing behaviour, perception and internal processing [Verschure, 2012]. The reactive layer handles basic reactions of the system, which is the quickest layer to respond to an external happening. The adaptive layer acts as a learning agent, identifying the outcome of particular events. When the adaptive layer predicts correctly, the contextual layer is activated. The contextual layer records these events in the adaptive layer (which are of a sensori-motor nature) and stores them as a temporal order of the sensori-motor events. As such, the final layer develops some notion of temporal ordering and context to actions, resulting in claims of this system bearing the underlying mechanisms that animals and humans engage in goal-oriented behaviour [Verschure, 2012].

DAC has been applied to a selection of robotic tasks. The results concluded that the model depends on dedicated task definitions [Renno-Costa et al., 2011] which doesn't grant the adaptivity desired from such an architecture. The non-symbolic nature of the architecture is mentioned as a key point of DAC, stating that it is superior to symbolic alternatives such as Subsumption Architecture [Verschure, 2012]. Despite this claim, this has not been tested. In general, the symbolic vs non-symbolic case is an open argument, due to both being applicable in different situations.

Non-symbolic systems, while taking inspirations from a human's cognitive system, are intuitively harder to work with and understand. It is difficult to understand the activity of DAC, making it hard to build and debug. The higher levels (such as reactive, adaptive and contextual control) are defined but how these interact with actuators is not explicitly specified. The concept is to understand how the human brain implements this through neurological research and then use models of those to fit the parts of the system. Interestingly, this is a cognitive architecture that claims the more specific schematics of the brain fill in the implementation level details. This in itself is respectable, as results have shown cerebellar controllers producing anticipatory behaviour that fits into the adaptive-reactive model of DAC [Herreros and Verschure, 2013]. However, there is no justification the overall model functions with all said parts, or even where all said parts might come from. While results are promising, the architecture is far from a complete implementation.

To summarise the DAC system, a variety of claims have been made that have been untested, but with backing of neurological research supporting DAC. It cannot be called superior to a symbolistic architecture by the evidence presented. The system does however show the ability to learn in basic behaviour in real environments, but has only been tested on niche tasks without much need for generalisation or adaptability [Renno-Costa et al., 2011]. The hierarchy is clearly defined, but how this hierarchy was derived is not specified.

The Global Workspace (GW) architecture was proposed by Baars [Baars, 1988, Baars, 1997, Baars, 2002] which bears similarities to Pandemonium theory [Selfridge, 1958]. GW is a theory of information transfer as an architecture based on knowledge about

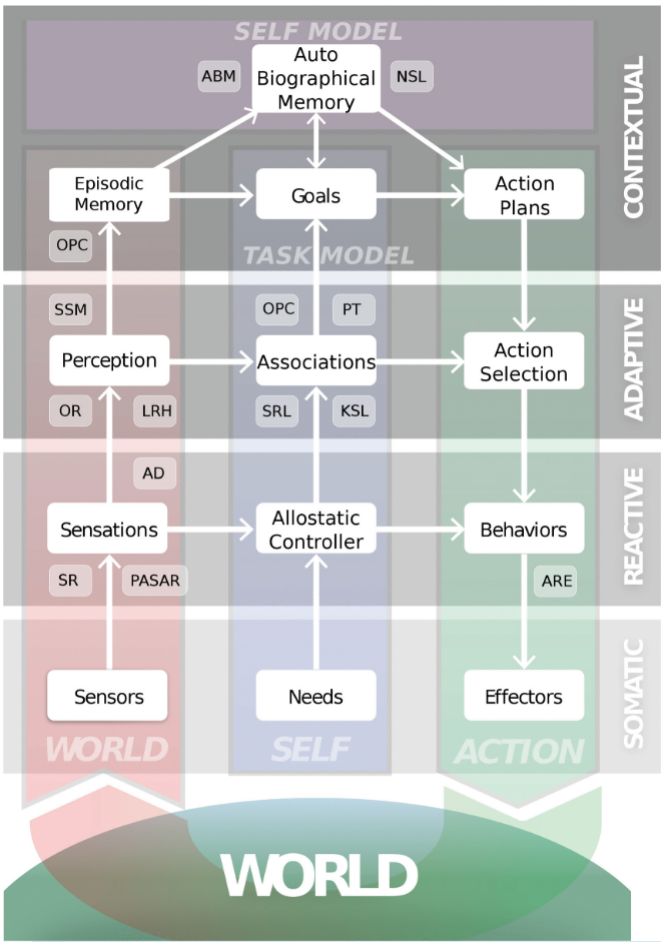

Figure 6: *A diagram showing the modules and processes in Distributed Adaptive Control, reprinted with permission from [Moulin-Frier et al., 2018]. The Somatic processes handle the actuators and inputs, with the layers handling progressively more complex tasks requiring more deliberation.*

consciousness. Consciousness is one of the most abstract concepts involved in the brain, bearing no agreed formal definition. The Global Workspace architecture aims to account for the conscious condition within the mind and produce it through its architecture.

A technical understanding of how the mind causes consciousness is non-existent but research has extended to the study and features of consciousness (see Shanahan's 'Embodiment and the Inner Life' chapter 3 for an overview [Shanahan, 2010]). Consciousness, functionally, has been stated to be the mind's mode of flexibility and novel thoughts [Baars, 1988, Searle, 1992], allowing the mind to produce novel ideas and solutions where the current state of mind doesn't allow a solution to appear.

The Global Workspace itself is the central node of the architecture which aims to mimic the conscious condition. Surrounding the global workspace are many specialist processes, which compete for access to the global workspace so their requirements can be broadcast to the other specialist processes. Already, analogies can be drawn to research on the Basal Ganglia, notably Redgrave's asserted architecture [Redgrave et al., 1999] and Houk's theory of DPMs [Houk et al., 2007].

The global workspace acts as a melding pot of ideas and concepts, allowing competing processes to be combined or applied to one another to produce new concepts based on the requirements. Specialist processors compete via relevance to the situation. It is noted that this accounts for a variety of features of consciousness [Shanahan, 2010], including the limited capacity of consciousness [Baars, 1988].

Although the theory presents a foundation of psychological research on consciousness, the technical details of GW are limited. How the global workspace 'combines' processes is not defined, nor are any of the finer details of any element of the architecture. On the contrary, Shanahan has produced an in depth study of GW and theorises how it might work in more detail [Shanahan, 2010], albeit still not enough to technically implement.

Learning Intelligent Distribution Agent (LIDA) is a GW implementation. LIDA is a cognitive architecture which uses modules for common cognitive processes (such as collections of different memory modules and action selection) and hosts a global workspace which allows them to be combined to produce novel processes [Ramamurthy et al., 2006, Franklin and Patterson Jr, 2006]. The analogies to consciousness from the GW are maintained, proposing that LIDA is a model of conscious behaviour. Furthermore, studies into the neurological support of LIDA have been produced [Franklin et al., 2012].

The LIDA implementation works on a cognitive cycle, which results in a perception of the surroundings being taken in and various modules sending codelets (which one might analogise here to concepts) to the workspace. The global workspace has a queue of these codelets and aims to merge or select them based on relevance and criticality. The coalitions of codelets is considered based on the working model and representation of the world, as contributed by the memory modules of the agent. Appropriately relevant coalitions is be sent to the global workspace which broadcasts to all the external modules [Franklin and Patterson Jr, 2006]. This influences their impact on the system and the actions the agent takes.

LIDA is explained in a variety of publications detailing the implementation. Analysis has been presented that LIDA is a suitable model for Artificial General Intelligence, through great reusability and generalisability sufficient to become the standard cognitive architecture [Snaider et al., 2011]. LIDA is claimed to be able to perform high-level management tasks [Baars and Franklin, 2009] but evidence is difficult to find. In fact, no implemented tasks have been published.

LIDA proves that the GW theory is implementable as a system. It is debatable whether LIDA is producing conscious behaviour. The more interesting point is to declare whether it can perform behaviour that would require high level cognitive skills, such as consciousness. Without evidence, the system is difficult to analyse in further detail and isolate the uses and weaknesses of the system.

The Global Workspace and the implementation LIDA both show evidence supporting their validity as a model or theory of consciousness. Aside from LIDA, GW has been tested as a model of consciousness separately, finding comparable results [Robinson, 2009]. Both stand on a foundation of support and research, but implementations of cognitive architectures are difficult. Their complexity makes artificial generalised intelligence a monumental task to model. LIDA or other GW implementations has not demonstrated conscious behaviour that is theorised to be possible with this architecture. The architecture represents the competitive model seen earlier when reviewing the Basal Ganglia, using context to derive an appropriate delegate for the task. However, how the Global Workspace selects or merges appropriate tasks is not clear.

Overall, a brief inspection of cognitive architectures has found them without clear

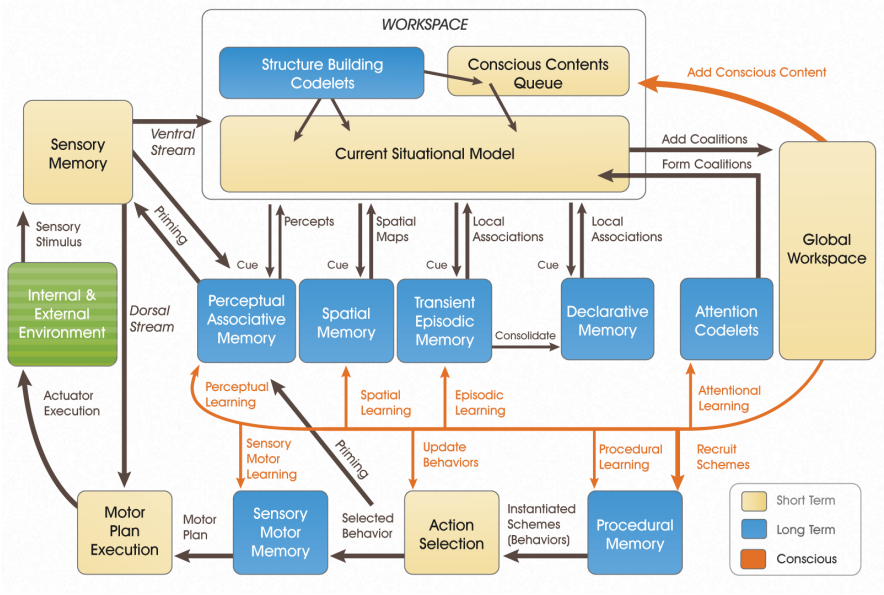

Figure 7: *A diagram of the LIDA cognitive architecture reprinted from [Franklin et al., 2014] with permission. The diagram emphasises the cycle that takes place in the architecture.*

answer of how a hierarchy is derived, or how it can be understood that one subsystem requires another. The architectures are rigidly defined, but little detail about how this can generally apply to other control tasks to show how control hierarchies could be made. DAC does demonstrate a hierarchical ordering based on types of input processing, but this does not inform clearly how it could be applied. What is a controller handling reactive control and what is handling adaptive control is not defined enough to produce a hierarchy from it.

3.9 Perceptual Control Theory

3.9.1 Overview

PCT was devised by William Powers who produced the Thesis 'Behaviour: The Control of Perception' in 1974, theorising that agents control their perceptions with closed-loop control, contrasting many existing theories in psychology. PCT is an implementation of how cognitive agents control their environment through distributed and hierarchical control.

Powers criticises the input to output model of cognitive processing that is commonly employed in Psychology [Powers, 1974]. Powers posits that input to output models, like stimulus-response, do not always frame a control problem appropriately and do not

represent how human agents solve problems. Consider a human agent that is attempting to catch a ball. It is rationalised that the agent does not calculate exactly where one must be to catch the ball and then initiate the required action, but rather the agent is constantly monitoring their perceptions and engaging in actions which bring their perceptions more in line with what they require. PCT aims to provide a framework for how problems should be thought about and represented, not just solved.

A single Perceptual Control Unit is a negative feedback loop. This perceptual signal is compared against a reference signal internal to the agent where the difference of these two signals produces an error signal. The error signal is transformed into an output signal which invokes some effect on the external world, aiming to reduce disturbance. This is not novel; engineering applications have utilised this solution for many years [Ashby, 1956]. However, PCT shows correlations with the behaviour of living agents [Powers, 1978, Bourbon, 1996, Marken, 2006], indicating the similarity of PCT's behaviour with respect to real agents.

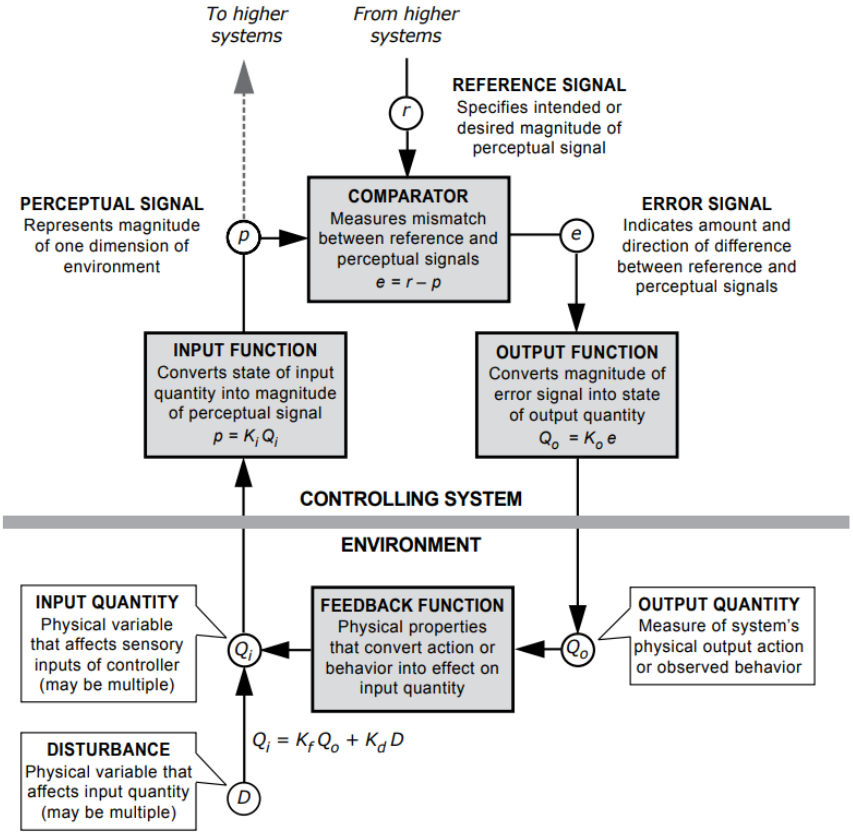

Figure 8: *An individual PCT loop reprinted with permission from [Powers et al., 2011]. Note that this example in a hierarchical PCT structure would be a node of the lowest level, one connected to the actuators of the system. The higher nodes would produce the reference signals for nodes in lower levels.*

3.9.2 Hierarchical PCT

HPCT is the use of multiple PCT nodes hierarchically to form a larger control structure. Demonstrations show elegant compromise of parts of a robotic system when considering

a simple goal [Matic, 2014].

Powers theorised the use of PCT could, with enough levels, model human thought and cognition [Powers, 1974]. The lower three levels cover basic perception, the next three cover basic patterns and the final three cover conceptual entities which guide and apply the appropriate patterns to the relevant situation. Further details of how memory, imagination and adaptive behaviour are covered as well but with few to no technical details. Importantly, Powers states that the levels do not finish at the 9th, it is merely the case that Powers could not think of further levels conceptually and further analysis and knowledge would be required to identify higher levels.

Since the origin of PCT, attempts have been made to test the system and engage it in control tasks. In 1990, a HPCT controller was built in a spreadsheet program [Marken, 1990] to manage four arbitrary perceptions. Important results of this work were the effectiveness of PCT, showing that the system could control the perceptions competently. Also, less obviously, this work shows the ease of producing a HPCT system. The system took one hour to build in a spreadsheet program.

The HPCT controller produced by Marken is a hierarchical structure with generalised levels. However, it is not known whether this structure and the theory behind it can be used to generate structures. It is possible Reinforcement Learning could account for the optimisation of said structure, once it has been generated. The answer to structural generation and reoptimisation in Perceptual Control Theory is a process called reorganisation.

3.9.3 Reorganisation of a HPCT system

The original concept for reorganisation in PCT was introduced by W.Powers in 1974 [Powers, 1974]. Reorganisation is the process which by the connections between nodes modify the weight of the connection based on newly learnt situations. Powers describes a separate reorganisation system, which based on critical learning criteria adapts the main system's connections.

Configuring systems to address perceptions and adapt often consider predictive models as the solution, implying the agent predicts what a perception is about and deduces how to control it from that. However, solutions have emerged that show that prediction is not the only way to address learning the control of stimuli [Ashby and Gott, 1988]. This inspired further approaches by Marken and Powers, looking at reorganisation as a random trial and error process. This proved effective in modelling E-Coli performing chemotaxis, showing that proportionally random movement based on error leads to reorganisation that balances over time [Marken and Powers, 1989].

Recently, adaptivity has been a large focus of neurologically inspired approaches, such as Reinforcement Learning and neural networks [Niv et al., 2012] [Nagata and Watanabe, 2011]. Live running of PCT in action with reorganisation has been shown *via* video demonstration [Young, 2015], but with no clarity as to the technical details, the area is still vague.

The most detailed results of reorganisation is a random generation feedback-oriented algorithm [Powers, 2008]. The algorithm intuitively changes the weight randomly with a chance of changing based on the current error, an extension on Marken and Powers' work previously [Marken and Powers, 1989]. So, a structure which currently has little error is unlikely to change. The work shows the algorithm resisting disturbances and bringing the reference and input signals closer together. However, this method is based on immediate experience. In a complex environment, a single episode of no error or large error could

occur. In large hierarchies, all that it would take is one unbalanced node to result in high error. This problem is analogous to the Credit Assignment problem mentioned in Hierarchical Reinforcement Learning [Kaelbling et al., 1996, p. 251]. Without a clear way of how the error is localised to a point where the algorithm can decide where reorganisation might need to occur, the algorithm risks randomly changing successfully balanced nodes and losing progress.

Reorganisation fits the benchmark for optimising an existing structure but does not specify how levels of a hierarchy can be derived. The process of acquiring a control structure is left unanswered. How PCT has been applied to problems shall be investigated to learn about PCT structures and their origins.

3.9.4 Applications of PCT

Perceptual Control Theory is currently being used in a variety of fields, mainly in psychology through experiments to support the credibility of the biological accuracy of PCT and to aid in the management of psychological distress [Higginson et al., 2011]. The most notable application of PCT to managing psychological distress is the Method of Levels behavioural therapy [Carey, 2006]. The applications to psychology are prevalent, forming a stronghold for PCT. In addition to psychology, models of human behaviour have been produced with PCT, attempting to simulate human behaviour [Zhao and Cziko, 2001].

Sociological applications of modelling rational agents are apparent. As well as PCT being claimed as a unifying theory of psychology [Marken and Mansell, 2013], applications in sociology have resulted in behavioural models for phenomena such as rioting [McPhail, 1994].

In business and marketing, PCT has been used for modelling agents but also understanding leadership and behaviour in a marketing work environment [Forssell, 2008]. Work was put forward by McClelland who simulated agents with conflicting goals in investing in stocks. Naturally, a stock investor has two conflicting goals. Buying stocks reduces money and to gain money one must sell stocks. As such, there is no clear behavioural divide to program these possibly conflicting goals. The model generalised to the behaviour which followed the principle of selling when the price was going up and buying when the price was going down [McClelland and Worthington, 2010]. All six PCT models produced bar one beat the control. This presents rare evidence of PCT performing competently in real practical settings. The hierarchies used are in abstract and simple environments, with little detail on how the hierarchies emerged.

McClelland critiques their own work, citing the simplicity of the model and the lack of realism in the testing, as well as the limitations of the inputs of the model [McClelland and Worthington, 2010]. As such, the work done has limiations but still indicates effective transferability of PCT to other fields.

Vancouver produced a model of a PCT system solving a specific task management problem [Vancouver and Putka, 2000] and drew analogies to the behaviour of the participants. The human participants responded similarly to the PCT agent when the system was altered. However, this analogy without a tested model can only be taken so far. While the points made by this article are elegant and perhaps correct, they don't validate PCT so much as draw similarities that might be coincidental.

Applications in Speech Technology by Moore devised the system 'PRESENCE', which uses the PCT methodology to answer speech interaction [Moore, 2007]. PRESENCE uses hierarchical PCT nodes to solve the problem of communicating via speech. Important focus is placed on the key concepts of PCT, that the system is built to understand the

needs and intentions of both parties in a communicative interaction. As such, many speech systems model an open-loop system which results in a disparity between the environment and the agent, causing the agent to have to compensate for this. Moore proposes the PCT system by nature of being closed loop is a more suitable system to solve this problem.

The experimental results are limited, but the agent is more natural and engaging, supporting the notion that closed-loop systems follow a more humane dynamic. This is a success of the model with room for further studies and expansion to a fully immersive machine for human interaction.

Further work is ongoing in speech, but from a more technical side. Models of the vocal tracts have been produced with the system using PCT to train the vocal system [Hofe and Moore, 2008]. A cognitive architecture for speech-based interaction was devised with a synthesiser robot produced using PCT that can increase the intelligibility of spoken words [Moore and Nicolao, 2017]. An architecture for a communicative agent is produced, showing how two symmetrical agents can communicate in a PCT framework [Moore, 2020].

Unfortunately, many of the areas which PCT are claimed to be used in (even with demonstrations available to show the results) do not have publications or papers to present as evidence for experimenting purposes. The psychological applications of PCT have been well documented, but areas of relevance to this Thesis are short of sufficient scientific certification. Overall, applications in psychology for PCT have shown useful application to human agents as well as the ability to mimic their behaviours in real environments. However, the structures are largely predefined with little detail of where they emerged from. However, PCT has been applied to robotics, which can be studied to identify where control structures emerge from in more detail. Approaches similar to PCT have yielded some success in robotics. Work on grasping an object follows a similar principle using methods very similar to PCT [Raspopovic et al., 2014] with earlier work supporting the idea in theory [Rodrigues et al., 2009].

A six legged robot was simulated with HPCT to successfully perform basic tasks [Kennaway and Date, 1999]. The robot, in simulation, was able to walk across a plane going up and down small sets of stairs to reach a pre-set goal. HPCT is compared to Subsumption Architecture, noting the difficulties of SA in managing conflicting behaviour. PCT is also compared against PID in simulation, where PID controllers fail to stably stand. The robot (named Archy) is able to navigate this space without any internal contextual knowledge of the space around or itself. These results are impressive for the simplicity of the system and lack of internal knowledge. However, as stated, a simulation is not as complicated as the real world opening this up to criticism.

In 2005, a real version of Archy was implemented. The robot was never able to walk fully, but was able to deal with minor disturbances to allow it to stand level [Lippett, 2005]. The robot could not do what the simulation could, seemingly due to the limitations in practicality of building the robot rather than the direct limitations of PCT. The project was not finished but the PCT implementation showed some responsive reaction to disturbance, albeit not to the extent of the simulation. This work has continued with recent publications on the effectiveness of PCT to control robots, through Living Control Systems III [Powers, 2008] and IV [Mansell, 2020], with key chapters and topics on the control of robotics [Johnson et al., 2020].

Kennaway produced a four-tier HPCT solution to solve the Inverted Pendulum problem [Kennaway, 2004], with an implementation of this later published [Young, 2020].

The Inverted Pendulum problem is usually modelled with a cart with a large pole extending above it and a weight on top of the pole. In the most basic version of the problem, the pole and weight can fall forwards or backwards and the cart must move forwards or backwards to keep the weight directly above the cart. This problem is analogous to any wheeled vehicle with a heavy weight to balance above the cart.

This particular implementation contained detailed analysis on why the particular hierarchy was chosen, which is useful information. PCT and HPCT both seem generalised due to the inspiration from real agents that they employ in their approaches. This bodes well on them being the basis and inspiration of a diverse self generating hierarchical algorithm. The quote that details the origins of the hierarchy is:

> *"If we had an actuator that could set the bob immediately to any desired position, no control system would be necessary. We don't have such an actuator; but if we had one which could set the pendulum's horizontal velocity, we could use this to control the position... We don't have such a velocity actuator, but if we had an actuator that set the bob's acceleration, we could control the velocity... The acceleration is proportional to the pendulum angle, which is proportional to o = b-c, where c is the position of the cart. So we can set the acceleration by setting o. We cannot set o directly, but we could control o if we could set the cart's velocity... We cannot set c directly, but we could control it if we could set the cart's acceleration... Finally, we can set the cart's acceleration by applying a force to the cart, which by hypothesis we are able to do."*
>
> — Kennaway, *2004 contributions to Powers [Kennaway, 2004] [Powers, 1999][p. 9]*

The position of the bob depends on the velocity of the bob, which depends on the acceleration of the bob, which has some proportional relationship to the position of the cart, which depends on the velocity of the cart which can be controlled by the cart's motors. It is not possible in this problem domain to control the position of the bob without accounting for or controlling the velocity of the bob. This logic repeats down the hierarchy, showing that they are a necessary part of the control and cannot be controlled without it. Each level identifies a key subcomponent to controlling that problem, eventually leading to an actuator that is simple to control.

The implementational method followed in Kennaway's work is consistent with the other hierarchies examined in this Section, but this analysis from Kennaway explains how it can be broken down into steps. The hierarchy connects a high level goal to low level actuators, producing a complex transform through multiple functions. Understanding a subsystem's control function and which subsystems could be part of that control function is the goal. A subsystem's control function depends on what input it is controlling, providing the required context.

A consensus among Living Control Systems shows a requirement for the completion of the control task being the criteria for a hierarchy. Perceptual Control Theory identified how this requirement can be progressively understood, by identifying subcomponents of the control function. This intuition allows multiple avenues to be explored for producing a methodology. A methodology could be derived to simulate this, detecting subcomponents of the control function. Alternatively, the method could be enacted in reverse to

find functions that have no subcomponents, or only subcomponents that exist in the hierarchy. Enough information has been attained to move to producing a methodology for automating the production of hierarchies.

Classic Control Systems proved lacking on detail with regards to how hierarchies are derived. Living Control Systems were investigated to identify whether controllers inspired by real agents allow further understanding of hierarchies. Living Control Systems showed that one subsystem requiring another is the semantic requirement for a hierarchy. However, it was PCT that answered the question on the derivation of hierarchies.

PCT hosts an abundance of capable and elegant controllers, unlike many other studied areas that struggle with the competence of their hierarchical structures. From control to psychology to resource management, something about PCT and their hierarchies works effectively. PCT's natural consideration for the perceptions a problem has and how they relate to one another captures important intuition on how hierarchies are built and thus results in PCT engineers choosing appropriate hierarchies. PCT across the board has simple tools that provide powerful effects, relying on the distributed nature of PCT nodes to produce complex behaviour together. This is behaviour that other architectures have aspired to but not achieved. Kennaway's work detailed clearly the intuition in building hierarchies that was used, which seems consistent with existing effective hierarchies. This next chapter of this Thesis utilises the principles found in work from PCT to produce a methodology that automatically derives hierarchies, using control subsystems that are compatible with PCT.

4 The Emergence of Hierarchical Structures

It is evident that hierarchies are formed when the control of one input depends on another. A formal definition can now be produced and whether this can be used to produce hierarchies automatically can be investigated.

4.1 What Constitutes a Hierarchical Relationship?

To better understand hierarchies, a dependency must be defined. It must convey the semantic information needed, while being centred around how a hierarchy works in hierarchical control. This Thesis defines a dependency as follows:

> *"Dependency: The control of one input to the system requiring the control of another input to the system"*

This definition warrants some explanation. Let us assert that input A is dependent on input B. This means that no effective control of A can occur without controlling B. The controller controlling A should be hierarchically above the controller controlling B. If there were no hierarchy, the controller controlling A must find some way of accounting for the lack of control over input B.

This is important to understand due to common practices with controllers and putting hierarchies into context. One could critique that it is possible to control a particular input without what is known to be hierarchically below it. An example may be a positional controller of a wheeled robot in two-dimensional space. This is a simple problem that one can model with a PID controller, for example. One might say that this invalidates the definition posed or the need for a hierarchy. In fact, this is an example of the poor practice of not considering the hierarchy properly.

Consider the same example, a wheeled robot in a two dimensional plane being controlled with a single controller. Assume the input to the controller is only the position of the robot and the output leads to the actuators that move this robot forwards or backwards. If one replaces the PID controller with a P controller, the controller would significantly weaken in their control of the task. With just a P controller, the controller only reacts proportionally to the difference in position. This means that when the error is small and the robot is close to their destination, enough force is not generated to move the robot and build velocity. If the robot is far away from the reference point, overshoot is possible as the proportional error does not account for the ongoing velocity which requires force to change. Without any understanding of the other inputs required, the I and D in the PID controller are implicitly accounting for the velocity and acceleration of the cart as well as how this affects the amount of torque needed. The dependencies still exist, even if one is not modelling a hierarchy around them. So, the controller must account for them or suffer. In this example, it is simply the case that a controller that accounts for these was picked. A nested arrangement of P controllers could achieve this if the dependencies were accounted for, as seen by the investigations into the inverted pendulum problem in this Thesis and elsewhere [Kennaway, 2004].

Thus, depending on the inputs and actuators given, the theoretical hierarchy that is commonly used to solve a problem may not be applicable. This explains why different agents controlling the same problem may require different hierarchies, as how they interact with the world is different if they have different actuators or inputs. An example

is how robots require torque changes in their joints, but humans have more complicated mechanisms with muscle tensions. A hierarchy is therefore based on the individual agent.

A control subsystem is a function which performs some transformation on the input. A control system which is comprised of multiple control subsystems is a selection of functions applied to each other or the output. When a control subsystem depends on other subsystems, the higher subsystem is requiring them to enact their function on its output. The higher subsystem is thus only part of the function that produces the output suitable to achieve the desired task. Thus, in a controller with functions f and g, input i and control output u,

$$u = f(g(i)) \tag{2}$$

control function g depends on f. This is because the output of g is processed through f to become u which is the control output. One might ask what the point of having functions f and g, when one could simply have a single function. This instead gives two simpler transforms to optimise, rather than one more complex one. Any hierarchical controller could be merged into a single control subsystem that is just a larger function of the whole. However, the separation of the functions allows easier management of control system.

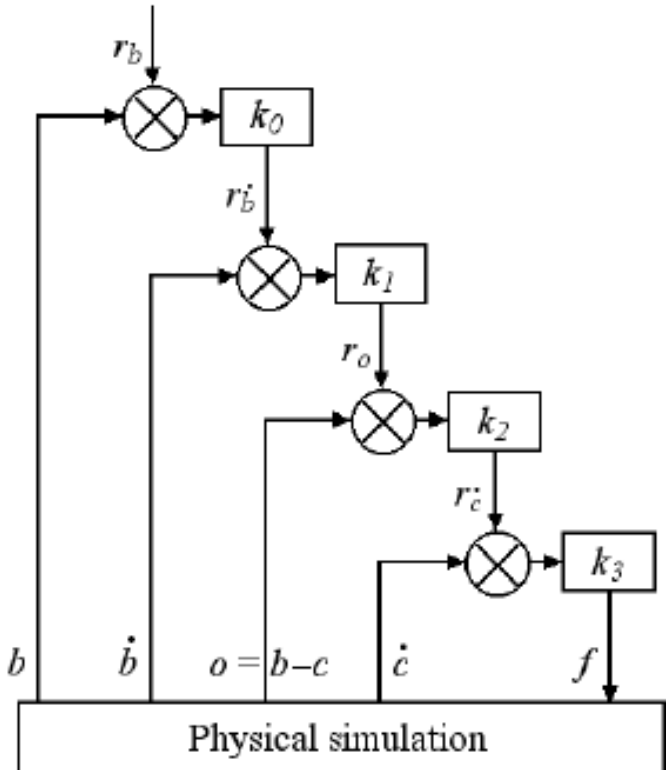

Figure 9: *The control diagram for the inverted pendulum problem, reprinted from [Kennaway, 2004] with permission.*

If a control subsystem is assumed to be a simple feedback controller, Kennaway's inverted pendulum solution is mathematically shown in equation 3.

$$u = k_3(k_2(k_1(k_0(r - b) - \dot{b}) - o) - \dot{c}) \tag{3}$$

Where k_0 through k_3 are parameters, r is the reference goal for the system, b and $\dot{b}$ are the position and velocity of the bob respectively, $\dot{c}$ is the velocity of the cart and o is the difference in position between the bob and cart, which is effectively the position of the cart. The control diagram for this is shown in Figure 8.

What can be seen in the equation is that the higher level function is contained within several other bracketed functions. This mathematical description is consistent with the definition of a dependency put forward in this Thesis. If the control output of one control subsystem goes into another control subsystem in order to achieve its objective, then it is a hierarchy. Therefore, two concurrent processes which compete to achieve their objectives would not be.

This allows a clearer understanding of what a hierarchy is from a structural and functional sense, rather than relying on a behavioural definition. The confusion comes where layered systems are posited to 'require' the behaviour of the other layers to be effective, but 'require' is subjective. Behaviourally, a layer in a layered system could require lower levels to establish the right context in the environment for the higher level to do its control task appropriately. However, this is not a hierarchy as the task enacted by the lower level is not part of the control equation of the higher layer. A hierarchical subordinate should enact part of the control function for completing the control task of the hierarchical parent. If two control functions are not nested within each other in some way, it is not a hierarchy.

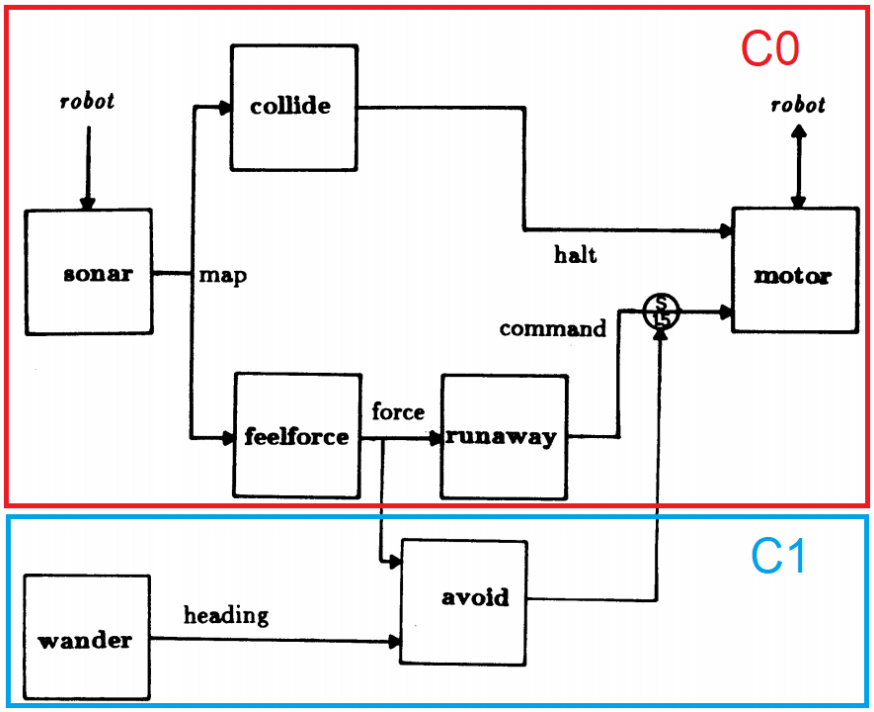

Figure 10: *A diagram showing the implementation of the zeroth and first levels of Subsumption Architecture, reprinted from [Brooks, 1986] with permission.*

Subsumption Architecture caused confusion when this Thesis reviewed whether it was a hierarchy or not. Examining the inputs lends itself to being a problem that could suit a hierarchy. One cannot wander effectively without avoiding objects. Importantly, this is required within that controller and not as an alternative layer. If they worked separately,

the controllers would often conflict. A wanderer would keep pointing the agent towards a new direction that the collision detection would deter them from. Figure 11 shows the zeroth and first layers of the 1986 implementation of Subsumption Architecture.

The upper controller influences the lower controller, but it does not directly send a reference to it. However, the behaviour of the higher controller decides whether the lower controller gets inhibited. This is what makes this confusing to analyse. When analysed as a set of cascading functions, it is clear that Runaway and Avoid are not hierarchically linked. Neither control equation for either function contains an output of the other. However, both supply output to the subsume node. What this means is Runaway and avoid are not hierarchically related, but they are both hierarchically dependent on the subsume node to mediate their output. The system is therefore a series of layered behaviours merged using a hierarchical control structure. This is shown in Figure 12.

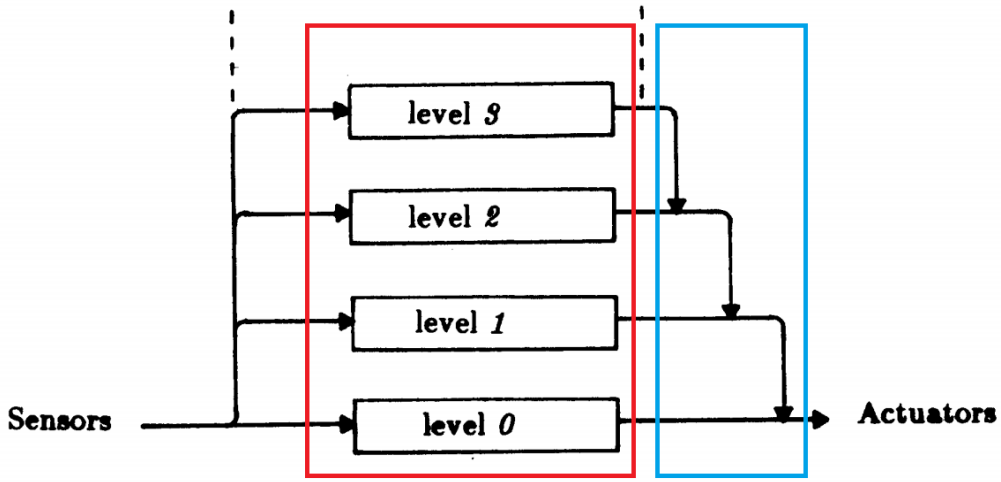

Figure 11: *A diagram of the generalised Subsumption Architecture, reprinted and modified from [Brooks, 1986] with permission. The box on the left in red highlights the layered part of the system. The box on the right in blue indicates the hierarchical part which selects or merges the outputs from the layered section to produce a single output.*

It is compelling that the provided definition produces a clear answer through understanding the control functions of the different nodes. It is also interesting that the definition shown here allowed a system to be defined as part hierarchical and part layered. This makes sense, as systems should be able to have aspects of both. John Doyle's hierarchical methodology, referenced earlier in this Thesis, does not conflict with the definition of a dependency in this Thesis, supporting both the dependence on what is below as the need for delegation to lower controllers through interfacing.

This definition of a dependency explains existing hierarchies effectively and fits with existing work. It provides an alternative to behavioural tests for identifying whether something is a hierarchy, as these can all depend on the controllers. From this definition, a principled methodology can be made to allow the generation of hierarchical structures through analysing dependencies.

4.2 Theorising the Automation of Generating Hierarchies

Now that a dependency has been defined, one can theorise how this can be utilised to produce a methodology which derives hierarchies. An agent must go from a bundle of inputs and actuators to a full control system, which means the answer is required to be derivable from the inputs and actuators themselves.

Each controller is a function, enacting some transform on some inputs to some actuators. A dependency is where the process of another controller is required, such that it would have to be accounted for in the higher controller if it did not exist.

There are a wealth of tools for analysing input-output relationships. Detecting the relationships between inputs and outputs has been employed widely in Control Theory [Vignali et al., 2014, Ying, 2006] and Machine Learning [Schroeder and Korel, 2000, Rabitz et al., 1999] and is the basis of sensitivity analysis [Helton and Davis, 2002, Roy et al., 2014]. Any of these can be used and the existing literature supports their effectiveness.

A controller should output to all actuators that affect the input. To have proper control of the input signal, the controller must be able to reach and control all actuators with a consequential effect. Consider some function which tests an input signal and finds every actuator that affects it. Unlike many input-output analysis algorithms, the only concern is whether there is some effect and not what that effect is. Figure 13 shows an example of running an input-output analysis function on an example problem. This problem has 5 actuators, numbered 1 through 5. There are 5 inputs to the system, labelled A through E.

Actuator	Affected Inputs
1	A, B, C, D, E
2	B, C, D, E
3	C, D, E
4	E
5	D

Figure 12: *An example table showing the results of analysing input output relationships on an example problem, searching for whether an input is affected in some way by activating an actuator. The right column indicates the actuators, which are numbered. The inputs are lettered, with all inputs affected by a specific actuator on the right. An entry of a letter in the right column indicates it is affected by the actuator in the left column.*

Consider the controller for input A being in control of the controller for input E. This would result in the controller for input A outputting to every actuator that input E must go to. This would mean input A would lead to actuators 2, 3 and 4. This should not be the case, as input A is not affected by and thus should not be sending control signals to actuators 2, 3 and 4. Given this, it is not possible for A to be hierarchically dependent on E. The other way round is fine, as A's set of affected actuators is a subset of E's set of affected actuators. With this intuition, the number of possible hierarchies

falls sharply.

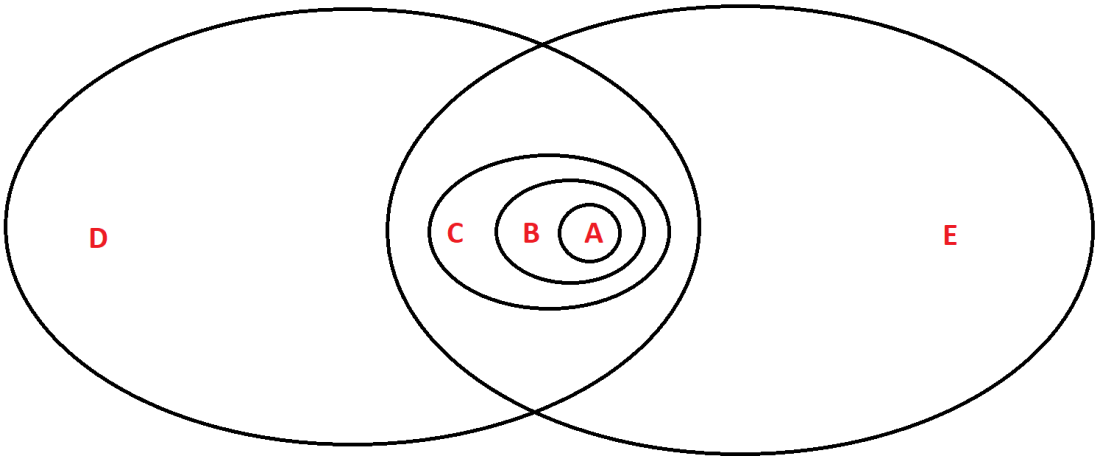

Figure 13: *Venn Diagram showing the possible dependencies that do not breach the closed loop control principles. Each oval represents the scope of an input, labelled inside the oval in red. Any oval which falls completely in another oval is a possible hierarchical relationship, because the inside oval connects only to actuators the outside oval must connect to. Given this, E and D can have hierarchical relationships with any other inputs except each other. C can hierarchically delegate to B or A, and B can hierarchically delegate to A. However, A cannot have a dependency as their actuator set is too small to permit delegating their control output to another controller.*

If one considers all possible hierarchical arrangements of a system, this is a significant decrease in the number of possibilities. Assume one controller per input which can connect to any number of other controllers. In theory, each controller could form connections to the actuators in many different ways. Assume the input-output analysis is done and thus each configured controller has a clear decision on how the hierarchical controller connects to the outputs. If one simply considers the controller, the number of hierarchies that can be made is of the order:

$$2^{n^2-n} \tag{4}$$

where n is the number of inputs. Each controller can output to every other controller. The subtracting of n represents each controller not being able to output to themselves, where n squared is each controller connecting to each other. To find every combination of these options, two is taken to the power of the number of connections. The process which has been undertaken using the closed loop control principles reduces it significantly, from approximately 33,000,000 to approximately 500. This moves the problem from being unsearchable to searchable, making it possible for an agent to find the best hierarchy.

The number can be decreased further through progressively building the hierarchy. Once one has this understanding of possible hierarchical arrangements, other intuitions become clear. In the example shown in this Subsection, A's controller must output to actuator 1. There are no other configurational possibilities. B's controller has two options,

which are to hierarchically delegate to A as a means of reaching 1, or bypass it and not hierarchically use A's controller. The rest of the hierarchy is not of concern to this decision, as the effective control of B does not require or depend on the controllers of C, D and E. Therefore, this can be analysed first. A process of this kind would take the 500 possibilities and turn it into a series of choosing between a small amount of options, allowing elegant and progressive traversal of these options.

These investigations point to an agent being able to develop their own hierarchy as plausible and computationally accessible. The next steps are to fully develop this theoretical analysis into an algorithm, methodology or function which a human or an AI could run to derive a hierarchy for a specific problem.

4.3 Detecting Dependencies through Experience in the Environment

Detecting dependencies from the environment is possible using input-output analysis. By understanding how each input signal is controlled, deductions can be made on the possible hierarchical combinations.

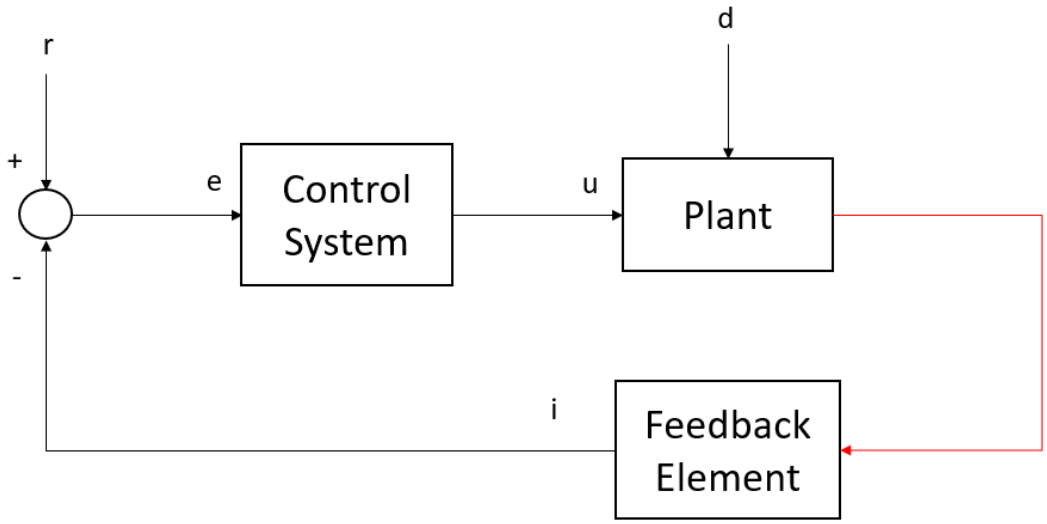

Figure 14: *A typical closed loop control system, with the feedback line indicated in red. The problem of detecting a relationship between a particular input and actuator is the same as detecting the presence of the red line.*

Figure 15 is a standard classic Control Theory negative feedback loop, with the red arrow indicating the part of the model which represents the outside world. To detect whether an input is affected by a particular actuator is detecting whether the red line exists. If no control output to the actuator can change the incoming input, then the red line does not exist and that input is not affected by that output. If any effect is present, then the red line exists. Given this, the input-output analysis needs to search for any connection between them, but does not need to identify any semantics of the connection between them.

A system which provides signals to the actuator and listens for corresponding input that detects changes in the world is suitable for this task. A series of outputs to change

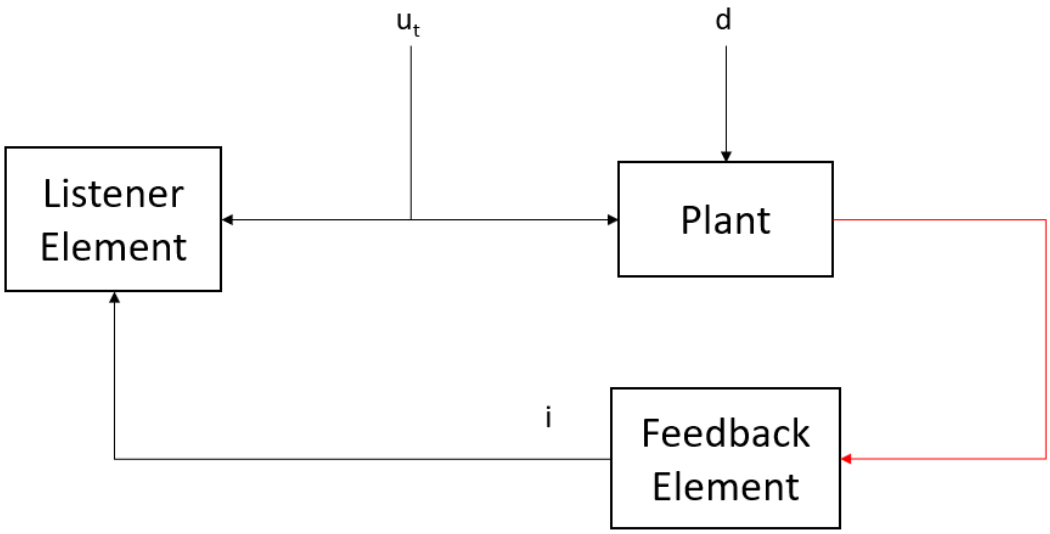

Figure 15: *A system to examine the relationship between an input and an actuator. A listener module is installed and the control system is removed. The listener module aims to examine the input to the controller given the output and find a relationship. The actuator is sent varying signals to provoke happenings in the external world.*

the real world, sent at time t (denoted u^t) are sent to the plant and the listener element. The listener element takes the outputs and inputs and attempts to identify whether the output sent has an effect on the incoming input.

With this investigation concluded, it is clear that an automated process to develop hierarchies can be produced. This shall be explained in three steps: Understanding the input-output relationships, building an order to develop the hierarchy in, and optimising the hierarchy progressively.

4.4 The Dependency-Oriented Structuring Architect

The Dependency-Oriented Structuring Architect, or DOSA, is a methodology to produce control hierarchies. The architecture analyses the relationship between inputs and actuators to devise what order specific inputs should be controlled in. DOSA progressively builds the hierarchy by adding nodes in a specific order, investigating where they best fit in the hierarchy. Finally, once the hierarchy is complete, the parameters can be optimised progressively.

For the subsystems, they are assumed to all have the same internal structure of control elements. Many different controllers could be employed, even with different controllers for different control subsystems. A standardised style of closed-loop control subsystem is used throughout this Thesis, but any controller can be used as a subsystem in DOSA. The subsystem takes the difference between the reference and input, which gives the error. The error is processed by the control function within the control subsystem. The control output it sent to any number of output locations, which can be either multiplied by the same gain or different gains.

4.4.1 Step 1: Extract real world experience

This Section covers an example of detecting input-output relationships from real world experience. A significant change on the input signal after applying large output signals to a particular plant indicates the former is affected by the latter. The algorithm is algorithm 1, with detectActivity(n) and affected(a,b) being functions that are modular. detectActivity(n) reads and processes input signal n. This can be simply the raw values over time, but some more complex analytics are feasible. Affected(a,b) takes the signals a and b, which are one signal where the actuator was activated and one where it was not. The function aims to classify whether signals a and b are sufficiently similar to conclude that activating the actuator made no difference. Two measurements shall be used to detect the activity; they are the average value over a time period and the average change of the value over a time period. An example of detectActivity(n) is given below the description of the algorithm. As for affected(a,b), a simple difference of over 25 percent are considered.

Algorithm 1 Identifying dependencies with DOSA

Require: Set of plants, P
Require: set of inputs, N
procedure AFFECTEDINPUTS(P, N) ▷ The set of affected inputs for outputs between all elements of P and N
    $D \leftarrow EmptySet$
    for *Every combination of* $\boldsymbol{p}$ *and* $\boldsymbol{n}$ *in* $\boldsymbol{P}$ *and* $\boldsymbol{N}$ **do**
        while *Loop count below threshold* **do**
            Send R as output signal to p
            $a \leftarrow detectActivity(n)$
            Stop sending R as output signal to p
            $b \leftarrow detectActivity(n)$
            if $affected(a, b)$ **then**
                Add (p+n) to D

An example of detectActivity(n) can be as simple as comparing the average value and average rate of the input when the plant is activated and when it is passive. The average value and average rate shall be defined for a discretely sampled input i at time t through n timesteps. The timesteps between times t and t+1 depends on how often p can be sampled based on the robot or software being used.

$$AvgValue(i, t, n) = \frac{(i_t + i_{t+1}...i_{t+n})}{n} \tag{5}$$

$$AvgRate(i, t, n) = \frac{(|i_t - i_{t-1}| + |i_{t+1} - i_t|...|i_{t+n} - i_{t+n-1}|)}{n} \tag{6}$$

Some threshold is necessary to define whether there is a significant enough of a difference to define the input being affected by the output. μ shall be defined as the threshold parameter. This threshold parameter represents a percentage of difference from the comparative value required to consider it affected.

$$depValue(i_t, T) = \begin{cases} 0 & \text{if } |AvgValue(i,t,n) - AvgValue(i,t+T,n)| < \mu * AvgValue(i,t,n) \\ 1 & \text{Otherwise} \end{cases} \tag{7}$$

$$depRate(i_t, T) = \begin{cases} 0 & \text{if } |AvgRate(i,t,n) - AvgRate(i,t+T,n)| < \mu * AvgRate(i,t,n) \\ 1 & \text{Otherwise} \end{cases} \tag{8}$$

Where T is the time between the start of the active interval (where the plant is being sent output signals) and the inactive period (when the plant receives no output signals).

$$affected_{t,T}(i,p) = \begin{cases} 1 & \text{if } p \cap depValue(i_t, T) = 1 \\ 1 & \text{if } p \cap depRate(i_t, T) = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{9}$$

t indicates the current time at the invocation of the function and T is the number of time steps before the plant changes between inactive and active. i is the input being tested and p is a boolean which indicates whether the plant is active at time t. The affected function can be repeated any number of times to test again or confirm whether one affects the other.

Finally, information on the value for each input is recorded over the phase. The default value and rate (that which occurs when no actuators are engaged) are also recorded. Additionally, the value and rate acquired through use of the plants is recorded giving two bands of values. These are used later to approximate gain values, giving a suitable starting point.

4.4.2 Step 2: Derive the hierarchical ordering of inputs

With a list of affected input and affecting output pairs from step one, an understanding of what hierarchies are possible can be derived. Importantly, the reduction in the number of possible hierarchies means the space can be navigated. Step two takes the dependency pairs and works out an order in which control subsystems can be added to the hierarchy to best navigate this computational space. The principle is that a control subsystem should not be added until all possible control subsystems that it could depend on are added.

When considering the progressive development of a hierarchy, one must aim to control what should be at the bottom of the hierarchy. What should be at the bottom of the hierarchy is the subsystems that have no dependents and interact with the actuators directly. To have no dependents, no other input must be a subset of the related actuators that affect it. This makes the problem logically simple to solve and presents an order in which one can progressively add nodes to the hierarchy to guarantee finding all dependencies.

To produce a hierarchy, the system must use D, N and P. D is the pairing of dependencies between inputs, N is the list of all inputs to the system and D is a list of all plants in the system. The first step is to convert this into a list of controllers in order of when they should be progressively added to the hierarchy. The second step given that list is to work out all options when each controller is added.

Hierarchy(D,N,P) generates C, which is a list of control subsystem candidates and possible connections they may have hierarchically. c is defined as a single control system candidate, which is formatted as a list of three lists. Those respectively are inputs c_n, outputs c_u and set of possible hierarchies c_p. Each iteration, n is selected to be the input with least pairings in D. δ_n is then extracted from D, which is all plants in every pairing with n. It searches for τ, which is defined as the control subsystem which has exactly the same affected outputs as δ_n. If τ exists, then this input must be considered at the same time as the input with identical dependencies. That part of the hierarchy needs to be deduced in stage three, when more is known about their relationship. If τ does not exist, this input requires their own control subsystem that may have hierarchical relationships. If τ is not found, then τ must be generated to be a new control subsystem for this input signal. A control subsystem is generated such that $\tau = \{[n], \delta_n, .$ Once this is generated, the list of possible hierarchical relationships can be developed. All controller candidates already in C are examined, specifically looking at their output list c_u. A temporary list as a copy of C is developed, denoted C_n. All of those that have an element in s_p that is not an element in τ_p are removed from C_n, since it cannot be a lower level hierarchical subsystem without breaking the closed loop rule. c_p is then populated with every different combination of controllers. This could be none of them, or all of them. So, the methodology must process every combination. k is iterated over from 0 to the length of C_n, selecting every option in C(k,length(C_n)). Finally, each option in c_p is examined for what dependencies it meets. For each hierarchy candidate, c_p of all hierarchical connections for this hierarchy candidate are summed to make a set named β. If β is not the same as τ_p, some outputs are missing from the hierarchy and thus must be direct connections to the plants. β is subtracted from τ_p, to produce a set of all plants that have not been accounted for. Direct connections to each of those plants are placed in this option, such that every connection to make this a valid hierarchy is present.

1. requires D, list of dependency pairings (n,p) of input and plant
2. (initialise C) C $\leftarrow$ []
3. (Identify τ):
    - n is set to the input that is in the least non-zero pairings inside D
    - δ_n is defined as a set of plants in D that contain n in their pairing.
    - remove all pairings in D that contain n.
    - Does a c exist in C such that c_u is equal to δ_n?
        - if not, $\tau \leftarrow \{[n], \delta_n, []\}$, a new control subsystem which has input n and required outputs δ_n. Then, add dependencies to τ:
            * (add dependencies to τ) If τ_u is empty, process dependencies. Every control subsystem is iterated through and if such is a subsystem that τ depends on then it is added as an output. For every s in C in order of descending size of s_u, do the following:
            * (initialise P_n) $P_n \leftarrow []$
            * For each s in C:
                · If set(s_u) - set(tau_u) = $\emptyset$: $P_n = P_n$ + s
            * $\tau_p \leftarrow$ every combination of items in P_n for all k (k is 0 to length of P_n)

        * for each s in τ_p:
            - $\beta \leftarrow \emptyset$
            - for each c in s: $\beta \leftarrow \beta + c_p$
            - s $\leftarrow$ s + $(s_p - \beta)$
    - $q \leftarrow \emptyset$
    - if D is not empty, return to beginning of loop and identify a new τ
    - otherwise, return C

The algorithm returns C, which is a list of subsystems in the order they need to be added to the control system. Each subsystem in C has a set of possible hierarchies that are valid and meet the required outputs of the controller. Now that this information is known, the computational space has been sufficiently reduced to progressively develop the hierarchy.

4.4.3 Step 3: Progressively balance the hierarchy's gains

Step 3 chooses the best hierarchy candidates and balances the parameters of the subsystems progressively. From the second step, a list of control subsystems C is provided. Each subsystem c in C has inputs N, output destinations U and set of possible hierarchical arrangements P.

Step three requires some minimisation function where a variety of functions from Machine Learning or mathematics can be employed. The algorithm is displayed below with minimiseError(c) being the modular function to minimise the unknown parameters of control subsystem c. minimiseError takes a control subsystem c and tests all hierarchical arrangements in c_p. It returns the candidate in c_p that minimises the error function best. For each c_p, it must learn suitable parameter combinations that minimise error.

As options for minimiseError(c), Reinforcement Learning or evolutionary algorithms can be employed. Evolutionary algorithms are commonly applied to parameter optimisation in robots [Rahmani et al., 2016, Guo et al., 2015]. Gradient descent with randomised starting positions is another option, with more computationally intense options available from Machine Learning. As an alternative, constant gain adjustment through analysing the gradient of the error is possible.

A simple example shall be considered. This is a simple problem with three actuators labelled 1, 2 and 3 and three inputs labelled A, B and C.

Through step one, the inputs are analysed for actuators that they're affected by. Input C is affected by all three actuators. Input B is not affected by actuator 3 and input A is only affected by actuator 1. This forms the list of input-plant pairings:

$$[(A,1),(B,1),(B,2),(C,1),(C,2),(C,3)]$$

This is taken by step two, which then converts this into a list of subsystems and possibly hierarchies. A is processed first, since it has one dependent actuator. There is only one option, which is to output to plant 1. So, control subsystem α is made to process A and output to plant 1. B then has two options. It must delegate to plant 2, as no other route exists. To access plant 1, it can either provide output directly or hierarchically delegate to the subsystem taking input from A. Control subsystem β is made to model this. Finally, the subsystem for input C has four options. It can either avoid the hierarchy altogether, hierarchically arrange with A, with B, or with A and B.

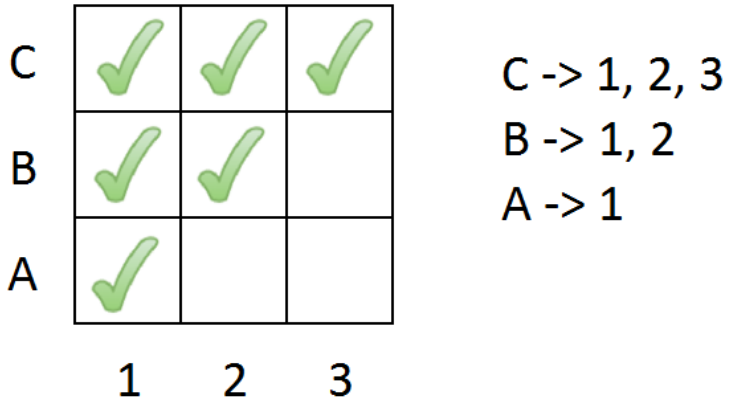

Figure 16: *A grid showing the dependencies. A tick in a box indicates that the input (lettered on the left) is affected by the output (numbered below).*

In all of those options, it must output to 3 directly as there are no other routes. This is represented by control subsystem κ. The three control subsystems are α which controls input A, β which controls input B, and κ which controls input C. The control subsystems list for entering the third stage is:

$$[\alpha : [\{A\}, \{1\}, \{1\}],$$
$$\beta : [\{B\}, \{1, 2\}, \{[1, 2], [\alpha, 2]\}],$$
$$\kappa : [\{C\}, \{1, 2, 3\}, \{[1, 2, 3], [\alpha, 2, 3], [\beta, 3], [\alpha, \beta, 3]\}]]$$

The control structure can now be progressively developed. Stage three begins with the inputs and plants and no control structures. This stage is entered with the control subsystems list as well as information on the ranges of input signals for A, B and C from investigations in step one.

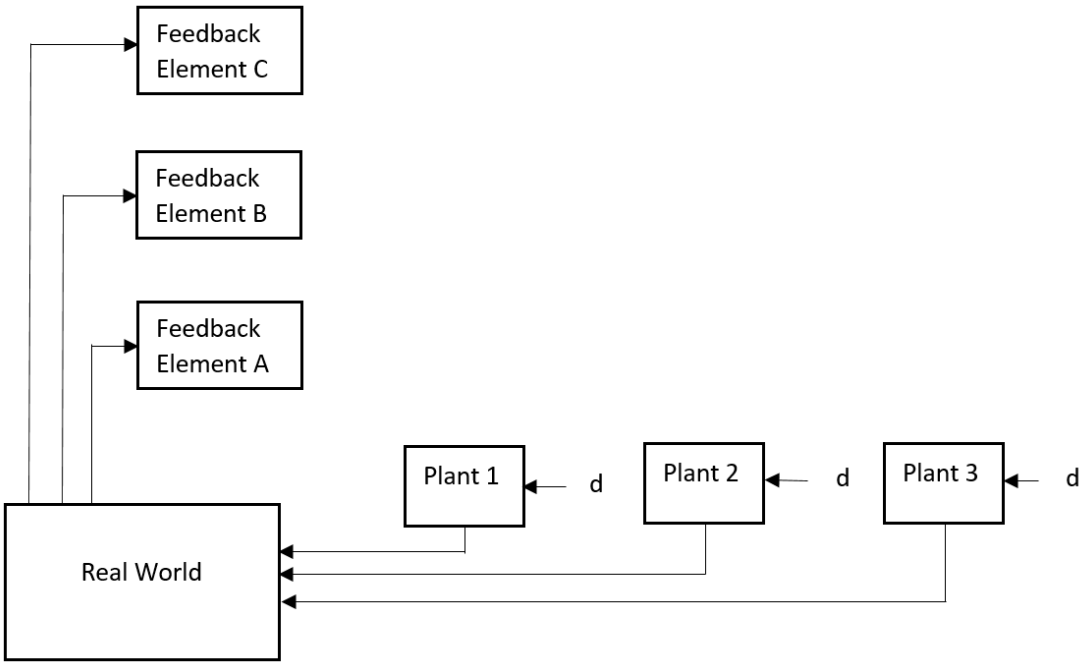

Figure 17: *A diagram of the control hierarchy before any subsystems are added.*

The first to be processed is α, which has an input from feedback element A. Once this subsystem is added with the correct input, the possible outputs are considered. There is only one entry in the possible hierarchies, which is to output to plant 1. This is because it has one dependent output (which is plant 1) and currently there is no way of reaching it other than a direct connection. As such, this connection is added. The hierarchy is then trained on controlling the signal from feedback element A to tune this controller with minimiseError(α). Information from stage one about the range of values for input signal A can be used to set appropriate dummy references for α.

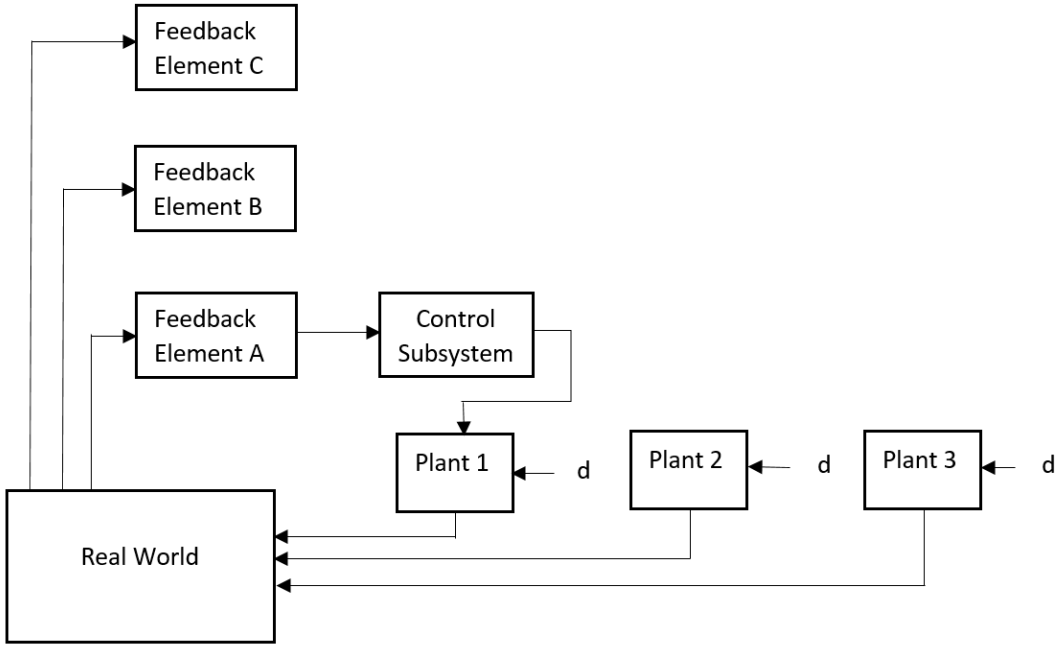

Figure 18: *A diagram of the control hierarchy once a subsystem has been added for input A.*

The second consideration is input B and the respective control subsystem β. Control subsystem β is added with an incoming input signal from feedback element B. There are two possible options for this control subsystem to output to. The control subsystem can output to plants 1 and 2 which avoids the hierarchy, or plant 2 and the first control subsystem α. Because α is already optimised, it is possible for minimiseError(β) to assess whether delegating to α is superior to directly sending signals to plant 2. Both options are explored and the one that has less error. For this example, let us assume it is hierarchically delegating to α and plant 2. The control system now has two controllers controlling two inputs through delegating to two plants. The progressive production of the hierarchy means later subsystems can make informed choices of which subsystems to delegate to since they are already optimised and have their hierarchies set in place. This is important for later additions as more control subsystems means more hierarchical possibilities.

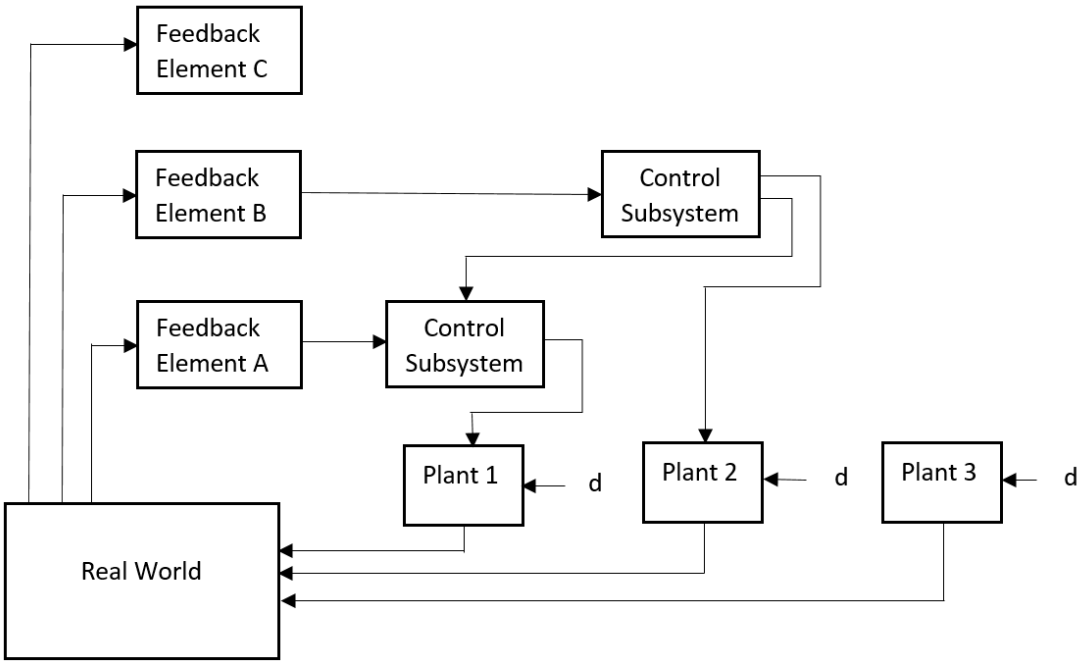

Figure 19: *A diagram of the control hierarchy once a subsystem has been added for input B. Note the new subsystem outputting to both Plant 2 and referencing the other control subsystem as a means of reaching plant 1.*

The final control subsystem to process is κ. κ is added with an incoming input from feedback element C. Because C has a lot of outputs to reach, there are more possible hierarchical combinations. κ can delegate to α, β, α and β simultaneously, or neither. Again, these options can be processed by minimiseError(κ) as both α and β have been optimised.

Now that all dependencies have been implemented, the hierarchy should have a closed loop for every affected output and one should *not* be present where an affected output isn't present. One can see this is the case by examining the diagram and following the hierarchy from input to output and back round. The identified dependencies clearly solve this problem.

The type of control subsystem being used is has exactly two stimuli (one as input, one as a reference). Thus, if a node has more than one input or more than one reference, then further refining and splitting into multiple nodes is necessary. However, this can be done in step 3 when more is known about the lower subsystems in the hierarchy.

4.5 A consideration of a robotic arm

Modelling of a robotic arm is common among research in robotics, but functioning hierarchical models are hard to find. Unexpectedly, a standardised solution is not present.

DOSA was ran over a theoretical robotic arm. For simplicity, the robotic arm has three digits on the hand with only two partitions and also move only in two dimensions. The reason for both of these is to reduce the number of plants and inputs for this example. The inputs are a perception of location at each joint in the X and Z axis. How this is perceptually coded as a signal at this point is not detailed. If inputs are lettered A

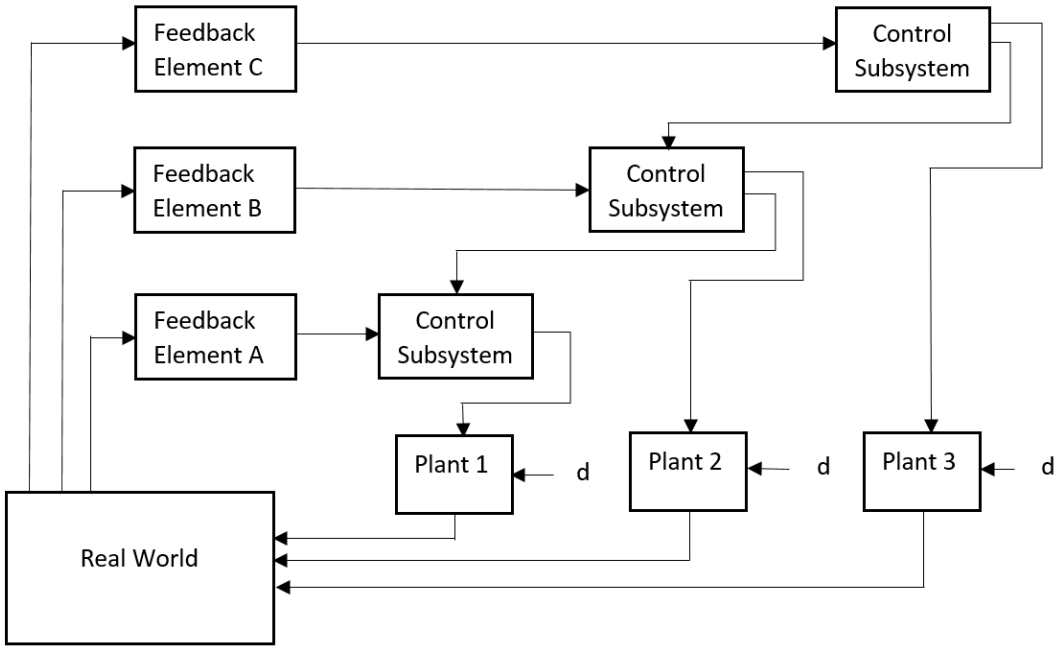

Figure 20: *A diagram of the control hierarchy once a subsystem has been added for input C. For example purposes, it is assumed that delegating to the second control subsystem is the one that minimises error the best.*

through I and plants numbered 1 through 6, below is a diagram of the robotic arm.

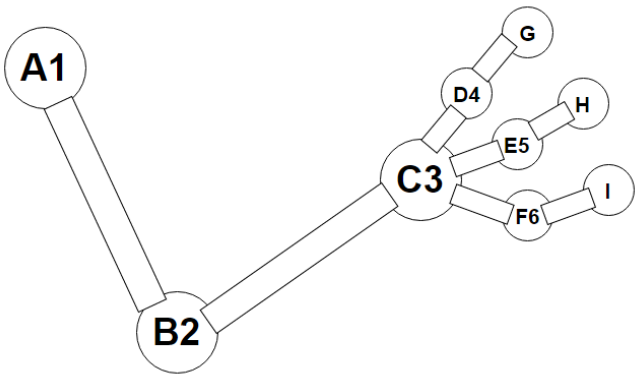

Figure 21: *A representation of the theoretical arm considered in this problem. Letters represent sensors which detect positions at the various joints whereas numbers represent motors which move in X and Z space.*

The exploratory phase would reveal the following affected input and plant pairings.

A						
B	✓					
C	✓	✓				
D	✓	✓	✓			
E	✓	✓	✓			
F	✓	✓	✓			
G	✓	✓	✓	✓		
H	✓	✓	✓		✓	
I	✓	✓	✓			✓
	1	2	3	4	5	6

Figure 22: *A grid showing which inputs (left, lettered) are affected by which outputs being activated (bottom, numbered).*

A control structure is initialised with the plants connected to the real world which then lead to the inputs. As a reminder, the algorithm must make sure there is a closed loop for each affected pair and stack appropriately where one input has a subset of affected plants.

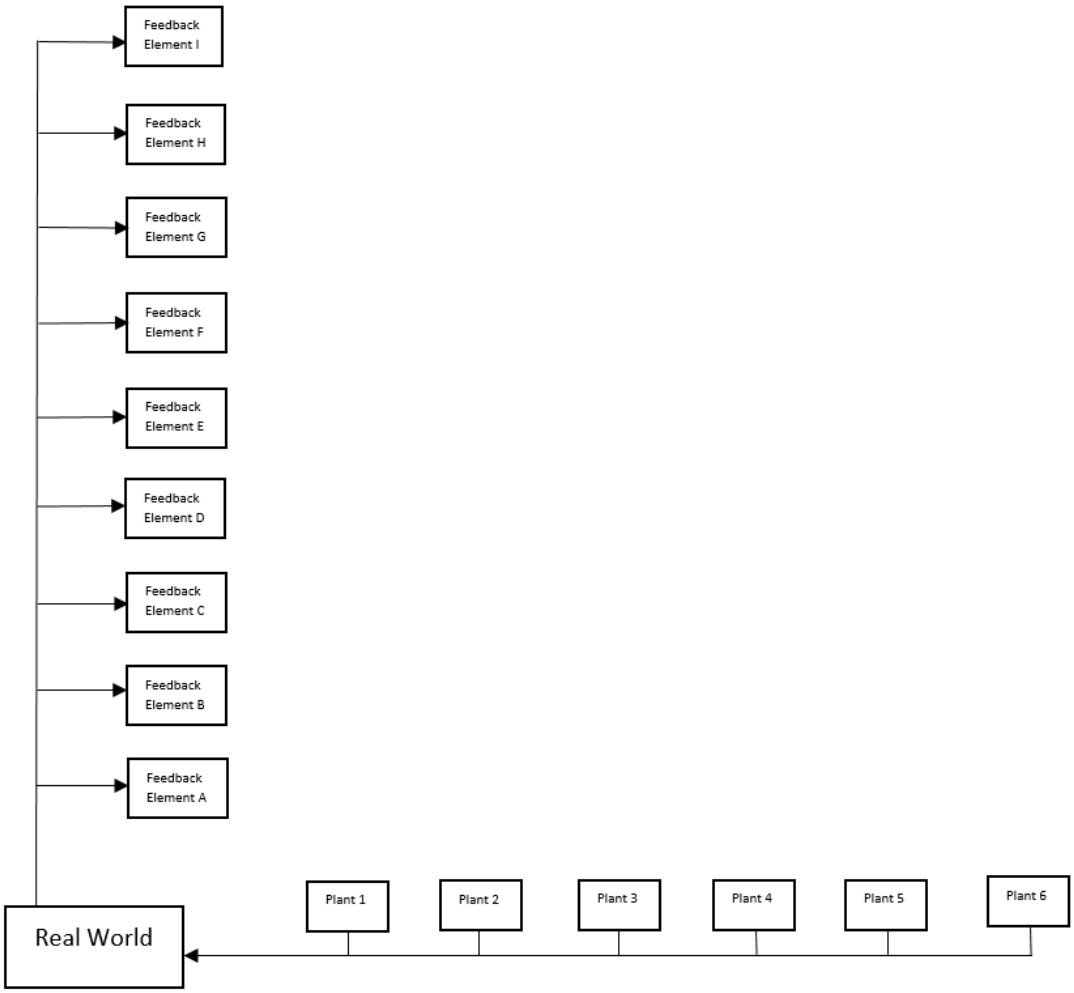

Figure 23: *The control system for the robotic arm. Currently, there are no control subsystems.*

Note that A has no affecting plants. Thus, it will not be in the final control structure. A subsystem for B is made with only one option, to delegate to plant 1. This makes sense, as the only joint to affect the position of the elbow is the shoulder. So, control of the elbow should only apply force to the shoulder joint. A subsystem for C is then added, to control the position of the wrist. There are two options: to control the elbow and shoulder directly, or control the elbow directly and the shoulder via delegating to B's control subsystem. Control of position of the elbow is a requirement for controlling the position of the wrist, so delegating to the control subsystem for B is the ideal option.

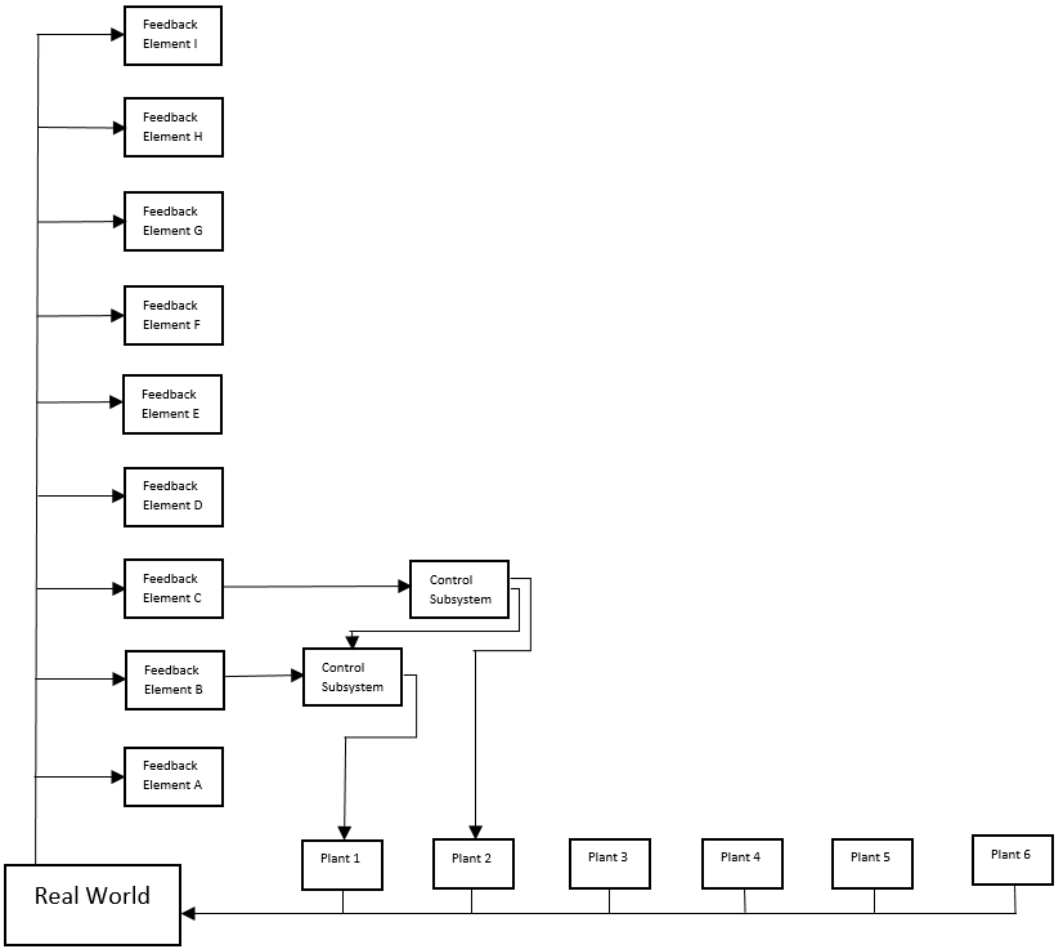

Figure 24: *The control system for the robotic arm. It is updated with inputs A, B and C accounted for with control subsystems.*

D, E and F must be considered together as they have the same affected plants. Given this, they could be hierarchically arranged or level and experimentation can ascertain this. This is analogous to the inverted pendulum problem explored in Sections 3.9.4 and 4.1, where all of the inputs were affected by the single same plant and thus a hierarchy was not directly deducible from that alone.

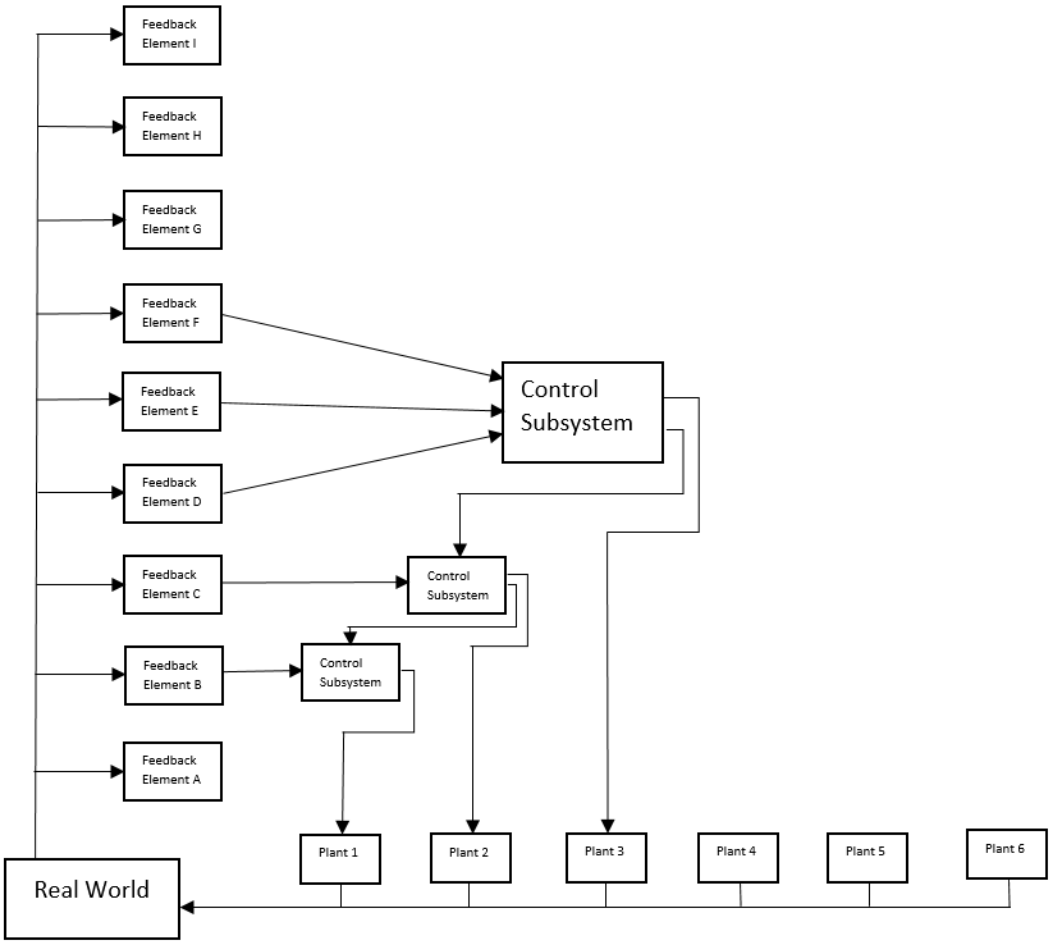

Figure 25: *The control system for the robotic arm. It is updated with inputs D, E and F accounted for with control subsystems as well as those previously accounted for.*

To identify the relationship between D, E and F, each needs an individual controller. To work out which is a transform that requires the others involves attempting to control these inputs. After exploration, D, E and F all achieve successful control by delegating to controller C and also plant 3. Each one then attempts to be hierarchically above the others. With intuitive knowledge of the problem, it is clear that a hierarchy between D, E and F would not be effective. Each of the controllers has intentions for the reference signal to be sent to lower plants. If they are hierarchically arranged, one particular control subsystem has priority. What this results in is uneven control between what should be independent joints each evenly conflicting for control. As such, D, E and F have three separate control subsystems. Each of the outputs of those go to plant 3 and the control subsystem for B.

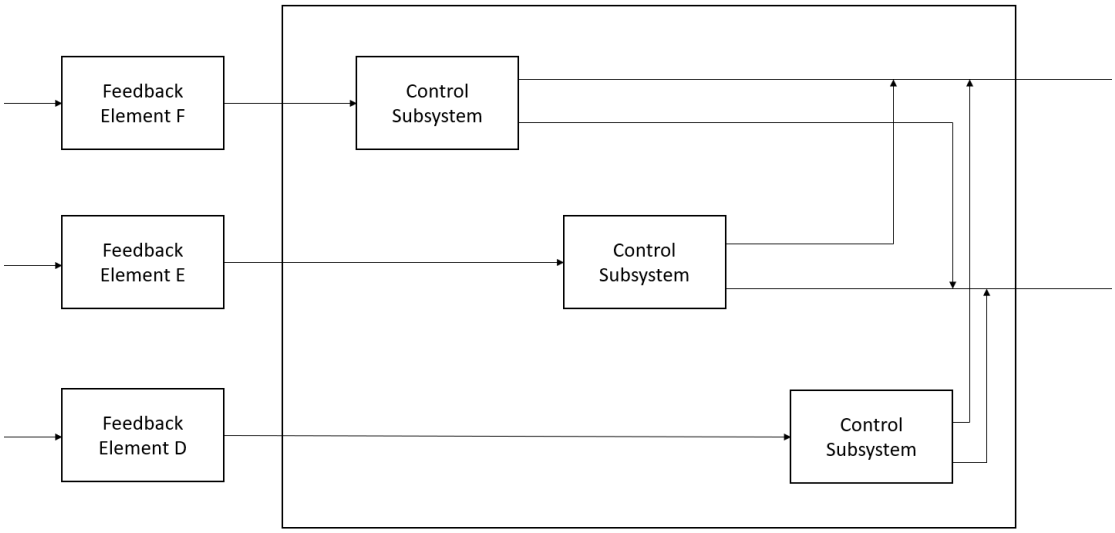

Figure 26: *The control subsystem for controlling inputs D, E and F. Three controllers are added and it is discovered through exploring the environment that they are concurrent, not hierarchical.*

Finally, G, H and I are considered in turn. Each one has an extra dependency which can only be handled by direct signalling to the plant. Through delegation, they can meet the needs of the other dependencies. Through exploration with the previously balanced systems, it is discovered that each subsystem operates best when it delegates to one of the three subsystems for D, E and F. This completes the hierarchy, as all control subsystems are added.

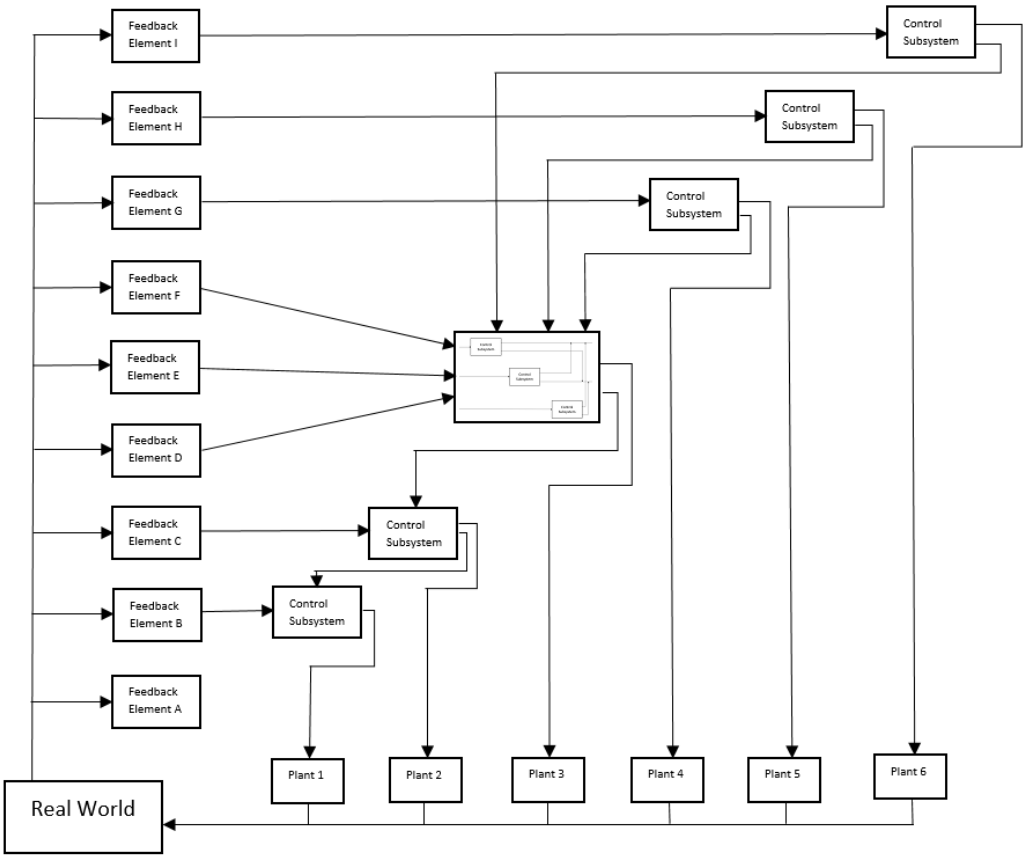

Figure 27: *The control system for the robotic arm. It is updated with inputs G, H and I accounted for with control subsystems. All control subsystems are added as all inputs have reached all their affecting outputs.*

From examining, there is a closed loop for each affecting plant only. Each input signal exerts control only on plants they are affected by and all the plants they are affected by. The hierarchy produced is based off the dependencies and the function that solves the control problem. Incorrect hierarchies can be built from not understanding what is being controlled or misunderstanding what depends on what.

This has demonstrated DOSA on an abstract problem, but analysis is needed on a solution with a comparable hierarchy rather than some abstract situation. DOSA navigates the computationally massive space sufficiently. It is important to analyse what hierarchy DOSA derives on a known hierarchical problem with a working solution.

4.6 Examining a Hierarchy Produced by DOSA

Large hierarchies that robustly solve problems are rare in published material. LCS III is one of the few published hierarchies that is of a large size [Powers, 2008], solves a problem robustly and clearly details the hierarchy used. Furthermore, it can be explored in the programs provided in the book. Living Control Systems III example code 8 provides the working PCT hierarchy of an arm. The three parts to this example cover reorganisation, multi-task management and target tracking. This hierarchy is large, containing 21 controllers. The hierarchy was extracted from the code, with the joints being labelled based on their location. Upper Shoulder and Lower Shoulder control the shoulder yaw and pitch respectively, Upper Elbow and Lower Elbow control the shoulder roll and elbow pitch respectively and Upper Wrist, Middle Wrist and Lower Wrist control the forearm roll, wrist pitch and hand roll respectively.

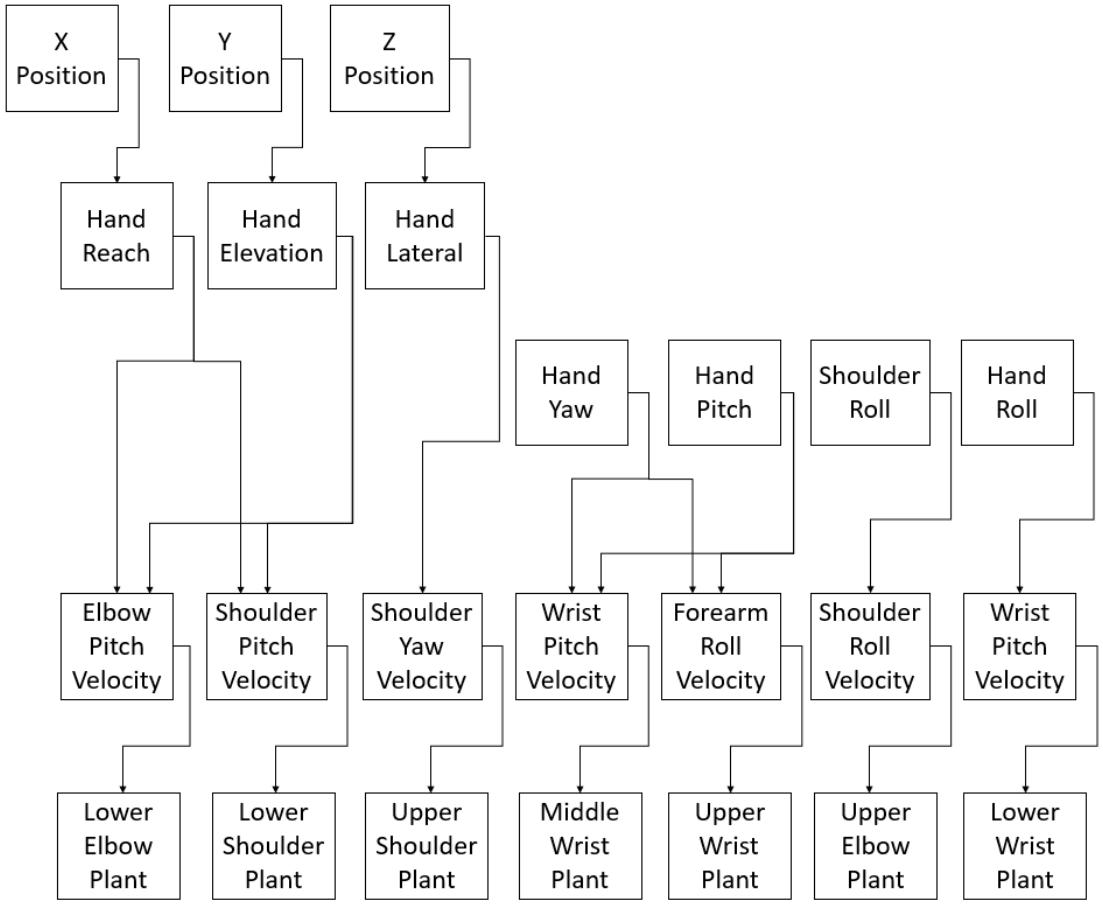

Figure 28: *The control diagram for the Living Control Systems III arm used in example 8 [Powers, 2008]. The actuators are labelled at the bottom and the inputs to the system are indicated by a short line entering the controller on the left. The inputs are the external measures of the text inside the controller, which indicates what perception that controller is controlling.*

The hierarchy, shown in Figure 28, was manually derived and seems to follow the dependencies of the problem. DOSA can take this problem and deduce a hierarchy. The interface allowed the process of deriving dependencies to be undertaken on the arm rather than theorised. The multi-task management allows references to be supplied to particular controllers. By sending signals to particular actuators or controlling particular higher level perceptions, it can be seen what other perceptions or controllers are affected. This gives an indication of the dependencies in this problem.

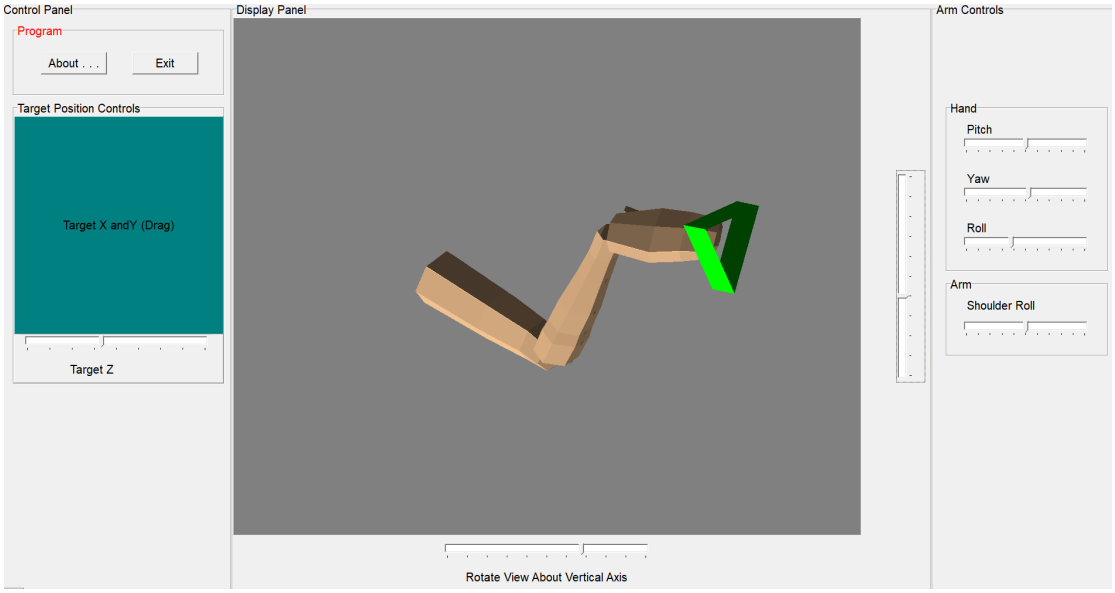

Figure 29: *The interface for Living Control Systems III example 8 part 2, multi-task management [Powers, 2008]. The sliding bars and interfaces all allow reference signals to be set amongst the existing hierarchy, allowing control of the joints. This allows experimentation to work out dependencies rather than deriving them in theory.*

The input-output analysis for this problem are listed in Figure 29. This shows which actuators affect which inputs. For ease, the table does not include the relationships for the individual joint velocities, as each is only affected by the actuator for that joint. These input-output relationships already indicate sensible choices in the Living Control Systems III hierarchy. Every controller in the Living Control Systems III hierarchy only delegates to controllers that control a subset of the actuators required for the higher goal. No controller delegates to another controller that outputs to an actuator the higher controller doesn't require, which would be a concerning sign as a choice of hierarchy.

There are a few solutions to this, which are either outputting directly to the required actuators or using the hierarchy. Figure 32 is a hierarchy that matches the input-output relationships. For step three of DOSA, the process was heuristically performed. Each step of adding a hierarchy, the hierarchical candidate that would perform better in comparison given the control problem was selected.

X position	✓		✓				
Y position		✓	✓		✓	✓	
Z position	✓	✓		✓		✓	
Hand Lateral	✓		✓				
Hand Elevation		✓	✓		✓	✓	
Hand Reach	✓	✓		✓		✓	
Shoulder Roll			✓				
Hand Yaw					✓	✓	
Hand Pitch					✓	✓	
Hand Roll							✓
	S0	S1	E0	E1	W0	W1	W2

Figure 30: *The dependencies produced by experimenting with Living Control Systems 3 example 8. The ticks indicate that the input (on the left) depend on the actuator (below). The joints are labelled S0, S1, E0, E1, W0, W1 and W2. S0 and S1 are the upper and lower shoulder joints respectively, E0 and E1 are the upper and lower elbow joints respectively and W0, W1 and W2 are the upper, middle and lower wrist joints respectively.*

Figures 28 and 31 are similar, supporting the effectiveness of the selected hierarchy by DOSA. It is important to consider why there are connections not included in the LCS III hierarchy but that are present in the DOSA hierarchy. The first note is that the hierarchy was built to manage multiple tasks, so that the hand pitch, hand yaw, shoulder roll and hand roll could be controlled by the interface. As a result of this, it is possible Powers did not want to include the hierarchical links to those four so that there was no conflict between external references from the interface user and higher level references. Furthermore, the connections that were detected by DOSA solely are not major contributors to the control of the particular perception. The added lines represent small effects that one could do without, but niche situations would require these connections. The controller is still able to functional without these, as the more impactful connections can compensate for these.

For context, it is important to consider how many hierarchies there were for DOSA to choose from. If there are n system inputs and thus n controllers, each controller is assumed to be able to connect to any number of other controllers besides themselves. This ignores every possible controller and how they connect to the outputs, which would increase the amount even more. The amount of hierarchies for n inputs would this be of the order:

$$2^{n^2-n} \tag{10}$$

For a 17 input hierarchy, this is 2^{272}, which is extraordinarily large. This is not considering the connections to the actuators, showing the combinatoric explosion as more inputs are added. Given the possibilities, DOSA found a hierarchy that is exceptionally similar.

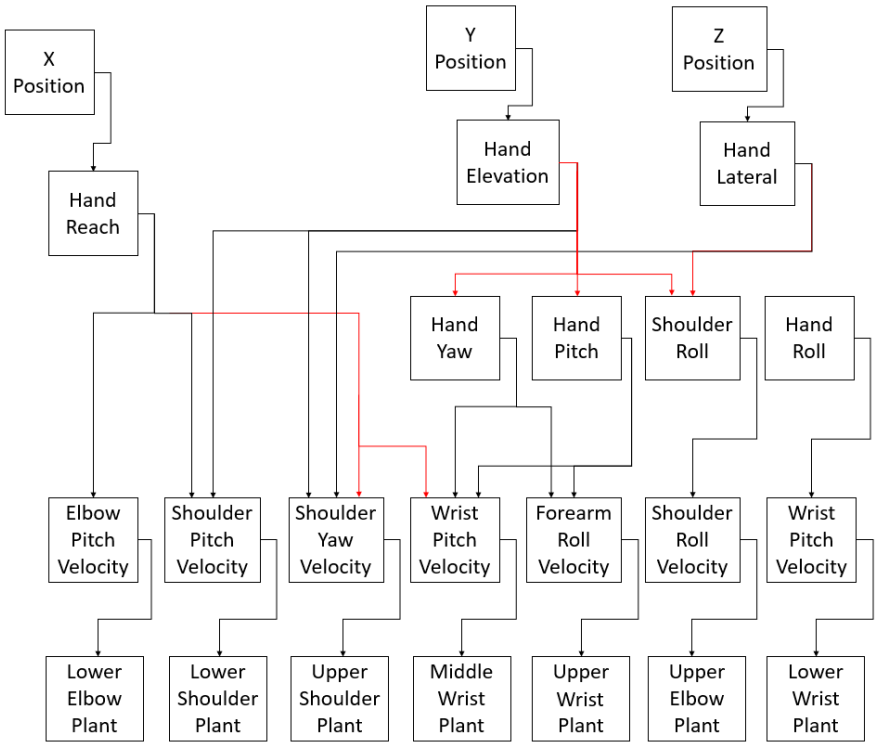

Figure 31: *An example hierarchy built from the dependencies shown in the previous Figure. Each control subsystem is a box with the input signal it receives labelled inside the box. The red lines are those that are different from the structure in Figure 28, which is the original hierarchy that was manually derived.*

4.7 Conclusion and Experimental Direction

A consistent definition for a hierarchy was theorised from analysing literature on living control systems. It was hypothesised a methodology could be made from this definition to allow control structures to be automatically generated. This chapter explored the automatic generation of control structures and produced DOSA, a methodology which makes control hierarchies. DOSA was processed on an theoretical problem and a problem for Living Control Systems III which has a competent hierarchy. The DOSA solution is almost identical to the LCS III solution, even though there are an uncountable number of hierarchies to choose from. Step three of DOSA requires further investigation to show that hierarchies can be progressively optimised as well as derived. The next chapter covers the experimental work undertaken to prove such.

5 The Value of Hierarchical Structures

5.1 Examining the Computational Complexity of Hierarchies

5.1.1 Abstract

Control hierarchies are used to achieve effective results, but whether they are more computationally efficient than non-hierarchical solutions is unknown. A controller must respond in a suitable time frame to maintain robust performance, meaning that lowering the computational complexity of the controller is valuable. No studies have shown whether hierarchical controllers are more computationally efficient than non-hierarchical controllers. This investigation shows that there is no significant computational difference between a hierarchical and non-hierarchical controller. A HPCT controller is compared to a PID controller on the inverted pendulum problem, examining the computational load over the task. These results show that the theoretical reduced computation of a hierarchy is insignificant, due to the computational load of acquiring and processing system inputs. This provides insight on the theoretical truth of the reduced computational complexity, showing that whether a hierarchy is more computationally efficient or not depends on the context of the system and inputs.

5.1.2 Introduction

Robust control requires timely response to control error. Furthermore, controllers often require responses to be enacted at regular intervals. If a controller is responding faster or slower than expected, the tuning of the controller can result in the controller acting disproportionately. Therefore, it is key to have reliable response times and thus manage the computational complexity of a controller carefully. This particularly affects time-sensitive control problems where it is unstable and failing to respond quickly leads to states that cannot be recovered from.

A control hierarchy allows the control process to be split into multiple simpler transforms. Hierarchies already allow effective processing of large-scale control problems. Hierarchies have been used to handle robotic limbs [Powers, 2008], robots on two wheels [Kennaway, 2004, Johnson et al., 2020], path planning problems [Digney, 1996], motor learning in limbs [Lee-Hand and Knott, 2015], cognitive architectures for robots [Franklin et al., 2012], and stabilisations of signals in mechanical parts [Sahrin et al., 2018]. Hierarchies have proved an attractive option for solving control problems in robotics [Powers, 2008, Brooks, 1986], as well as layered control [Bianchi et al., 2011]. The benefit of an architecture that is divided into parts is that each part can be considered individually for optimisation and building. This makes adapting the control behaviour of the overall agent easier than managing one complex function.

However, no studies have been found to prove or refute the claim of reduced computational complexity in hierarchical control. Computational complexity is an important issue in the domain of robotics, as robots that must act in challenging environments may not be afforded large processing power. Furthermore, controllers are built to respond at a regular time interval so control can be consistent. If a controller regularly goes over this time interval between control loops, the controller behaves disproportionately and the tuning parameters do not scale to the new time interval between loops.

A problem was selected that requires timely response to error, the Inverted Pendulum

problem. A suitable PID controller and hierarchical controller inspired by PCT were devised and made as computationally simple as possible. The controllers then ran over a minute and their computational load was compared. This was done 40 times for both approaches. The results conclude that there is no significant difference in the computational load from both approaches, showing that either hierarchies do not have an intrinsic computational benefit or it is sufficiently small as to be insignificant compared to the computation of the overall system.

5.1.3 Experimental Setup

The inverted pendulum problem is attempted using a Levo EV3 mindstorm robot, configured to have two motors with wheels and a gyrosensor. The EV3 mindstorm brick is oriented vertically, such that the robot is top heavy. The wheels are extended below the brick, meaning the brick acts as the 'bob' and the wheels are the 'cart'.

Each trial lasted 1 minute and 5 seconds with the robot being positioned at 0 degrees then being left to balance. Before the robot is halted, the command -uptime is run on the linux terminal accessed through ssh to acquire the computational load. The first value returned from this command gives the average computational load over the last minute. This gives us the result for the trial. 40 trials were run for a HPCT inspired controller as well as a PID controller.

Hierarchical PCT-inspired controller Perceptual Control Theory has been used hierarchically to solve the inverted pendulum problem [Kennaway, 2004]. This experiment used the same hierarchy in that paper, but with each controller reduced to a proportional controller. Other parameters and parts of the equation are not required with the fully built and optimised hierarchy.

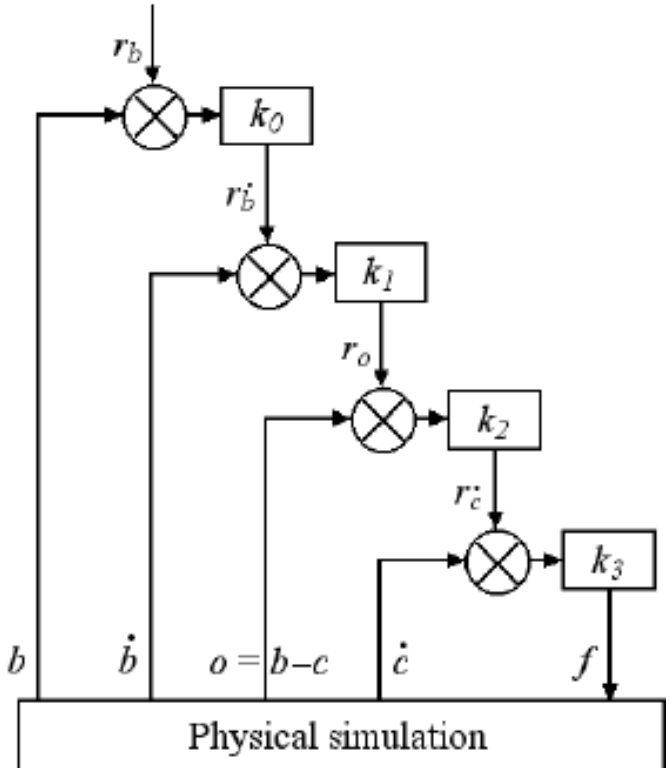

Figure 32: *The control diagram for the inverted pendulum problem. Reprinted from [Kennaway, 2004] with permission.*

The hierarchy is this reduced to the following equation:

$$u = k_3(k_2(k_1(k_0(r - b) - \dot{b}) - o) - \dot{c})$$

This function has 8 operations. First, r-b, then that value times k_1. That value then has $\dot{b}$ subtracted from it, then is multipled by k_2. This process repeats by then subtracting o, multiplying by k_3, subtracting $\dot{c}$ and finally multiplying by k_4.

PID controller The Proportional Integral Derivative controller is a staple solution for control problems. The proportional aspect of the function handles the current error, which uses the benefits of negative feedback control to minimise error. Where this is not enough and more than a proportional force is required, the integral part of the function accounts for a lack of required control output through examining past error. Finally, to reduce turbulence, the derivative part of the function accounts for future error by examining the gradient and resisting steep movements. As a reminder from Section 1, if u(t) is the output for the controller at time t, then the equation is as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Where e(t) is the error at time t and τ is the time constant, intuitively representing the step response between iterations. For programming this solution, simplifications can be made. First of all, the calculation of the integral doesn't need to be performed every phase. Summing the error each iteration gives a rolling integral, but the error must first

be divided by τ. Furthermore, the derivative can be worked out by subtracting the error in the last iteration from the error in this iteration. This then needs to be multiplied by τ. Given this, the equation implemented is:

$$esum = esum + (e(t) * \tau)$$

$$u(t) = K_p * e(t) + (K_i * esum) + K_d \frac{e(t) - e(t-1)}{\tau}$$

Including the calculation of esum which is required, this is 9 operations. The computational cost of saving variables depends on the system, but the PID controller also has more variables that require saving to memory. In theory, the PID controller is more computationally demanding than the HPCT-inspired controller.

5.1.4 Results

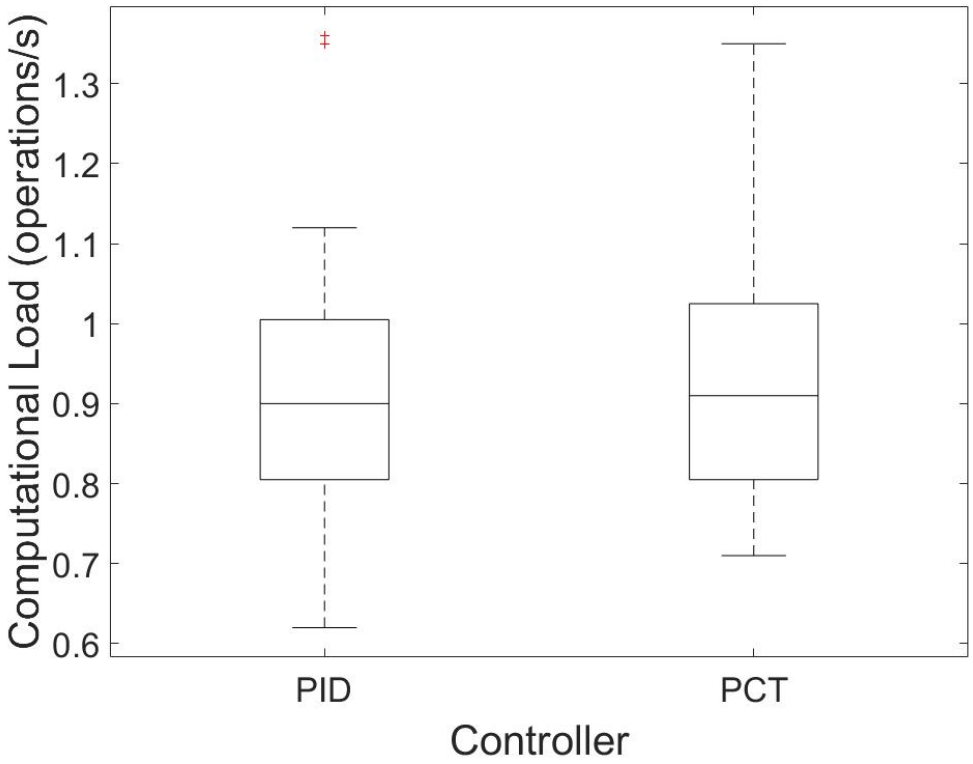

Figure 33: *A pair of box plots indicating the average computational load over 40 trials spanning 1 minute. The left box plot indicates the average computational load for a PID controller and the right box plot shows the same for a HPCT controller.*

The results show the computational load of both approaches. As can be seen, the median performance is near identical. Using one-tailed T-tests, it is found that neither mean is significantly less than the other. While the means remain similar, the upper and lower adjacent values are different due to a number of cases where the controller achieves exceptionally low or high computation. Given the number of operations does not change and PID accesses more variables, this is likely down to the number of inputs the HPCT

system has. The hardware must be accessed to retrieve the input values, which can take variable time to process. If a particular sensor takes longer, this allows computations to build up on the stack. It is clear from this that any benefit from less computations in the controller is erased by the complexity of having extra sensor values.

However, when examining the average error, there is a sincere difference in the performance of PCT and PID. The performances are starkly different, even if this was

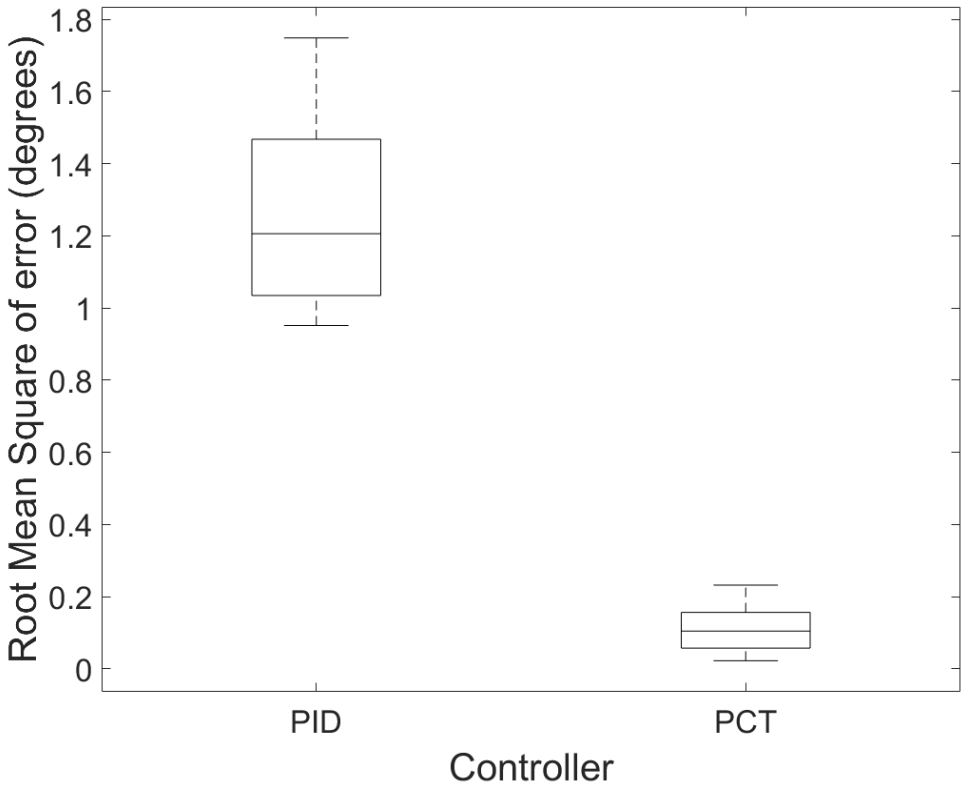

Figure 34: *A pair of box plots indicating the root mean square error over 40 trials spanning 1 minute. The left box plot indicates the average error for a PID controller and the right box plot shows the same for a HPCT controller.*

not visible from examining the performance with the naked eye. Extensive literature exists supporting the effectiveness of PID in a variety of problem domains, so it is important to understand why these results differ significantly. The controllers were both tuned with heuristic methods, which may well explain the performance difference. The PCT controller has four proportional gains, which are much easier to tune heuristically than the PID controller's integral and derivative gains due to their complexity. The PID controller uses only a single input signal, but the integral and derivative account for the signals that the problem does not have access to, such as angular velocity and position of the cart. While a PID controller aims to use the integral to counter error, it is unlikely to be as effective as PCT's approach of controlling the perceived input signal directly. From a user experience, the PCT controller was much easier to produce suitable parameters for. Each layer had a clear objective and a single parameter, which meant selecting and honing in on suitable parameters was easier. The breakdown of the hierarchy also allowed for optimising parameters progressively, as DOSA does in step 3. In

summary, the PID controller has less parameters and less input signals, but the resulting parameter space is much more complicated to navigate. This bears notable similarity to a study comparing PCT control with a Linear-Quadratic Regulator [Johnson et al., 2020]. Both of the controllers were manually tuned using popular optimisation techniques and the Linear-Quadratic Regulator controlled well but not as effectively as PCT. This is a common in both approaches and the Linear-Quadratic Regulator had access to all four input signals, the same as PCT. This suggests it is indeed the parameter space which is easier to navigate and not a lack of access to input signals. This requires experimental investigation to conclude whether hierarchies or PCT make the optimisation process easier and more effective.

5.1.5 Conclusions and Forward Actions

It is clear that the benefit of hierarchies being computationally simpler is context-dependent and also dwarfed by the complexities of the rest of the hardware and software processing. Therefore, a control engineer should not choose one by default for reduced computational complexity, but consider the hardware and computational cost of the extra inputs that come with a hierarchy.

It is of note that optimising the HPCT controller was easier. Attaining a working solution with less parameters per controller made progressively training the controller simpler. While only a single controller, the PID controller has more complex processing of the inputs that make heuristic optimisation challenging. While Ziegler-Nicholls gives a suitable range, manual tuning of this afterwards is often required. This is something a control engineer must consider. It does raise the question of whether hierarchical controllers are easier to optimise compared to their non-hierarchical counterparts. This requires formal investigation and experimentation to see if optimising parameters in hierarchies is easier. If proven, this would conclude that hierarchies are a powerful tool for navigating complex problems and parameter spaces when built correctly, extending the capacity of robotic solutions to real world problems.

5.2 Building a Curriculum to Progressively Learn a Control Hierarchy

5.2.1 How to Build a Progressive Curriculum

From investigations thus far, progressively optimising control hierarchies is of interest and has no precedent in published work. It is also evident that the heuristic process of progressive optimisation is common in practice and optimising the hierarchical candidate in the Inverted Pendulum problem was less challenging. Before investigating the benefits of progressive hierarchical optimisation, how a curriculum to progressively develop a hierarchy must be discussed.

Optimising a collection of parameters in a hierarchy is done heuristically in many ways. Two dividing approaches are (a) using the hierarchy to progressively optimise parameters, or (b) taking all the parameters at once and optimising them together. (a) is less understood and present in research, where (b) has limitations due to being more computationally demanding. A criticism of progressive optimisation is that improperly balanced lower levels can cause the higher goals to be impossible. Provided a suitable curriculum is selected and the hierarchy is valid, this should not be an issue. However, neither format is immune to a training set that is not expressive of the problem. If a

dependency exists, the higher levels should require competent lower levels or be required to compensate for that themselves. The challenge of building a hierarchy is understanding how to train lower levels and make them competent.

The simplest answer is that each controller is controlling an input, thus this is the control goal. The higher levels and their goals are irrelevant, therefore a controller is simply learning to control its inputs. To build a curriculum to manage this, one requires context on the input. While there is no context for the reference signal, this signal should be in the same range as the input signal. From analysing the input signal in the first stage of DOSA, a range of values that can be achieved are evident. After all, a curriculum of reaching 100 and -100 would be unhelpful if the input goes from 1 to -1. Once the input range is understood, sample references can be taken across the range as well as different starting points. The starting error should be varied among tasks in the training set. The only limit on how many tasks to have in the training set is the time constraints of the training. Non-linearities could cause this process to choose troublesome starting positions and targets, due to context of the problem unknown to the agent. An example might be if a robot moves too fast from left to right, safety procedures interrupt the speed and limit it. This may not occur when moving from position 0 to 20, but occurs when moving from 0 to 100. These fringe cases are important to account for and could require further study from the optimisation algorithm. Heuristically, this is not an issue as an engineer can use the context of the problem to tailor the curriculum appropriately. If automated, it would be recommended to set limits on extreme cases or ensure the optimisation algorithm pays particular care to troublesome tasks in the training set.

5.2.2 Producing a Curriculum for a Two-Tier Problem

A problem was proposed to test producing a hierarchy on the control of a robotic arm with limited joint activity. The problem considers the shoulder joint, which rotates the arm around the robot. The control task is to control the relative angle of the elbow to the robot, while the rest of the joints and arm remain in place. By applying force to the shoulder joint's motor, the arm is rotated left or right about the shoulder, changing the elbow's relative position. The complexity of the task is converting the position of the elbow to the velocity of the shoulder by applying an amount of force to the shoulder motor. This was in essence the two tier hierarchy that was tested. The higher controller controls the position of the elbow and delegates to a lower controller. The lower controller regulates the velocity of the elbow, which then calculates how much force is needed and applies this to the shoulder motor.

Two curricula were initially developed. The curricula were required to be short but expressive, given many robots do not have time to train with an exhaustive training set in a real environment. The first curriculum for the lower layer was to train the controller on controlling the elbow's speed. Once this was in place, the second curriculum trained the higher controller to utilise this controller to control the position of the elbow. The velocity controller contained the following tasks:

- beginning at -20 degrees, reference velocity is 0.6
- beginning at 100 degrees, reference velocity is -0.6
- beginning at 0 degrees, reference velocity is 0.1
- beginning at -20 degrees, reference velocity is -0.3

- beginning at 100 degrees, reference velocity is 0.3

The limits of the arm's reach were approximately -40 degrees and 140 degrees, with 0 degrees being directly in front of the robot. Each task lasted *circa* 4 seconds, which was how long it would take the elbow to reach the limits of the arm's range. The higher curriculum was as follows:

- beginning at 0 degrees, reference position is 50 degrees.
- beginning at 0 degrees, reference velocity is 10
- beginning at 0 degrees, reference velocity is 90

These trials required the controller to stabilise at a location to show good control, meaning they took longer. Given they took 8 seconds, 3 tasks was the reasonable limit. The robot is reset between tasks, which again added further to the time requirement of both curricula.

The progressive curriculum was compared against an all-in-one approach. The all-in-one approach simply took both controllers and optimised them based on the higher curriculum. The progressive approach trained the lower controller on the lower curriculum and then the higher controller on the higher curriculum. Both controllers had several candidates which performed well in the training set but when tested on a test set of simple manoeuvres failed to enact any control at all.

It was important to understand why this was. A progressive approach should protect against poor navigation of the parameter space, meaning the only other option was the curriculum was not adequate. Through examining the parameters chosen in the progressive approach, the integral parameter for the lower controller was very high. This is unusual, as the problem should be using a heavy proportional parameter and only using the integral to avoid steady state error. Through investigations, it became clear why this was. The controller was learning the curriculum too rigidly and not the task. Because the robot was reset between tasks, the controller did not have to consider the consequences of winding up the integral parameter. The integral would cause the robot to go very fast towards the objective speed, with the trial ending before the ramifications of this could be accounted for. Evidently, this would transfer very poorly to a real problem.

The first observation is that the breaking between tasks is unrealistic. Given this, it should not be used in a task where the transfer between the tasks is important to capture. Instead, a new continuous curriculum was built. There were no longer any breaks between trials, but rather one continuous trial that tested the transitions between behaviours. The new lower curriculum was as follows:

- Maintain a velocity of -0.3m/s until past -10 degrees.
- Maintain a velocity of 0 for 3 seconds.
- Maintain a velocity of 0.6m/s until past 110 degrees.
- Maintain a velocity of -0.6m/s until past -10 degrees.
- Maintain a velocity of 0.3m/s until past 110 degrees.
- Maintain a velocity of 0 for 4 seconds.

The controller started at 110 degrees, finishing when the time for the trial ran out. When this curriculum was tested, the progressive approach had less candidates that poorly transferred to the test set. However, this needed properly investigating. The two questions to investigate were whether the progressive approach achieved more consistent results on the training set and whether the progressive approach achieved better results on a test set.

5.3 A Structural Approach to Dealing with High Dimensionality Parameter Search Spaces

5.3.1 Abstract

In the field of robotics, searching for effective control parameters is a challenge as controllers become more complex. As the number of parameters increases, the dimensionality of the search problem causes results to become varied because the search cannot effectively traverse the whole search space. In applications such as autonomous robotics, quick training that provides consistent and robust results is key. Hierarchical controllers are often employed to solve multi-input control problems, but multiple controllers increases the number of parameters and thus the dimensionality of the search problem. It is unknown whether hierarchies in controllers allows for effective staged parameter optimisation. Furthermore, it is unknown if a staged optimisation approach would avoid the issues high dimensional spaces cause to searches. Here we compare two hierarchical controllers, where one was trained in a staged manner based on the hierarchy and the other was trained with all parameters being optimised at once. This Section shows that the staged approach is strained less by the dimensionality of the problem. The solutions scoring in the bottom 25% of both approaches were compared, with the staged approach having significantly lower error. This demonstrates that the staged approach is capable of avoiding highly varied results by reducing the computational complexity of the search space. Computational complexity across AI has troubled engineers, resulting in increasingly intense algorithms to handle the high dimensionality. These results will hopefully prompt approaches that use of developmental or staged strategies to tackle high dimensionality spaces.

5.3.2 Introduction

Optimising parameters in functions is key to tailoring their competences to a problem. The more parameters to optimise puts strain on the chosen search algorithm. Eventually, a sufficiently challenging search will result in inconsistent results from the search algorithm, or no success at all [Trunk, 1979]. In the field of robotics, this has limitations and challenges particularly in robots searching for parameters autonomously. Many of the solutions rely heavily on kinematic and mechanical information that is implicitly or explicitly applied to the search algorithm to minimise the complexity of the search. Such information is not always available without an expert in the particular mechanical objective being learned. Furthermore, this information can vary greatly based on subtle properties of the robot or environment. While rewarding, the process of acquiring and implementing a lot of these priors is demanding. Furthermore, there are many domains where this approach isn't feasible due to the required information or expertise being unavailable. Being able to learn a problem without expressed detail of the problem is a valuable skill for autonomous agents to have.

Model-based approaches solve the issue of complex search spaces through exhaustive search. Models can often be evaluated quicker than the robot can run in real time. This allows rudimentary algorithms to brute-force search with many trials in order to find a suitable solution [Wang, 1997]. However, exact models of a particular robot, environment and task are not always available. To build these have thorough knowledge of the robot and environment. Even then, it is easy to forget key details of the problem resulting in the parameters requiring manual tuning afterwards to optimise. Any time saved by making a simpler model places the engineer in a situation later where extra effort is required to manually tune the parameters to better fit the problem.

Machine Learning has a variety of approaches that generalise the kinematic properties in an environment. The extrapolation employed by the statistically based approaches allows inferences to be drawn about the search space, giving success in parameter selection where the parameters have generalisable or predictable behaviour [Matsubara et al., 2011, Lizotte et al., 2007, Martinez-Cantin et al., 2007]. However, approaches can require many trials in order to be successful. Modifications have been developed to improve the search to allow better generalisation in a limited number of trials. Approaches that succeed with consistent results in a handful of trials exist, but often require heavily informative priors or sensitive selection of key meta-parameters to guide the search algorithm [Chatzilygeroudis et al., 2018]. This can be in the explicit model of the problem, or implicitly via a policy which guides the search to suit a particular demographic of problem. Again, these require an expert on the agent's environment who must select or build an informed policy or model. A 'general policy' with which to solve robotic kinematic problems is not available due to the diversity among robots and environments.

Hierarchical Control as a field has considered developmental approaches to optimisation [Digney, 1998, Brooks, 1986, Morimoto and Doya, 2001]. Fields such as Perceptual Control Theory have noted that optimisation of higher levels of a hierarchy requires the lower levels to function [Powers, 1974]. What remains untested is whether the hierarchy is an indicator of which parameters can be optimised independent of the others. Can each level of the hierarchy, starting at the bottom, be optimised independent of what comes above it in the hierarchy? Whether this has been done has not been tested. Furthermore, if this is possible, it is not clear if this approach avoids the downsides that increased dimensionality causes.

This requirement of expert knowledge to minimise complexity presents limitations in autonomous robotics. Furthermore, autonomous robots have a restricted number of trials with which to find a new parameter set. New methodological approaches that aid reducing the complexity of searches would benefit autonomous robots.

This paper describes an approach to the problem based on hierarchical control and staged optimisation of parameters. An experiment was conducted in order to show whether the staged approach suffers less from inconsistent results which is a common effect of dimensionality issues.

5.3.3 Experimental Setup

Baxter Robot The experiment was conducted with the Baxter Robot, a six foot 14-DOF industrial robot. The task concerned the left arm, specifically the joint shown on the left of Fig. 35, named s0. This joint rotates the arm along the X-Z plane. The rest of the arm was held in the position shown in the picture on the left in Fig. 35, so the controller could consistently achieve control.

The task was to control the angular position of the elbow (e1) with respect to the

Figure 35: *A pair of images of the Baxter Robot. The left image shows the whole robot, the controlled joint (s0) and the location that was being controlled (e1) through moving s0. The right image shows the effect of applying force to the s0 joint, either positive or negative.*

shoulder joint (s0) in the X-Z plane. Applying force in either direction of the s0 joint moves the arm around Baxter, changing the angle between s0 and e1 as indicated in the right panel of Fig. 35.

Cascading PID Control A Proportional-Integral-Derivative Controller (also referred to as a PID Controller) is a negative-feedback controller widely used within control systems engineering due to the simplicity and effectiveness of control provided [DiStefano et al., 1967].

A negative feedback controller controls a particular external variable by continuously minimising error, where error is defined as the difference between the actual value and the desired value for the controlled variable[Wiener, 1948]. If e is the error, then the control process can be defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{11}$$

Where $u(t)$ is the control output at time t, $e(t)$ is the error at time t and k_p, k_i and k_d are parameters. The original inspiration was from manual control of steering ships, where it was realised that a sailor would not just aim to minimise error proportionally but also aim to account for lingering error and avoid large rates of change [Minorsky., 1922]. The elegant and simple design affords utility while being Bounded-Input Bounded-Output Stable, making the general responses predictable.

Cascading PID Control (also known as Cascade Control) refers to two (or more) PID controllers where the reference signal for one PID controller is the control output (u) from the higher controller. Cascade control is used for many control applications in recent literature both as is [Sahrin et al., 2018] and with modifications [Reyes-Ortiz et al., 2020, Deng et al., 2019].

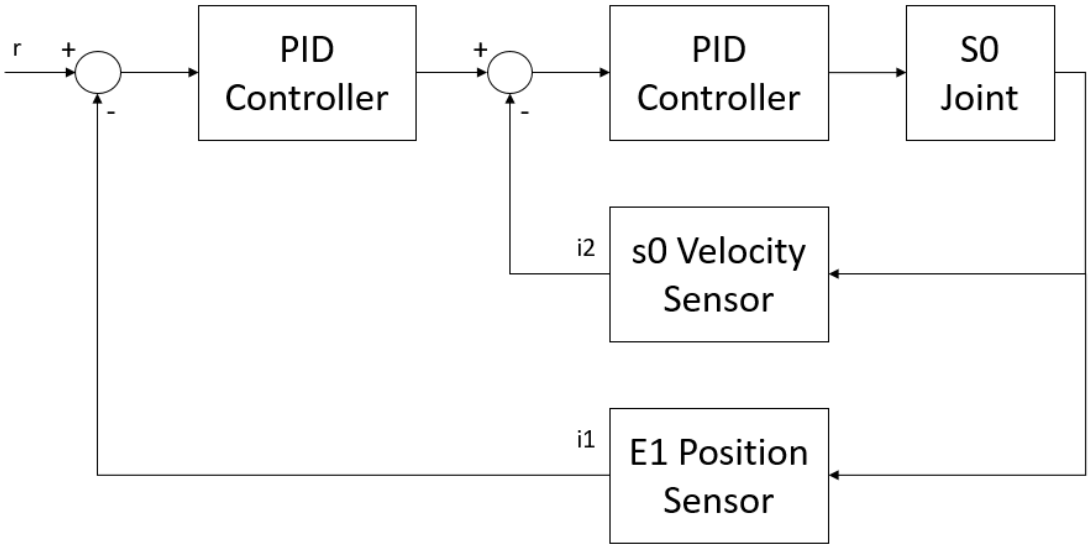

Figure 36: *A diagram showing the cascading PID controller used in this experiment.*

Control System for this Experiment For this experiment, a cascading PID controller was employed to control Baxter's inner shoulder joint (known as s0) to position the elbow at a particular angular position. The higher order controller controlled the angular position of the elbow, sending signals to the lower controller which controlled the velocity of the s0 joint. The lower controller sent a control signal applying torque to the joint. The controller is shown in Fig. 36.

Bat Algorithm Evolutionary Algorithms, inspired by the Genetic Algorithm, benefit from good convergence in a small amount of trials. Evolutionary Algorithms are inspired by patterns noticed in nature, where Bat Algorithm is inspired by the echo-location used by bats to search an area for possible prey [Yang, 2010]. These properties have made the Bat Algorithm useful in control of robots [Rahmani et al., 2016] and more generalised AI tasks such as path planning [Guo et al., 2015].

The variant of the algorithm used in this experiment extends Yang's work. A velocity based approach to updating the candidates [Fister et al., 2013, Fister et al., 2014] and a levy-flights based random walk are utilised. The algorithm optimised candidates to minimise error on the staged and all-in-one curricula, with 30 iterations in total (which were divided equally between the two training stages in the staged approach). See Fig. 37.

Designing Curricula for Developmental Learning Two curricula were developed for learning the problem. One expressed the higher level problem of controlling the angular position, which both approaches used. The staged curriculum also trained the lower controller on how to control the velocity of the s0 joint. For each curriculum, the average error over each task is the score. A curriculum could be built based on a particular task where the candidate simply passes or fails. This is realistic to the environment, as often a difference between average error is not important as long as the candidate passes the task. However, pass or fail tasks are usually domain specific. Aver-

Algorithm 1 Bat Algorithm

Require: Loudness Parameter (1)
Require: Pulserate Parameter (0.75)
Require: Number of bats, $\mathbf{N_b}$ (20)
Require: Number of trials, $\mathbf{N_t}$

$X \leftarrow$ *Establish Bat Populations randomly* $\boldsymbol{X_i}$ *for 1...*$\boldsymbol{N_b}$
$F \leftarrow$ *Establish* $\boldsymbol{F_i}$ *as runTrial(*$\boldsymbol{X_i}$*) for 1...*$\boldsymbol{N_b}$
$Q \leftarrow$ *Establish* $\boldsymbol{Q_i}$ *between 0.0 and 0.2 for 1...*$\boldsymbol{N_b}$
while *Loop count below* N_t **do**
    for $\boldsymbol{X_i}$ *in* $\boldsymbol{X}$ **do**
        $\mathbf{V_i} \leftarrow \boldsymbol{V_i}$ + ($\boldsymbol{X_i}$ - $\boldsymbol{X_{best}}$) * $\boldsymbol{Q_i}$
        $\mathbf{X_{temp}} \leftarrow \boldsymbol{X_i}$ + $\boldsymbol{V_i}$
        if *rand(0,1) > Pulserate)* **then**
            $\boldsymbol{X_{temp}} \leftarrow$ *New Random Solution from Levy Distribution * 0.001*
        $\boldsymbol{fitness} \leftarrow runTrial(\boldsymbol{X_{temp}})$
        if *fitness <* $\boldsymbol{F_i}$ **and** *rand(0,1) < Loudness* **then**
            $\boldsymbol{X_i} \leftarrow \boldsymbol{X_{temp}}$
            $\boldsymbol{F_i} \leftarrow \boldsymbol{fitness}$
            if $fitness > \mathbf{F_{best}}$) **then**
                $\boldsymbol{X_{best}} \leftarrow \boldsymbol{X_{temp}}$
                $\boldsymbol{F_{best}} \leftarrow \boldsymbol{fitness}$

Figure 37: *The algorithm employed in this experiment, inspired by Fister's velocity adaptations of the Bat Algorithm [Fister et al., 2013].*

age error, while not necessarily indicative of passing or failing, implicitly tests important properties of a controller. The rise time, settling time, overshoot and steady state error all impact the average error and are four important properties which one would test in a domain specific environment. Therefore, average error suffices as a good indicator of improving performance. Modifying the curriculum to account for particular properties would be simple to do, if knowledge of the domain is provided to indicate which of the four properties is most important to control.

Top Level: Position Control The position curriculum had three trials that the candidates were tested on. Between each of these trials, the controller and position of the robot were reset. The reset point was the middle point of the range of movement, which is approximately 40 degrees. The error over time for all three trials was recorded and averaged.

- Move to 5 degrees, 8 second time limit
- Move to 55 degrees, 8 second time limit
- Move to 95 degrees, 8 second time limit

Bottom Level: Velocity Control The Velocity Control curriculum was designed as one continuous trial, so changes in behaviours are accounted for in the curriculum. The agent began at the middle point as before, but then each of these tests immediately moved onto the next. Again, the average error over the whole period was the score for those parameters.

- Maintain a velocity of -0.3m/s until past -10 degrees.

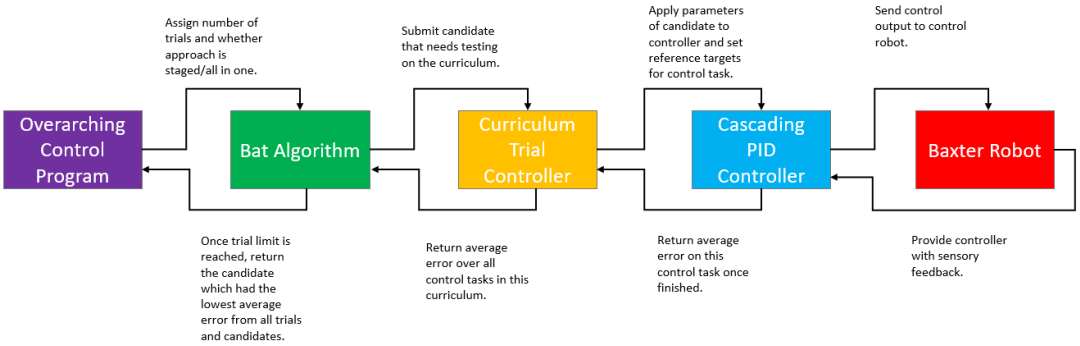

Figure 38: *A Flow Chart showing the program flow of the combined architecture. Each arrow indicates some information or a command being sent from one part of the architecture to another.*

- Maintain a velocity of 0 for 3 seconds.
- Maintain a velocity of 0.6m/s until past 110 degrees.
- Maintain a velocity of -0.6m/s until past -10 degrees.
- Maintain a velocity of 0.3m/s until past 110 degrees.
- Maintain a velocity of 0 for 4 seconds.

The Full Architecture An overarching control program assigns which optimisation approach the Bat Algorithm will use, staged or all in one, as well as the number of trials to be run. The Bat Algorithm produces possible parameter combinations (hereafter called candidates) which need to be tested. When one needs testing, it is sent to the curriculum trial controller, which tests the candidate on the curriculum through a series of control tasks. On receiving a candidate to test, the curricula trial controller will set the parameters of the Cascading PID Controller to those of the candidate. Then, it passes reference signals to the Cascading PID Controller for each control task. It will keep doing this until all control tasks that are part of this curriculum have been sent. Once the Cascading PID Controller receives reference signals for a control task, the Cascading PID Controller sends control signals to the robot which returns sensory feedback. From this feedback, the Cascading PID Controller calculates the average error over the period of the control task. This average error is fed back to the curricula trial controller, which then averages the average error across all the control tasks. This is fed back to the Bat Algorithm, which feeds into whether this candidate should be kept or discarded. Eventually, when all the trials are complete, the Bat Algorithm feeds back to the overarching control program the best candidate at minimising average error.

5.3.4 Results and Discussion

Execution Time Due to the size of Baxter and the heavy weight of the limbs, each test on the curriculum required 20 to 30 seconds. With 20 trials and 20 candidates, this results in a running period of several hours, which is not suitable across all robotics solutions. However, in each run of the algorithm, effective candidates were found in the first two to four trials. Each staged approach took only two to four trials to acquire a

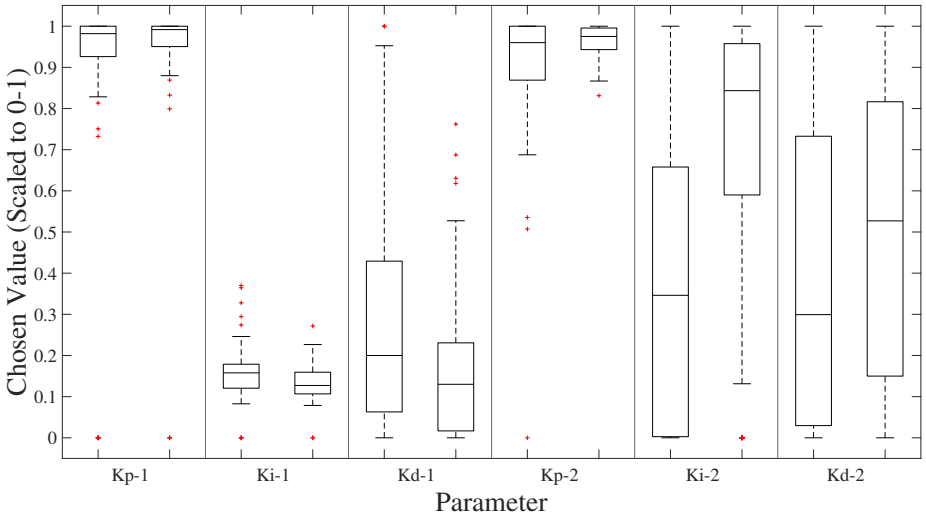

Figure 39: *A graph showing the spread of choices for the six parameters chosen by each approach. For each parameter labelled on the x-axis, there are two boxplots representing the spread of parameter values chosen. The middle line represents the mean, the box's upper and lower bounds represent the 75th and 25th percentile respectively, and the upper and lower whiskers are the upper and lower adjacent values respectively. The left box in each Section indicates the chosen values by the all-in-one approach and the right box represents the values chosen by the staged approach. The most notable difference is the choice of K_i in the velocity controller, Ki-2, where the staged approach went for an integral-heavy parameter set.*

candidate that was below or equal to 110% of the average error of the eventually found best candidate. For the all-in-one approach, this was between four and eight trials. This presents a quicker time frame than the maximum number of trials used, but is important to test the effectiveness in situations where greater time is allowed. Furthermore, many autonomous robots will be able to act faster than Baxter, whose joints are not built to be quick or responsive. With a robot which enacts trials quicker combined with the low number of trials required, this reduces the time to be effective from hours to minutes.

Comparison of the Chosen Parameters The Staged Approach had a separate training procedure for the three parameters in the lower controller. However, the values chosen for the lower controller influenced the choices of the second stage of training. Given this, it is notable that both approaches found similar parameters for the higher controller. This can be seen in the first three pairs of boxes and means (labelled kp-1, ki-1 and kd-1) in Fig. 39. For each pair in Fig. 39, the all-in-one approach (orange) has chosen parameters similar to the staged approach (blue).

The most notable difference between the two schemes is in the K_i value for the lower controller indicated by the third and fourth columns from the right in Fig. 39. The staged approach on average has a much higher K_i value, whereas the all-in-one approach favours a lower value. The integral typically causes the controller to overcome steady state error which would be expected in a velocity controller. The amount of force required to counter a small error (or apply a small amount of velocity) is more than the proportional term

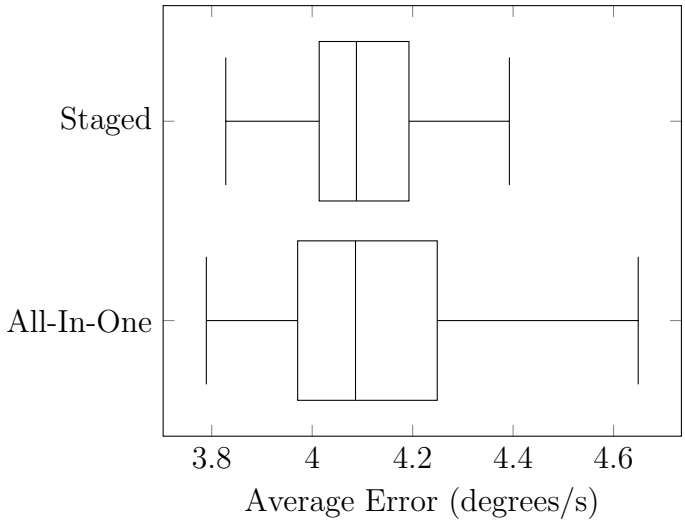

Figure 40: *Box Plots of the average error of the best solutions found by the all-in-one and staged approaches. The middle line represents the mean, the box's upper and lower bounds represent the 75th and 25th percentile respectively, and the upper and lower whiskers are the upper and lower adjacent values respectively.*

would allow. As such, an integral is expected here to allow error to build and apply more torque to the joints. The slightly higher K_d value is also expected as a result, as the K_d value offsets the overshooting a high K_i value can often cause.

Comparison of Error Both medians are similar with no significant difference, but the spread of results differs. The all-in-one approach has a greater degree of both excellent and poor results further from the median. This is as hypothesised, as the higher dimensionality of the search space allows for all possible combinations to be considered. However, the dimensionality also increases the complexity of the search space. Given the initial candidates are randomly selected, these can be a poor selection from the state space and not allow the algorithm to appropriately minimise error.

When comparing all the solutions and their scores from both approaches, neither has significantly lower error than the other. However, The poorest 25% of solutions from the staged approach perform significantly better than the poorest 25% of solutions from the all-in-one approach. The best 25% of the all-in-one solutions significantly outperform the best 25% of the staged solutions.

This result is applicable in fields where consistent reoptimisation of parameters is preferred, as poor results can result in catastrophic failure and are not worth the occasionally better performances such as autonomous robotics. Furthermore, it is notable that the solutions do not have distinctly different medians given the staged approach is computationally simpler. Two three-dimensional search spaces are less complex to traverse than one six-dimensional search space, meaning the same results on average are being achieved on a simpler version of the problem. The staged approach is computationally simpler as it does not consider every possible combination of all six parameters, but rather optimising three independent of what values may be selected for the other three. This could theoretically limit the controller by not allowing it to find suitable parameter combinations between the higher and lower controller. However, given the medians are

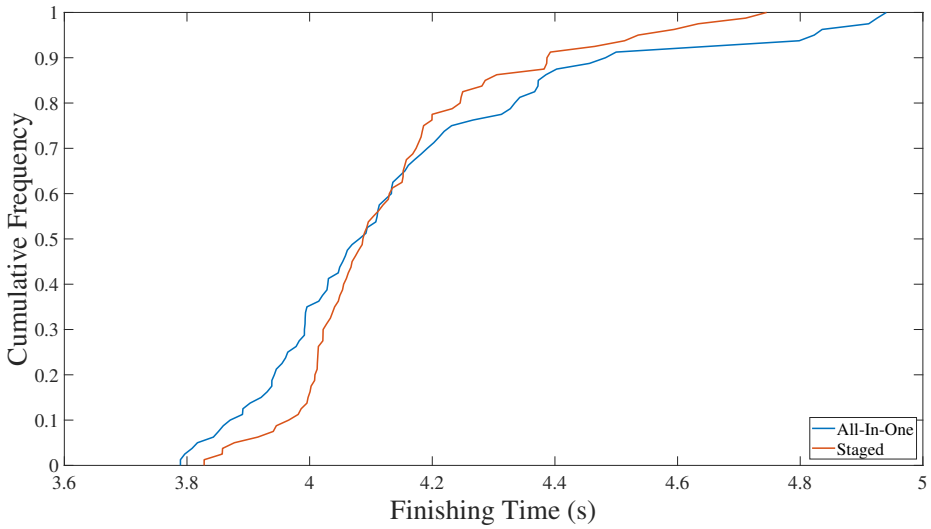

Figure 41: *A graph showing a Cumulative Distribution Function of the average error of the best solutions found by the all-in-one and staged approaches.*

similar, this indicates that the poor results from the higher dimensionality offset the benefits of having access to more parameter combinations.

5.3.5 Conclusion and Future Work

In this paper, results have been presented comparing a staged parameter selection approach with the standard all-in-one approach for control of a joint in a robotic arm. It has been shown that the staged approach has more consistent results, particularly that the worst solutions of the staged approach are better than the worst solutions of the all-in-one approach. The staged approach is computationally simpler yet retains a similar median performance. The value of consistency in autonomous robotics has been discussed. However, what remains to be determined is the extent to which this effect would scale and how effective these candidates are in a general setting. This paper shows the effect of different parameter optimisations and how they're affected by dimensionality, but does not express how effective the controllers are per se. A set of trials aimed at testing generalised performance would need to be used for this.

It is not concluded whether the resistance to the effects of dimensionality in the staged approach scales to higher dimensions. When the algorithm struggles to search the space due to high dimensionality, inconsistency will occur. However, as the dimensionality continues to increase, the effectiveness of the solutions should worsen rapidly. Further work needs to be done to demonstrate how resistant the staged approach is in higher dimensional searches.

Finally, more formal work could be performed to detail exactly what a good curriculum is. It is evident that the curriculum designed here met the purpose of maintaining good results by achieving consistent staged parameter optimisation. However, further discussion and methodological analysis is necessary to identify what comprises an effective learning curriculum.

5.4 Using Hierarchies to Provide Better Generalisation in Robotic Control Tasks

5.4.1 Abstract

A robotic agent is required in many fields to generalise from their training environment to their working environment. When training time is limited, such as in autonomous robotics with new environments, robots can struggle on problems unseen in their training. Previous studies have used control hierarchies to implement a staged approach to parameter optimisation with more reliable results, but this approach has not yet been used to learn control parameters with limited training time. Two hierarchical controllers have been compared, one was trained using a staged hierarchical approach and the other with all parameters optimised at once. The controllers were trained on controlling the position of a single joint using one actuator. The solutions to both approaches were tested on a set of tasks that had not been learned in the training set. The results show that the staged approach has more candidates that retain high performance in the test set than the all-in-one approach, as well as having a better average performance. Lightweight solutions to generalising from the training set to the real world are scarce, limiting their application in autonomous robotics where environments are unknown or cannot be simulated. This investigation demonstrates the effectiveness of a hierarchical approach, showing effective learning without extensive training time or complex priors.

5.4.2 Introduction

Training on Limited Data in Robotics Parameter optimisation is a challenge in robotics as searching for correct parameters can be challenging in high-dimensional spaces [Trunk, 1979, Wang, 1997]. When a robot is learns parameters from a particular training set, the robot risks learning the data too rigorously, particularly if the robot is trained with too many trials [Janabi-Sharifi and Hassanzadeh, 2011] or too few, resulting in the robot copying known patterns without generalising to the environment. Machine Learning has a variety of approaches that aim to improve generalisation by providing the robot with priors that help express the problem domain. These approaches result in effective parameter selection where the problem domain has generalisable or predictable behaviour [Matsubara et al., 2011, Lizotte et al., 2007, Martinez-Cantin et al., 2007], but require selection of key meta-parameters or priors that suit the problem [Chatzilygeroudis et al., 2018]. Deep Learning has been used with these priors to significantly improve results [Wahid et al., 2019, KONDO et al., 1999]. Some approaches noted that a curricular approach to training the robot may further improve generalisation [Levine et al., 2015]. A solution applied to Artificial Neural Networks known as Dropout successfully allows the AI to generalise [Ashiquzzaman et al., 2017, Huynh, 2017]. Dropout randomly removes nodes from contributing to the network temporarily, forcing the training to be spread across the network [Srivastava et al., 2014]. Once trained, the nodes contribute their outputs in the network proportional to their dropout probability. This process avoids weights becoming too large due to being trained on too much data and spreads the training across the network. This approach applies to Artificial Neural Networks, but is not as applicable to other controllers. Furthermore, the dropout probabilities need to be manually assigned, which if selected poorly can change the effectiveness of the training. Other approaches, particularly in Supervised Learning, focus on how the training data is processed [Zhao et al., 2019]. In particular, Cross-Validation is a popular choice for

dealing with poor generalisation. Cross-Validation has an additional data set which the robot uses, known as the validation set. The robot is trained on the training set but is also regularly tested on the validation set, to check the robot can generalise to the tests in the validation set [Santos et al., 2018, Lever et al., 2016]. This approach works across many controllers, but requires even more data than standard approaches to populate the validation set. This isn't appropriate for contexts where available training data is limited. In summary, the existing approaches require extensive time or large amounts of data to allow time-limited learning that transfers well to the environment.

Hierarchical Control Hierarchical Control as a field has considered developmental approaches to optimisation [Digney, 1998, Brooks, 1986, Morimoto and Doya, 2001]. Hierarchical arrangements are common in robotics and the field of AI, with hierarchies being used in cognitive systems [Pairet et al., 2019] and hierarchical Reinforcement Learning [Basu et al., 2019]. Fields such as Perceptual Control Theory have noted that optimisation of higher levels of a hierarchy requires the lower levels to function [Powers, 1974]. Hierarchical modular control nodes have been used to manage multiple control tasks with self-identification of the hierarchy [Digney, 1995, Digney, 1996]. Hierarchically staged parameter selection has been shown to produce more consistent results on the training set [Hawker and K Moore, 2020] but not whether a staged optimisation based on the hierarchy presents better generalisation when using limited testing data. The work by Hawker [Hawker and K Moore, 2020] shows suitable parameter selection for generalised behaviour on a limited training set, but this has not been methodically tested.

Figure 42: *A pair of images of the Baxter Robot. The left image shows the robot, the controlled joint (s0) and the location that was being controlled (e1). The control system applies torque to the joint s0, which rotates the arm in the XZ plane. This moves the e1 joint around Baxter's body. The control system is controlling the relative angle of the e1 joint with respect to Baxter. The right image visualises the effect of applying force to the s0 joint to demonstrate the domain of the task.*

Summary Requiring complex priors or additional datasets is unfeasible in many applications of robotics. Robots that require quick adaptation will not have large datasets. Robots trained in simulations can fail when exposed to the real environment due to the simplicity of the simulation in comparison. This paper demonstrates an approach to parameter optimisation that is staged through a control hierarchy. This approach was compared against a standard all-in-one approach to parameter optimisation on performance on a generalised problem on which the robot had not been trained.

5.4.3 Experimental Setup

Baxter Robot The Baxter Robot was used for this experiment, with the task undertaken described in Fig. 42. The complexity of the task is compensating for the weight of the arm, which makes building speed and stopping at precise points difficult. The rest of the joints in the arm are held in place, so that the controlled joint is the only actuator being considered.

Proportional-Integral-Derivative Control A Proportional-Integral-Derivative Controller is a controller that is used to solve many problems in control engineering [DiStefano et al., 1967, Arimoto, 1995]. PID Controllers have been applied to many control tasks, such as controlling robotic limbs or aerial robots [Hernández-Guzmán et al., 2008, Bouabdallah et al., 2004, Sun and Er, 2004, Cervantes and Alvarez-Ramirez, 2001].

A PID Controller is a form of closed-loop system, where the output of the system has an impact on the input signals to the controller, allowing constant and continuous control of the incoming input [Wiener, 1948]. A closed-loop controller requires an input that is being controlled and a desired value for that input, which are i and r respectively. $e = r - i$ is used to calculate the error, which is the difference between the desired and actual input value. From the error, the system output u is calculated. If $u(t)$ is the control output at time t, $e(t)$ is the error at time t and k_p, k_i and k_d are parameters, $u(t)$ is defined as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Cascading PID Control Cascading PID Control is multiple PID controllers that are hierarchically connected, such that at least one PID Controller's control output is the reference for another PID Controller. Complex control problems have been solved robustly with Cascading PID Control [Sahrin et al., 2018, Reyes-Ortiz et al., 2020, Deng et al., 2019]. Cascading PID Control is common in problems with multiple inputs that require controlling, as multiple individual controllers can be employed to control each input. The relations between these inputs and their control decides how the controllers need to be hierarchically arranged.

Two PID Controllers were used in this experiment, forming a Cascading PID Controller. The task of controlling the position of the shoulder joint (labelled e1 in Fig. 42) is handled by the higher controller, which sends a reference to the lower controller. The lower controller controls the velocity of the shoulder joint (labelled s0 in Fig. 42) since the velocity of the s0 joint directly affects the relative position of the shoulder joint to the body. A diagram of the Cascading PID Controller can be seen in Fig. 43.

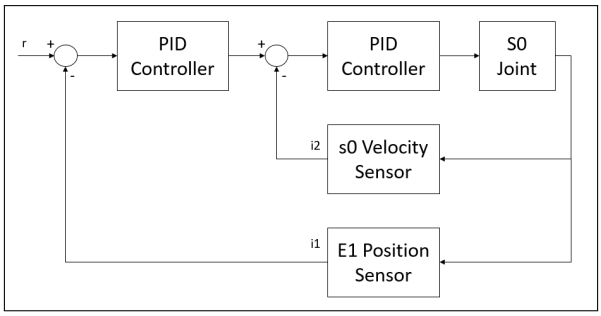

Figure 43: *A diagram showing the cascading PID controller used in this experiment. The reference, r, is the desired target location of the shoulder joint. The input i1 is the actual location of the shoulder in relation to the body, ranging from 40° to -140° due to the limited reach of the arm. The output signal from this controller is the reference to the lower controller, which controls the speed of the joint s0. The input signal i2 is the actual velocity of the s0 joint, which goes from -0.9m/s to 0.9m/s due to safety limiting. The output of this controller is sent as a torque signal to the s0 joint, prompting movement.*

The Cascading PID Controller has several parameters that need optimising to achieve this task. For each controller, there are three parameters (k_p, k_i and k_d). The standard approach is to use a learning algorithm which trains all the parameters together, searching for different combinations of all parameters. The contrasting approach being tested in this experiment is optimising the parameters hierarchically. Specifically, that is training the lower controller's parameters first and then training the higher controller's parameters.

Training A training algorithm is required that can take the control task and controller and identify good parameter sets. Furthermore, an approach is required that can train the controller by changing all parameters at once, or perform a staged training approach. Evolutionary Algorithms, inspired by the Genetic Algorithm, benefit from good convergence in a small amount of trials and thus are commonly used in robotics. The Bat Algorithm, which is inspired by the use of echo-location in bats [Yang, 2010], has particularly good convergence in limited trials. These properties have made the Bat Algorithm useful in robotic control [Rahmani et al., 2016].

The Bat Algorithm is used to train the controller, with one Bat Algorithm training all the parameters in one run and the other training the lower controller in the first run and the higher controller in the second. Modifications to Yang's original work include using a velocity-based candidate updating system and a levy-flights random walk when candidates are to be reset [Fister et al., 2013, Fister et al., 2014]. Each approach has 30 trials maximum, meaning that the approach that trains all the parameters at once has 30 trials to produce the best candidate possible, whereas the progressive approach splits the 30 trials equally on the two runs. See Fig. 44.

Designing Curricula for Developmental Learning The curricula are used by the learning algorithm to evaluate the effectiveness of a controller in the task domain. A curriculum is built for each controller, based on the input they are required to control and what values are a good representation of the task space. The all-in-one approach only used the position control curriculum to train both controllers, with the aim of the

Algorithm 2 Bat Algorithm

Require: Loudness Parameter (1)
Require: Pulserate Parameter (0.75)
Require: Number of bats, $\mathbf{N_b}$ (20)
Require: Number of trials, $\mathbf{N_t}$

```
1: X ← Establish Bat Populations randomly X_i for 1...N_b
2: F ← Establish F_i as runTrial(X_i) for 1...N_b
3: Q ← Establish Q_i between 0.0 and 0.2 for 1...N_b
4: while Loop count below N_t do
5:     for X_i in X do
6:         V_i ← V_i + (X_i - X_best) * Q_i
7:         X_temp ← X_i + V_i
8:         if rand(0,1) > Pulserate) then
9:             X_temp ← Random Levy * 0.001
10:        fitness ← runTrial(X_temp)
11:        if fitness < F_i and rand(0,1) < Loudness then
12:            X_i ← X_temp
13:            F_i ← fitness
14:            if fitness > F_best) then
15:                X_best ← X_temp
16:                F_best ← fitness
```

Figure 44: *The Bat Algorithm used in this experiment, inspired by modifying velocity of the candidates [Fister et al., 2013, Fister et al., 2014]. Random Levy refers to drawing a new selection of parameters randomly from a Levy Distribution.*

bottom controller implicitly picking up parameters that suitably allow position control. The staged approach trains the lower controller on a specific curriculum built for that controller and those parameters are locked in. Then, the higher controller is trained with the bottom controller aiming to have effective parameters in place already.

Top Level: Position Control The position curriculum had three runs with a reset occurring between each run. The reset involves the robot returning to the middle of the range of movement and having no velocity. The score on this curriculum is the average error over time in each run, then the average of that over all three runs.

- Reference: 5 degrees. Duration: 8 seconds
- Reference: 55 degrees. Duration: 8 seconds
- Reference: 95 degrees. Duration: 8 seconds

Bottom Level: Velocity Control The curriculum for the bottom controller did not reset between runs, but rather was one continuous run. This was to ensure behavioural transitions between differing references were tested. The average error over time over the whole run was the candidate's score. Below is the list of references during the run, with each detailing the reference value and the condition for moving onto the next reference in the list.

- Reference: -0.3m/s. The reference remains until the position of joint e1 is at or below -10 degrees.
- Reference: 0m/s. The reference lasts for 3 seconds.
- Reference: 0.6m/s. The reference remains until the position of joint e1 is at or above 110 degrees.
- Reference: -0.6m/s. The reference remains until the position of joint e1 is at or below -10 degrees.
- Reference: 0.3m/s. The reference remains until the position of joint e1 is at or above 110 degrees.
- Reference: 0m/s. The reference lasts for 4 seconds.

Testing The Bat Algorithm produced candidates which performed best on the training curriculum. However, a test set was required to evaluate each candidate's performance in a general setting. This required similarity to the training curriculum, but with control tasks that had not been explicitly tested on. In the domain of robotics, it is common for robots to require precise movement one after the other without strict resets. The training curriculum provided resets between tasks, meaning moving to a variety of locations was implicitly but not explicitly tested. A test set was devised that had the controller reach 10 points as soon as possible, dubbed the 'Slalom Test'. The Slalom Test is a series of references which the controller must reach as quick as possible in sequence. Reaching one reference and maintaining within a close range of the target for a set time unlocks the next reference. Moving on to the next reference occurs when the controller maintains being within 3 degrees of the target for 1 second, or after a 20 second timeout. This

means the effective worst score is 200 seconds on the whole slalom. Four Slaloms were derived, so that each controller could be tested on different combinations of reference values. The first slalom was hand-engineered to provide as much variation as possible between adjacent targets. This included some small and large changes in reference in both positive and negative directions. The three remaining slaloms were randomised versions of Slalom One, to provide differing coverage. The slalom's references are listed below, with each list being in order and each item in the list being the reference for the angular position of the e1 joint that the control system has to achieve.

- **Slalom One References:** 5°, -15°, 25°, -40°, 55°, 75°, -15°, 0°, 35°, -25°
- **Slalom Two References:** 55°, -40°, 0°, -15°, 75°, -25°, 25°, 5°, -15°, 35°
- **Slalom Three References:** 55°, 25°, -15°, 5°, -25°, 75°, 0°, -15°, -40°, 35°
- **Slalom Four References:** 5°, 35°, 75°, 0°, 25°, -15°, -40°, -15°, -25°, 55°

Each candidate was trained on all four slaloms, giving four scores to test how well the controller did when transitioning to the test set. The Bat Algorithm was run 80 times on each approach, meaning there were 360 test set scores for both approaches.

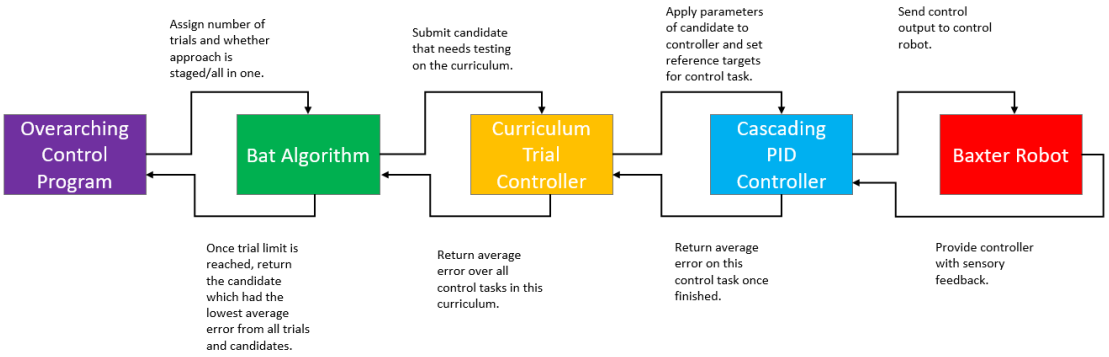

Figure 45: *A Flow Chart showing the program flow of the learning system, including the training of candidates, the control system and the robot.*

The Full System The overarching control program handled the program code for instantiating the experiments, including parameters and the training regime (either all-in-one or progressive parameter optimisation). The overarching control program instructs the Bat Algorithm of the training regime and number of trials, where the Bat Algorithm controller instantiates the training. The Bat Algorithm controller needs to test candidates on the training set to evaluate their fitness. A Curriculum Trial Controller handles this, which is the program that takes a single set of parameters and returns their fitness score. The Curriculum Trial Controller, based on the training regime, selects the appropriate curriculum to test on and instantiated a Cascading PID Controller with the parameters that need testing. The Curriculum Trial Controller then begins the task, allowing the Cascading PID Control to send control signals to Baxter. Baxter responds with sensory feedback so the controller can complete the control task, forming a control loop. When the end condition for the trial is met, the fitness of the candidate is calculated and returned to the trial controller. Once all trials handled by this controller are complete, the average score of those is returned to the Bat Algorithm to evaluate the fitness of the

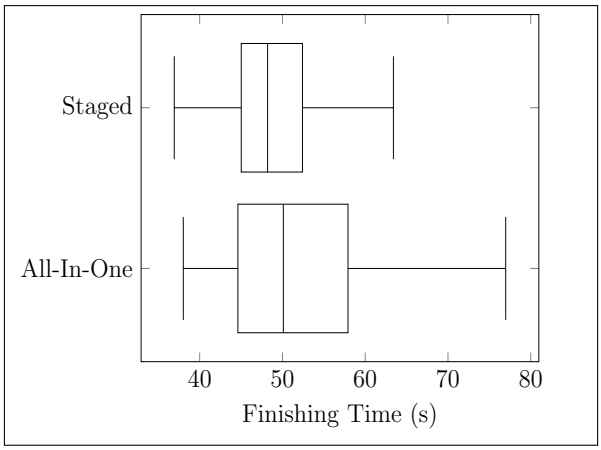

Figure 46: *Box Plots of the finishing time on the generalised test for the best solutions found by the all-in-one and staged approaches.*

candidate. Over many trials, the Bat Algorithm returns the candidate that performed best and returns this to the overarching control program.

5.4.4 Results

Scores on the Generalised Slalom Test The box plots in Fig. 46 show the average finishing time of the best solutions from both approaches on the test sets. The staged approach has significantly better performance than the all-in-one approach. The median, 25th percentile and lower adjacent are similar, but the upper adjacent and 75th percentile deviate from each other. This implies either that there are more poorly performing candidates in the all-in-one approach or there are a similar number of poor candidates but they perform worse. The Cumulative Distribution Function shown in Fig. 47 better demonstrates the statistical difference in results. The better performing solutions of both approaches score similarly. However, there are more poor performers that overtrained in the all-in-one approach.

Comparing Top Candidates on the Training Data The top scoring 25% of candidates on the training set were compared on the test sets. Even though the all-in-one approach's top 25% solutions outperformed the staged approach's top 25% solutions on the training set [Hawker and K Moore, 2020], this trend reverses on the test sets. There is no significant trend correlating score on the training set and test set, with candidates of a range of scores on the training set generalising poorly to the test sets.

Examining the Chosen Parameters There were slight differences in the chosen parameters between the two approaches, shown in Fig. 48. The integral parameter of the velocity controller (the fifth set of boxplots in Fig. 48) have different ranges of values. A high integral value is expected, given the problem domain. The build up of torque is required to produce velocity, which is required to produce a change in position of the elbow joint. Many of the failed candidates of the all-in-one approach had a near-zero or zero proportional parameter in the lower controller. Having a near-zero proportional value is uncommon in controllers, since the difference between the desired and actual input value should be indicative of the desired effort required to achieve the control goal. Thus,

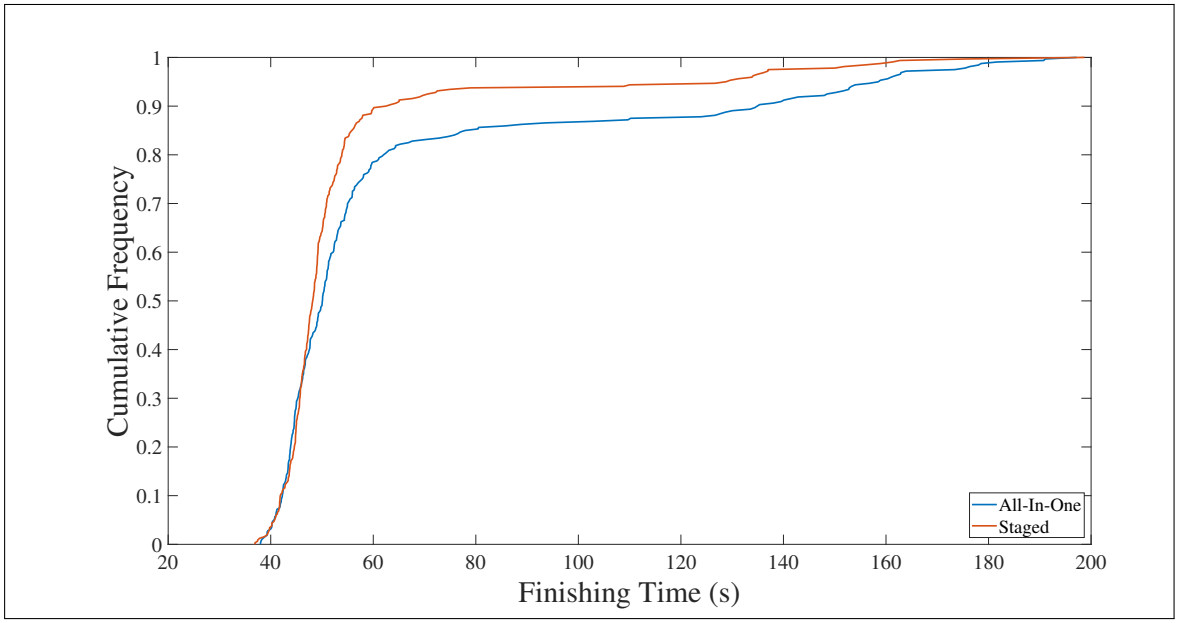

Figure 47: *A graph showing a Cumulative Distribution Function of the average error of the best solutions found by the all-in-one and staged approaches on the generalised test.*

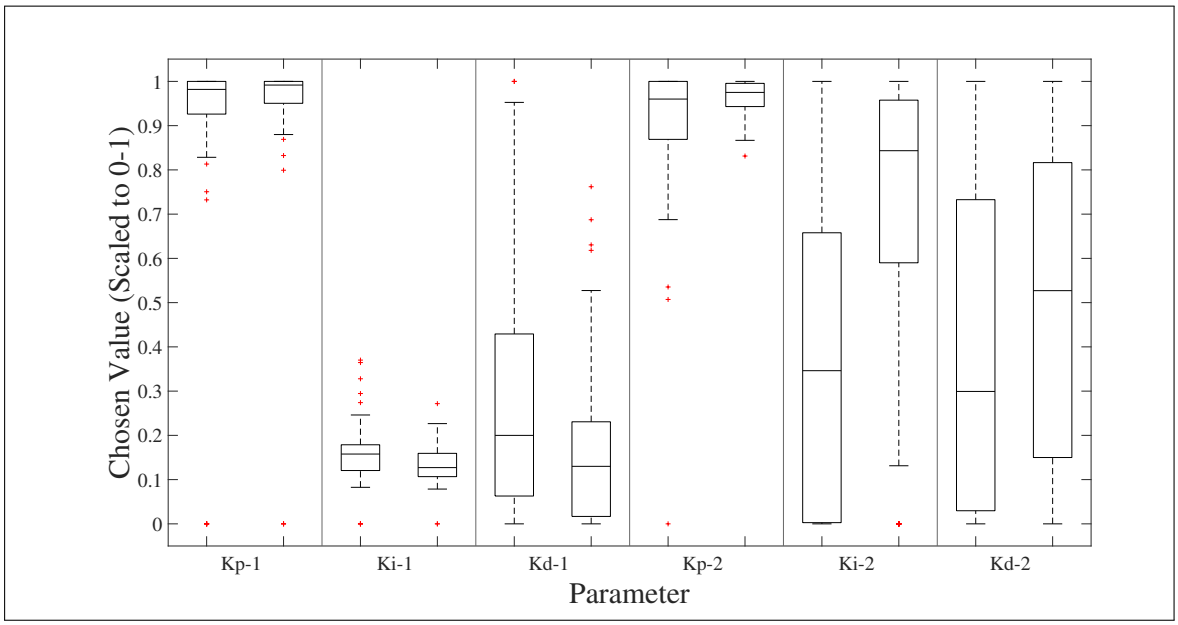

Figure 48: *A graph showing the six parameters chosen by the staged and all-in-one approaches. For each parameter labelled on the x-axis, there are two boxes representing the spread of parameter values chosen. The left box indicates the chosen values by the all-in-one approach and the right box those chosen by the staged approach.*

it is a possible indicator of a poorly optimised controller. Zero staged approaches selected a zero or non-zero proportional parameter for the lower controller, indicating a suitable training phase that avoided selecting poor parameters. Both approaches produced similar parameter values for the higher controller, even though both of them produced the values for the lower controller differently. This shows that the higher controller is still able to select appropriate parameters for effective control even when the lower controller's parameters are trained separately.

5.4.5 Conclusion and Discussion

This paper has shown that using a hierarchically staged approach to parameter optimisation in a control problem secures solutions which learn significantly better in time-limited learning than a standard all-in-one approach. The values found by the staged approach solution's parameters better represent what an engineer would heuristically pick than those of the all-in-one approach. Time-limited learning is a significant problem in autonomous robotics, with environments being unpredictable, hazardous and allowing limited training. Therefore, effective training to situations unseen in the training set is key. Hierarchically progressive optimisation has been shown to effectively handle this problem, significantly limiting the number of candidates that transferred poorly to the problem environment.

Further experiments could be done to test if the results provide better generalisation in a version of the problem not represented at all in the test set, such as with limb alterations or failure. Furthermore, the staged approach could train an agent in such a generalised way that it avoids issues of transferring from simple simulations to the real world, known as the Reality Gap. The robot with a different arm configuration could be used as a test set, such as the arm being outstretched. These can be undertaken to test if progressive optimisation extends to tasks unseen in the training set.

It is important to test how well the developmental approach scales to bigger hierarchies, given the successful performance in a two level hierarchy. Larger hierarchies are often avoided due to the complexities of the parameter search space. It is possible a staged approach makes large hierarchical structures easy to train.

Acknowledgement

With thanks to Adam Hartwell and Jonathan Aitken, for their technical support and advice on Control Engineering with the experiments conducted in this chapter.

The research that produced these results was funded by the Engineering and Physical Sciences Research Council (EPSRC).

5.5 Summary

The previous chapter showed how hierarchies can be automatically derived but did not show that they can be progressively optimised up the hierarchy. This chapter defined experimental work to show that progressive optimisation of parameters gives consistent results and results that perform well on an unseen test set. A robot arm with a two-tier hierarchy learned a movement task with a limited time to train. Even with the limited training time and curriculum, progressive optimisation yielded competent and consistent results which transferred well to unseen versions of the problem. This has shown the effectiveness of the third step of DOSA, testing all three parts of the methodology.

6 Contributions

The main contributions of this Thesis are: a methodology for automatically deriving hierarchies (DOSA), hierarchical parameter optimisation is effective, and that hierarchical parameter optimisation produces better results on unseen situations in robots.

6.1 Revisiting the Thesis Objectives

6.1.1 What is a Hierarchy?

A hierarchy is semantically composed of dependencies. A dependency is where the control of one input requires the control of another input. Therefore, the controller of the dependent input must have some command over the controller for the input it depends on. The literature showed inconsistent and incorrect applications of hierarchies, explaining ineffective results in the literature.

6.1.2 Can the Process of Deriving a Hierarchy be Automated?

The process can be automated. DOSA was produced, which derives hierarchies from having an agent explore their world to detect dependencies. Once the dependencies are detected, the scope of possible hierarchies is reduced. This allows the agent to progressively explore hierarchical options in a realistic timeframe. Multiple examples were derived, including the robotic arm problem from Living Control Systems III [Powers, 2008] which has a large hierarchy. DOSA derived a very similar hierarchy to one that is robust in solving the problem.

6.1.3 Can the Process of Deriving Control Parameters in a Hierarchy be Automated?

The process can be automated through DOSA's third step. The parameters can be optimised progressing up the hierarchy, because of the dependency relationships. Experiments on a robotic arm were undertaken to prove it makes the computational space of control parameters easier to navigate and secures robust parameters when trained on variants of the problem.

6.1.4 What are the Benefits of Using a Hierarchical Approach?

Hierarchies have been shown in the literature to break down complex problems, enabling buildable solutions to complex problems. Experiments on a robotic arm have shown that hierarchies allow for progressive parameter optimisation, which reduces the computational complexity of the parameter search. It also results in parameters that perform better on variants of problems, rather than over-learning the training data.

6.2 DOSA, standardising hierarchical control

Hierarchies, when built correctly, have allowed controllers to solve complex problems with simpler systems by having the complexity of the system distributed over multiple sub-functions. However, this power was locked behind a confusing heuristic process for deriving and producing hierarchies. This Thesis has unlocked this process and made it methodologically consistent, through DOSA. This allows hierarchies to be used on

complex problems and consistently achieve robust results. Without DOSA, engineers must manually define the hierarchy based on expert knowledge and the problem domain. Even the most learned engineers can fail to derive an appropriate hierarchy, resulting in poor control. DOSA solves this issue and additionally unlocks the capacity of hierarchies to people without years of expertise in control engineering.

DOSA was demonstrated on a large and effective complex hierarchy that was heuristically derives and the resulting hierarchy from DOSA was remarkably similar, when there were uncountable possibilities for the hierarchies chosen. Furthermore, DOSA has been demonstrated on several theoretical problems to show effectiveness in deriving a hierarchy. DOSA was demonstrated to be a full package, also providing hierarchical parameter optimisation to take a control agent from a selection of inputs and outputs to a fully functional control system.

6.3 How to Make Curriculums for Hierarchical Controllers

Understanding of learning in hierarchical structures is lacking in control literature. While many engineers in practice would hierarchically optimise parameters, this was not tested or well understood. This Thesis outlines the process of developing a curriculum for each level of the hierarchy, allowing an engineer to build a curriculum that would allow learning of parameters effectively in short time frames. Many domains of robotics struggle with time-limited learning, such as autonomous robotics. Living agents do not struggle with generalisation or time-limited learning, showing that different approaches were needed.

This Thesis demonstrated that curriculums based on the individual control subsystem's control objective are an effective tool for learning hierarchical structures, moreso than optimising the entire hierarchy based on the highest control subsystem's objective. This again is a stumbling block that limits many effective controllers and results in poor control. This Thesis shows how parameters in hierarchical systems can be effectively learned by deriving curriculums, resulting in control engineers being able to make better control systems more often.

6.4 Hierarchical Optimisation Generates Effective Parameters

Hierarchical optimisation of parameters was shown to be an effective parameters for both the control problem and unseen versions of the problem. Not only has this Thesis defined how to use curriculums to hierarchically optimise parameters, it has shown that this produces more effective control parameters on unseen domains. This is a clear and powerful benefit to using hierarchies that has not been seen in any literature, reinforcing the value of using DOSA and hierarchies in general.

The experiment showed that when the robot was trained on an unseen domain that was similar to the training problem, the agents trained with hierarchical parameter optimisation transferred to the new domain significantly better. The learning was performed in a very limited time frame, also showing hierarchical parameter optimisation's resilience to a lack of training data. While the previous experiments showed how hierarchies can be done consistently, this work demonstrated why hierarchies are an excellent control choice.

7 Future Work and Conclusion

7.1 Analysing Different Modular Functions with DOSA

DOSA has shown prowess in control domains and generalised learning, but more could be unlocked with DOSA by using other functions in the modular parts of DOSA than those supplied in this Thesis. Machine Learning has a wealth of tools, such as Bayesian Inference, Linear Regression and Deep Learning. Furthermore, Perceptual Control Theory has demonstrable tools for control that could be used in the modular aspects of this Thesis. The reorganisation algorithm implemented in LCS III [Powers, 2008] could be implemented into the hierarchical parameter optimisation.

Each of these approaches could have differing behaviours on the end hierarchy. Furthermore, it strengthens the case for DOSA if it is shown to be effective in deriving competent control hierarchies with many different control subsystems, input-output analysis algorithms and hierarchical parameter optimisations.

An experiment shall be undertaken comparing variants of DOSA with new modular components on the robotic arm problem used in Sections 5.3 and 5.4. Each variant will be tested against the results of the hierarchically optimised controller in Sections 5.3 and 5.4, to see what differing hierarchies occur when the modules are changed for common techniques in control and learning. The variants that will be used are:

- Replacing the PCT Control Subsystems with an Artificial Neural Network
- Replacing the Input-Output Analysis Module with Bayesian Inference
- Replacing the Input-Output Analysis Module with Linear Regression
- Replacing the Hierarchical Parameter Optimisation Module with the reorganisation from [Powers, 2008]

Each of these variants will be tested on the slalom test, to examine how the derived hierarchies transfer to a new problem. The results for root mean square error, rise time, settling time, overshoot and steady state error will be examined. The work will conclude which approaches work best for this domain of problem, educating further uses of DOSA and showing if the methodology has effective results with different modules.

7.2 Can DOSA Produce a Deep Learning Network?

This Thesis has shown the effectiveness of understanding dependencies to build a hierarchy within Control Theory. However, this has applications outside of Control Theory. Levine investigated the uses of curricular learning in Deep Learning [Levine et al., 2015], training Deep Learning Networks in a hierarchical curricular method that produced a controller that allowed a humanoid agent to run and jump over obstacles, which is a challenging task. However, the process by which one can derive the hierarchy or the curriculum was not defined. Approaches in AI that perform control can use DOSA's principles to adapt hierarchical derivation and progressive optimisation to a variety of structures. It is therefore worth concluding whether DOSA could expand its uses to derive and optimise a full deep learning architecture.

Further work will be undertaken to modify DOSA to allow the self-structuring of layers of neural nets, allowing their derivation. This will use DOSA's first and second

steps to derive a hierarchy, then step 3 shall be undertaken but Artificial Neural Networks will be used in place of PID controllers, producing a hierarchical Deep Learning network. The problem domain will be a fully functional robotic arm, used in LCS III.

A suitable contrast is required to make an effective comparison. Therefore, a Deep Learning approach that doesn't use DOSA is required. Two further implementations shall be compared against from the literature. The first will be a deep reinforcement learning implementation, as seen in [Gu et al., 2017] and the second will be based on Levine's work with policy-influenced deep learning, shown in [Levine et al., 2015] and [Levine et al., 2018]. The standard deep learning approaches will use a series of consecutive nets detailed in the previously mentioned papers, shown to be suitable for robotic control. The learning functions will then learn the required neural nets to optimise control of the X, Y and Z position of the robotic arm. The DOSA implementation will produce a structure of neural nets indicated by DOSA's suggested structure based on steps 1 and 2 of DOSA.

For all three structures, they will be trained in an exhaustive model and also in a time-limited setting, giving six structures to test. The stuctures will be tested on their control of the task they were trained on and then on a varied test set, similar to the experiments undertaken in Sections 5.3 and 5.4. The aim is to demonstrate that DOSA generates a suitable and robust deep learning network that learns in less time.

7.3 Testing Perceptual Control Theory and DOSA On The LCS III Robotic Arm

Perceptual Control Theory has shown powerful yet elegant control in complex problems, along with control that mimics the approach of a living agent. The process of producing HPCT structures was known intuitively but not defined. DOSA utilises the principles from PCT to automate the derivation of HPCT structures, filling an empty area in the theory and function of PCT.

DOSA enables PCT's strong results by showing how the hierarchies they produce have been derived and can be derived in future. Powers was able to produce effective control hierarchies with his expertise as a control engineer, some of which are larger in size than many applied hierarchical controllers [Powers, 2008]. This Thesis has shown how such a hierarchy can be derived. DOSA derived a hierarchy for the problem domain of a robotic arm, producing a hierarchy remarkably similar to the HPCT solution produced by powers in LCS III [Powers, 2008]. What remains untested is how effective these controllers are in comparison, despite their similarity.

The hierarchy derived by DOSA will be implemented on the LCS III robotic arm problem, as well as the version originally derived by Powers. Powers' hierarchy may be best suited to /acPCT's reorganisation and DOSA may be better suited to the techniques used in this Thesis. However, it would be an invalid comparison to use differing optimisation techniques for the parameters. Therefore, both hierarchies will be trained twice, once with PCT's reorganisation and once with DOSA's hierarchical optimisation. This gives four hierarchies, each of the two hierarchies with each of the two learning algorithms.

The four hierarchies will be tested on their control of the X, Y and Z position of the end effector of the arm, as this is the highest level goal of the LCS III robotic arm problem. The test will evaluate their mean root square error over the training set as well as a novel test set that has been unseen to the agents, as seen in the experiments in Sections 5.3 and 5.4. The aim is to conclude that the hierarchy developed by DOSA is

equally or more effective at minimising error as well as drawing concluding that DOSA's hierarchical optimisation of parameters is more effective than the standard approach of reorganisation in PCT.

7.4 Modifying Existing Hierarchies: Can DOSA Manage Conflict, Reorganise and Branch?

While DOSA's methods are compatible with modifying an existing hierarchy, this was not detailed or tested. Robots may need to change quickly if their implementation changes after their hierarchy is defined and balanced. In PCT, various concepts are defined for editing an existing hierarchy. Reorganisation is the changing of parameters or connections to suit new control objectives. Conflict is where the controllers are in a state of fighting against each other due to the hierarchy being poorly optimised. Branching is where a hierarchical node may need to change from one hierarchy to another in the same control system. All of these are possible if circumstances change and concepts that would make DOSA more effective.

DOSA will be extended to have a fourth step, which will provide regular maintenance or changes when the controller is active. Reorganisation will occur if significant error is detected in areas of the control system with an inability to control the higher control objective. DOSA will use a variant of step 1 to detect local error in areas of the hierarchy, diagnosing where the control issue is. Using this, the problematic node can then be re-optimised and any hierarchical node above it can have a brief period of re-optimisation to account for any changes in function as a result of the lower node being changed. An extended hierarchy of the one used in Sections 5.3 and 5.4 will be used, with three levels. These will be controlling position of the elbow, the velocity of the shoulder joint, and the torque of the shoulder joint.

This will be tested on a series of example problems:

- The bottom subsystem is poorly optimised. This will demonstrate appropriate diagnosis of local error and reorganisation.
- The middle subsystem is poorly optimised. This will demonstrate appropriate diagnosis of local error and reorganisation.
- The top subsystem is poorly optimised. This will demonstrate appropriate diagnosis of local error and reorganisation.
- The torque output is halved after control output is sent. This will demonstrate appropriate diagnosis of local error and reorganisation, as well as diagnosing on a real problem.
- A new 'imposter' middle node is added. It is also connected to the lowest subsystem and receives signals from the highest subsystem. It is actively working against the control of the other node on the middle tier of the hierarchy. DOSA will need to learn to identify the conflict, learn that the imposter node is fighting against its objective and reconfigure the hierarchy to not include it.
- The higher node is connected to a new lower and middle node that control random signals. Separate to this, the original middle and lower node sit in their own hierarchical arrangement. DOSA must diagnose that there is an issue and reorganise

to have the higher node branch into the hierarchy it is not connected to to solve the problem.

The aim of this experiment is to show competency in each of the domains tested on, indicating that DOSA can solve the problems of conflict, branching and reorganisation as defined in PCT. Once this is done, experiments can be designed to test against solutions available elsewhere in control theory.

7.5 Analysing the Generalisation of Progressive Optimisation

Analysis of the success of optimising parameters progressively was undertaken, showing superior parameter choices compared to non-progressive approaches on both a training set and a test set. It has been shown that progressive optimisation secures parameters that provide robust control and would be chosen by a seasoned engineer rather than those that solve the training task. Progressive optimisation has been heuristically used in practice, but inconsistently and without justification as to if or how this approach is superior. This Thesis has shown the benefits of using progressive optimisation when applying to unseen versions of the problem, as well as resisting dimensionality issues. This allows engineers to use hierarchically progressive parameter optimisation in situations where those limitations have impact, allowing the training of robots performing complex tasks to be made quicker and easier. Robust control relies on properly selected parameters and is a task required over the field of Control Theory. Showing this approach is effective enables control engineers in many disciplines to apply hierarchies and build powerful and adaptive controllers.

To extend this work, it is posited that a progressive approach to optimisation produces better results when transferred over to a variant of the problem, which is known as generalisation. Handling domain shifts is a persistent issue in robotics, particularly in autonomous applications. The control parameters chosen in progressive optimisation were closer to what an engineer would heuristically choose, showing a possibility of transferring better to a generalised domain.

Experiments will be undertaken to show how the progressive parameter optimisation approach compares to the all-in-one parameter optimisation approach when trained on a control problem and then some limitations are placed on the robot. An example of a limitation that would require a controller to be able to generalise to solve would be the robotic arm then being outstretched rather than hanging below. This is a common issue in robotics, as training environments cannot be assured to be the same as the real environment without exhaustible testing. An unmatched capability of living agents is their ability to adapt to new environments, whereas autonomous robots are not able to perform basic tasks outside of their training without extensive work. Hierarchies and progressive optimisation have shown indications of selecting parameters that better generalise, as seen in the selection of parameters for the control of a robotic arm in this Thesis. The integral parameter in the hierarchical approach better suited the problem to allow the controller to build up force when there is an error in velocity. Further work to show this applying in a variety of problems and when proven on a generalised test can be undertaken. If DOSA does secure better parameters for generalisation, it is a significant leap towards properly autonomous robots.

A further application of this work is checking if progressive optimisation provides parameters that provide better control when there is a reality gap. The reality gap is

another common problem in robotics, where the training environment is not exactly like the real environment, meaning the parameters learned by the robot may be inappropriate. This is common when robots are trained in virtual environments or simulators, which is common due to simulators being able to run training trials quicker than a real robot. Showing that progressive optimisation better handles issues related to generalisation would enable control engineers to navigate a limitation that is heuristically countered. The Reality Gap is a limitation across robotics in allowing robots to be effectively and quickly trained to handle tasks. If DOSA's choice of parameters better resists this phenomenon, it allows robots built in simulation to be more suitable for work in the real world once trained in simulation.

7.6 Final Remarks

Control hierarchies are used in practice, but with little consistency due to no process existing to produce hierarchies besides heuristic selection. A hierarchy designed well produces competitive results on complex control problems. This Thesis has defined a consistent method of producing hierarchies that does not rely on heuristic hierarchical derivation and parameter tuning, allowing a control engineer to avoid making mistakes in the hierarchy and parameters of a controller, saving time and also securing robust results. Hierarchies have proved challenging to implement due to the heuristic process becoming increasingly complex as the size of the problem increases. Furthermore, the number of parameters increases as the controller increases in size. DOSA shows that this process is methodological and can be automated, making the benefits of hierarchical controllers accessible to control engineers without attempting many failed hierarchies first. By building the hierarchies correctly through DOSA, the control of complex tasks is not limited to those with decades of expertise nor intense training data requirements. This empowers robotics in fields with little available training data or where the problem is complex enough that engineers will struggle to solve the problem heuristically.

This Thesis shows the effectiveness of using hierarchies and developing them progressively. While common in practice, no publications exist to compare progressive parameter tuning with optimising the parameters all at once. Progressive parameter optimisation has been shown to produce excellent results in this Thesis, as well as outlining how to appropriately build curricula to make this approach effective and reliable. Large hierarchies, despite their effectiveness in breaking down complex control tasks, are avoided due to the optimisation process. This work shows how large hierarchies can be made accessible and possible through a clear hierarchical progression that allows control hierarchies to be applied effectively.

New approaches are required to control, rather than extending existing techniques. DOSA enables control hierarchies to be utilised. Distributing control over multiple controller subsystems has been shown by this investigation to be powerful and DOSA has made this power accessible. Through this accessible functionality to control, complex robotics tasks can be achieved in manageable time constraints and significantly less prone to erroneous control structures being selected.

References

[Abbeel et al., 2007] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. MIT

Press.

[Aicardi et al., 1995] Aicardi, M., Caiti, A., Cannata, G., and Casalino, G. (1995). Stability and robustness analysis of a two layered hierarchical architecture for the closed loop control of robots in the operational space. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 2771–2778 vol.3. IEEE.

[Albus et al., 2004] Albus, J., Barbera, T., and Schlenoff, C. (2004). RCS: An intelligent agent architecture. In *Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems, AAAI Press, no. WS-04-07 in AAAI Workshop Reports*.

[Albus, 1975] Albus, J. S. (1975). A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97(3):220.

[Albus, 1993] Albus, J. S. (1993). A reference model architecture for intelligent systems design. *An introduction to intelligent and autonomous control*, pages 27–56.

[Albus et al., 1981] Albus, J. S., Barbera, T., and Nagel, R. (1981). Theory and Practice of Hierarchical Control.

[Albus and Rippey, 1994] Albus, J. S. and Rippey, W. G. (1994). RCS: A reference model architecture for intelligent control. In *Proceedings - From Perception to Action Conference, PerAc 1994*, pages 218–229. Institute of Electrical and Electronics Engineers Inc.

[Amir and Maynard-Zhang, 2004] Amir, E. and Maynard-Zhang, P. (2004). Logic-based subsumption architecture. *Artificial Intelligence*, 153(1-2):167–237.

[Anderson, 2005] Anderson, J. R. (2005). Human symbol manipulation within an integrated cognitive architecture. *Cognitive science*, 29(3):313–41.

[Anderson et al., 2005] Anderson, J. R., Albert, M. V., and Fincham, J. M. (2005). Tracing problem solving in real time: fMRI analysis of the subject-paced Tower of Hanoi. *Journal of cognitive neuroscience*, 17(8):1261–74.

[Arimoto, 1995] Arimoto, S. (1995). Fundamental problems of robot control: Part i, innovations in the realm of robot servo-loops. *Robotica*, 13(1):19–27.

[Ashby and Gott, 1988] Ashby, F. G. and Gott, R. E. (1988). Decision Rules in the Perception and Categorization of Multidimensional Stimuli. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(1):33–53.

[Ashby, 1956] Ashby, R. (1956). *An Introduction to Cybernetics*.

[Ashiquzzaman et al., 2017] Ashiquzzaman, A., Tushar, A. K., Islam, M. R., Shon, D., Im, K., Park, J. H., Lim, D. S., and Kim, J. (2017). Reduction of overfitting in diabetes prediction using deep learning neural network. In *Lecture Notes in Electrical Engineering*, volume 449, pages 35–43. Springer Verlag.

[Baars, 1988] Baars, B. J. (1988). A Cognitive Theory of Consciousness.

[Baars, 1997] Baars, B. J. (1997). *In the theater of consciousness: The workspace of the mind*. Oxford University Press.

[Baars, 2002] Baars, B. J. (2002). The conscious access hypothesis: origins and recent evidence. *Trends in Cognitive Sciences*, 6(1):47–52.

[Baars and Franklin, 2009] Baars, B. J. and Franklin, S. (2009). Consciousness is Computational: The LIDA Model of Global Workspace Theory. *International Journal of Machine Consciousness*, 01(01):23–32.

[Barto and Mahadevan, 2003] Barto, A. G. and Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

[Basu et al., 2019] Basu, C., Biyik, E., He, Z., Singhal, M., and Sadigh, D. (2019). Active Learning of Reward Dynamics from Hierarchical Queries. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 120–127. IEEE.

[Bianchi et al., 2011] Bianchi, D., Rolando, L., Serrao, L., Onori, S., Rizzoni, G., Khayat, N. A., Hsieh, T. M., and Kang, P. (2011). Layered control strategies for hybrid electric vehicles based on optimal control. *International Journal of Electric and Hybrid Vehicles*, 3(2):191.

[Birnbaum and Collins, 1991] Birnbaum, L. and Collins, G. C. (1991). *Proceedings of the Eighth International Workshop on Machine Learning*. Elsevier Science.

[Bohn and Atherton, 1995] Bohn, C. and Atherton, D. (1995). An analysis package comparing PID anti-windup strategies. *IEEE Control Systems Magazine*, 15(2):34–40.

[Bouabdallah et al., 2004] Bouabdallah, S., Noth, A., and Siegwart, R. (2004). PID vs LQ control techniques applied to an indoor micro Quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2451–2456.

[Bourbon, 1996] Bourbon, W. T. (1996). On the accuracy and reliability of predictions by perceptual control theory: Five years later. *The Psychological Record*, 46(1):39.

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

[Brooks et al., 2015] Brooks, R., Angle, C., and Greiner, H. (2015). Details of sales of Roomba Robot published by iRobot. http://www.irobot.com/About-iRobot/Company-Information/History.aspx.

[Brown et al., 1997] Brown, L. L., Schneider, J. S., and Lidsky, T. I. (1997). Sensory and cognitive functions of the basal ganglia. *Current opinion in neurobiology*, 7(2):157–63.

[Carey, 2006] Carey, T. A. (2006). *The method of levels: How to do psychotherapy without getting in the way*. Living Control Systems Publ.

[Cervantes and Alvarez-Ramirez, 2001] Cervantes, I. and Alvarez-Ramirez, J. (2001). On the PID tracking control of robot manipulators. *Systems and Control Letters*, 42(1):37–46.

[Chatzilygeroudis et al., 2018] Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., and Mouret, J.-B. (2018). A survey on policy search algorithms for learning robot controllers in a handful of trials.

[Choueiry and Walsh, 2000] Choueiry, B. Y. and Walsh, T., editors (2000). *Abstraction, Reformulation, and Approximation*, volume 1864 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Cireşan et al., 2012] Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column Deep Neural Networks for Image Classification. page 20.

[Cisek, 2007] Cisek, P. (2007). Cortical mechanisms of action selection: the affordance competition hypothesis. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1485):1585–99.

[Cocosco, 1998] Cocosco, C. A. (1998). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving by RE Fikes, NJ Nillson, 1971.

[Dagher et al., 2001] Dagher, A., Owen, A. M., Boecker, H., and Brooks, D. J. (2001). The role of the striatum and hippocampus in planning: a PET activation study in Parkinson's disease. *Brain : a journal of neurology*, 124(Pt 5):1020–32.

[Dash et al., 2015] Dash, P., Saikia, L. C., and Sinha, N. (2015). Automatic generation control of multi area thermal system using Bat algorithm optimized PD–PID cascade controller. *International Journal of Electrical Power & Energy Systems*, 68:364–372.

[Dean et al., 2009] Dean, P., Porrill, J., Ekerot, C.-F., and Jörntell, H. (2009). The cerebellar microcircuit as an adaptive filter: experimental and computational evidence.

[Deng et al., 2019] Deng, H., Li, Q., Cui, Y., Zhu, Y., and Chen, W. (2019). Nonlinear controller design based on cascade adaptive sliding mode control for PEM fuel cell air supply systems. *International Journal of Hydrogen Energy*, 44(35):19357–19369.

[Dennis et al., 2016] Dennis, L., Aitken, J., Collenette, J., Cucco, E., Kamali, M., McAree, O., Shaukat, A., Atkinson, K., Gao, Y., Veres, S., and Fisher, M. (2016). Agent-based Autonomous Systems and Abstraction Engines: Theory meets Practice.

[Dennis et al., 2014] Dennis, L. A., Fisher, M., Aitken, J. M., Veres, S. M., Gao, Y., Shaukat, A., and Burroughes, G. (2014). Reconfigurable Autonomy. *KI - Kunstliche Intelligenz*, 28(3):199–207.

[Descartes and Cottingham, 2013] Descartes, R. and Cottingham, J. (2013). *Ren{é} Descartes: Meditations on First Philosophy: With Selections from the Objections and Replies*. Cambridge University Press.

[Dietterich, 1998] Dietterich, T. G. (1998). The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*, pages 118–126. Citeseer.

[Digney, 1996] Digney, B. (1996). Learning and Shaping in Emergent Hierarchical Control Systems.

[Digney and Gupta, 1993] Digney, B. and Gupta, M. (1993). A distributed adaptive control system for a quadruped mobile robot. In *IEEE International Conference on Neural Networks*, pages 144–149. IEEE.

[Digney, 1995] Digney, B. L. (1995). A Distributed Adaptive Control Architecture for Autonomous Agents.

[Digney, 1998] Digney, B. L. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In *From Animals to Animats 5*, pages 321–330. MIT Press.

[Dinsmore, 2014] Dinsmore, J. (2014). *The symbolic and connectionist paradigms: closing the gap*. Psychology Press.

[DiStefano et al., 1967] DiStefano, J. J., Stubberud, A. R., and Williams, I. J. (1967). *Theory and Problems of Feedback and Control Systems*. McGraw-Hill.

[Dominey et al., 1995] Dominey, P., Arbib, M., and Joseph, J. P. (1995). A model of corticostriatal plasticity for learning oculomotor associations and sequences. *Journal of cognitive neuroscience*, 7(3):311–36.

[Dong and Chen, 2006] Dong, D. and Chen, C. (2006). Reinforcement Strategy Using Quantum Amplitude Amplification for Robot Learning. In *2007 Chinese Control Conference*, pages 571–575. IEEE.

[Dong et al., 2012] Dong, D., Chen, C., Chu, J., and Tarn, T.-J. (2012). Robust Quantum-Inspired Reinforcement Learning for Robot Navigation. *IEEE/ASME Transactions on Mechatronics*, 17(1):86–97.

[Doyle and Csete, 2011] Doyle, J. C. and Csete, M. (2011). Architecture, constraints, and behavior. *Proceedings of the National Academy of Sciences of the United States of America*, 108 Suppl(Supplement 3):15624–30.

[Duke et al., 1995] Duke, E. A., Hicken, W. F., Nicoll, W. S. M., Robinson, D. B., and Strachan, J. C. G. (1995). Platonis opera, vol. i.

[Findeisen, 1984] Findeisen, W. (1984). The essentials of hierarchical control. pages 38–61.

[Fister et al., 2013] Fister, I., Fister, D., and Yang, X.-S. (2013). A hybrid bat algorithm.

[Fister et al., 2014] Fister, I., Fong, S., and Brest, J. (2014). A Novel Hybrid Self-Adaptive Bat Algorithm.

[Forssell, 2008] Forssell, D. (2008). *Management and leadership: Insight for effective practice*. Living Control Systems Publ.

[Franklin et al., 2014] Franklin, S., Madl, T., D'Mello, S., and Snaider, J. (2014). LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development*, 6(1):19–41.

[Franklin and Patterson Jr, 2006] Franklin, S. and Patterson Jr, F. G. (2006). The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *pat*, 703:764–1004.

[Franklin et al., 2012] Franklin, S., Strain, S., Snaider, J., McCall, R., and Faghihi, U. (2012). Global Workspace Theory, its LIDA model and the underlying neuroscience. *Biologically Inspired Cognitive Architectures*, 1:32–43.

[Gao, 2016] Gao, Z. (2016). Excitatory Cerebellar Nucleocortical Circuit Provides Internal Amplification during Associative Conditioning. *Neuron*, 89(3):645–657.

[Girard et al., 2003] Girard, B., Cuzin, V., Guillot, A., Gurney, K. N., and Prescott, T. J. (2003). A basal ganglia inspired model of action selection evaluated in a robotic survival task. *Journal of integrative neuroscience*, 2(2):179–200.

[Goel et al., 2000] Goel, V., Buchel, C., Frith, C., and Dolan, R. J. (2000). Dissociation of mechanisms underlying syllogistic reasoning. *NeuroImage*, 12(5):504–14.

[Graves et al., 2009] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–68.

[Grossberg, 1969] Grossberg, S. (1969). Learning and energy-entropy dependence in some nonlinear functional-differential systems. *Bulletin of the American Mathematical Society*, 75(6):1238–1242.

[Grush, 2004] Grush, R. (2004). The emulation theory of representation: Motor control, imagery, and perception. *Behavioral and Brain Sciences*, 27(03):377–396.

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3389–3396. Institute of Electrical and Electronics Engineers Inc.

[Guo et al., 2015] Guo, J., Gao, Y., and Cui, G. (2015). The path planning for mobile robot based on bat algorithm. *International Journal of Automation and Control*, 9(1):50–60.

[Gurney et al., 2001] Gurney, K., Prescott, T. J., and Redgrave, P. (2001). A computational model of action selection in the basal ganglia. II. Analysis and simulation of behaviour. *Biological Cybernetics*, 84(6):411–423.

[Hawker and K Moore, 2020] Hawker, B. and K Moore, R. (2020). A Structural Approach to Dealing with High Dimensionality Parameter Search Spaces. *Annual Conference Towards Autonomous Robotic Systems*.

[Hawker and Moore, 2020] Hawker, B. and Moore, R. (2020). Robots Producing Their Own Hierarchies with DOSA; The Dependency-Oriented Structure Architect. pages 66–68. UK-Robotics and Autonomous Systems (RAS) Network.

[Haynes, 2014] Haynes, W. (2014). When anatomy drives physiology : expanding the actor-critic model of the basal ganglia to new subthalamus connections.

[Hebb, 1949] Hebb, D. (1949). *D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949*.

[Heess et al., 2015] Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning Continuous Control Policies by Stochastic Value Gradients.

[Helton and Davis, 2002] Helton, J. C. and Davis, F. J. (2002). Illustration of Sampling-Based Methods for Uncertainty and Sensitivity Analysis. *Risk Analysis*, 22(3):591–622.

[Hernández-Guzmán et al., 2008] Hernández-Guzmán, V. M., Santibáñez, V., and Silva-Ortigoza, R. (2008). A new tuning procedure for PID control of rigid robots. *Advanced Robotics*, 22(9):1007–1023.

[Herreros and Verschure, 2013] Herreros, I. and Verschure, P. F. (2013). Nucleo-olivary inhibition balances the interaction between the reactive and adaptive layers in motor control. *Neural Networks*, 47:64–71.

[Hicks and Others, 1907] Hicks, R. D. and Others (1907). *Aristotle De Anima*. University Press.

[Higginson et al., 2011] Higginson, S., Mansell, W., and Wood, A. M. (2011). An integrative mechanistic account of psychological distress, therapeutic change and recovery: the Perceptual Control Theory approach. *Clinical psychology review*, 31(2):249–59.

[Hofe and Moore, 2008] Hofe, R. and Moore, R. (2008). Towards an investigation of speech energetics using 'AnTon': an animatronic model of a human tongue and vocal tract.

[Horgan and Tienson, 1991] Horgan, T. and Tienson, J., editors (1991). *Connectionism and the Philosophy of Mind*, volume 9 of *Studies in Cognitive Systems*. Springer Netherlands, Dordrecht.

[Houk et al., 2007] Houk, J. C., Bastianen, C., Fansler, D., Fishbach, A., Fraser, D., Reber, P. J., Roy, S. A., and Simo, L. S. (2007). Action selection and refinement in subcortical loops through basal ganglia and cerebellum. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1485):1573–83.

[Humphries et al., 2007] Humphries, M. D., Gurney, K., and Prescott, T. J. (2007). Is there a brainstem substrate for action selection? *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1485):1627–39.

[Huynh, 2017] Huynh, D. (2017). Applying Dropout to Prevent Shallow Neural Networks from Overtraining. Technical report.

[Janabi-Sharifi and Hassanzadeh, 2011] Janabi-Sharifi, F. and Hassanzadeh, I. (2011). Experimental analysis of mobile-robot teleoperation via shared impedance control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(2):591–606.

[Joel et al., 2002] Joel, D., Niv, Y., and Ruppin, E. (2002). Actor–critic models of the basal ganglia: new anatomical and computational perspectives. *Neural Networks*, 15(4-6):535–547.

[Johnson et al., 2020] Johnson, T., Zhou, S., Cheah, W., Mansell, W., Young, R., and Watson, S. (2020). Implementation of a Perceptual Controller for an Inverted Pendulum Robot. *Journal of Intelligent & Robotic Systems*, pages 1–10.

[Jonsson, 2006] Jonsson, A. (2006). A Causal Approach to Hierarchical Decomposition in Reinforcement Learning.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, pages 237–285.

[Kennaway, 2004] Kennaway, J. R. (2004). A simple and robust hierarchical control system for a walking robot. In *University of Bath*.

[Kennaway and Date, 1999] Kennaway, R. and Date, L. R. (1999). Control of a multi-legged robot based on hierarchical PCT http://www2.cmp.uea.ac.uk/˜jrk/PCT/jp9913p.pdf.

[Khamassi, 2005] Khamassi, M. (2005). Actor-Critic Models of Reinforcement Learning in the Basal Ganglia: From Natural to Artificial Rats. *Adaptive Behavior*, 13(2):131–148.

[Knowlton et al., 1996] Knowlton, B. J., Mangels, J. A., and Squire, L. R. (1996). A neostriatal habit learning system in humans. *Science (New York, N.Y.)*, 273(5280):1399–402.

[Kohonen, 1972] Kohonen, T. (1972). Correlation Matrix Memories. *IEEE Transactions on Computers*, C-21(4):353–359.

[KONDO et al., 1999] KONDO, T., ISHIGURO, A., UCHIKAWA, Y., and EGGENBERGER, P. (1999). Realization of Robust Controllers in Evolutionary Robotics. *Transactions of the Society of Instrument and Control Engineers*, 35(11):1407–1414.

[Lee-Hand and Knott, 2015] Lee-Hand, J. and Knott, A. (2015). A neural network model of causative actions. *Frontiers in neurorobotics*, 9:4.

[Lenz et al., 2015] Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724.

[Lever et al., 2016] Lever, J., Krzywinski, M., and Altman, N. (2016). Points of Significance: Model selection and overfitting. Technical report.

[Levine et al., 2015] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2015). End-to-End Training of Deep Visuomotor Policies.

[Levine et al., 2018] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 37(4-5):421–436.

[Lippett, 2005] Lippett, A. (2005). http://vizlog.com/robot/ Hexapod project by Lippett (2005).

[Lizotte et al., 2007] Lizotte, D., Wang, T., Bowling, M., and Schuurmans, D. (2007). Automatic Gait Optimization with Gaussian Process Regression - Proceedings of the 20th international joint conference on Artifical intelligence. Technical report.

[Maffei et al., 2017] Maffei, G., Herreros, I., Sanchez-Fibla, M., Friston, K. J., and Verschure, P. F. (2017). The Perceptual Shaping of Anticipatory Actions. *doi.org*, page 184333.

[Mansell, 2020] Mansell, W. (2020). *The interdisciplinary handbook of perceptual control theory: Living control systems IV*. Academic Press.

[Marken, 1990] Marken, R. S. (1990). Spreadsheet analysis of a hierarchical control system model of behavior. *Behavior Research Methods, Instruments, & Computers*, 22(4):349–359.

[Marken, 2006] Marken, R. S. (2006). A Quarter Century of Research on Perceptual Control Theory: Opening the Door to Closed-Loop Psychology.

[Marken and Mansell, 2013] Marken, R. S. and Mansell, W. (2013). Perceptual control as a unifying concept in psychology.

[Marken and Powers, 1989] Marken, R. S. and Powers, W. T. (1989). Random-Walk Chemotaxis: Trial and Error as a Control Process. *Behavioral Neuroscience*, 103(6):1348–1355.

[Martinez-Cantin et al., 2007] Martinez-Cantin, R., de Freitas, N., Doucet, A., and Castellanos, J. (2007). Active Policy Learning for Robot Planning and Exploration under Uncertainty. In *Robotics: Science and Systems III*. Robotics: Science and Systems Foundation.

[Matic, 2014] Matic, A. (2014). PCT robot arm with visual servoing and pressure control https://www.youtube.com/watch?v=wQ6FGeSjN9c.

[Matsubara et al., 2011] Matsubara, T., Hyon, S. H., and Morimoto, J. (2011). Learning parametric dynamic movement primitives from multiple demonstrations. *Neural Networks*, 24(5):493–500.

[McClelland and Worthington, 2010] McClelland, K. and Worthington, G. (2010). http://pctweb.org/McClellandWorthington.pdf.

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

[McPhail, 1994] McPhail, C. (1994). THE DARK SIDE OF PURPOSE:. Individual and Collective Violence in Riots. *The Sociological Quarterly*, 35(1):1–32.

[Melo et al., 2008] Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 664–671, New York, New York, USA. ACM Press.

[Metta et al., 2008] Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The iCub humanoid robot. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems - PerMIS '08*, page 50, New York, New York, USA. ACM Press.

[Minorsky., 1922] Minorsky., N. (1922). Directional Stability of Automatically Steered Bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309.

[Mitchinson and Prescott, 2016] Mitchinson, B. and Prescott, T. J. (2016). MIRO: A Robot "Mammal" with a Biomimetic Brain-Based Control System. pages 179–191. Springer, Cham.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Moore, 2007] Moore, R. (2007). PRESENCE: A Human-Inspired Architecture for Speech-Based Human-Machine Interaction. *IEEE Transactions on Computers*, 56(9):1176–1188.

[Moore, 2020] Moore, R. K. (2020). PCT and beyond: toward a computational framework for 'intelligent'communicative systems. In *The Interdisciplinary Handbook of Perceptual Control Theory*, pages 557–582. Elsevier.

[Moore and Nicolao, 2017] Moore, R. K. and Nicolao, M. (2017). Toward a needs-based architecture for 'intelligent'communicative agents: Speaking with intention. *Frontiers in Robotics and AI*, 4:66.

[Morimoto and Doya, 1998] Morimoto, J. and Doya, K. (1998). Hierarchical Reinforcement Learning of Low-Dimensional Subgoals and High-Dimensional Trajectories. In *ICONIP*, pages 850–853. Citeseer.

[Morimoto and Doya, 2001] Morimoto, J. and Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51.

[Moulin-Frier et al., 2018] Moulin-Frier, C., Fischer, T., Petit, M., Pointeau, G., Puigbo, J. Y., Pattacini, U., Low, S. C., Camilleri, D., Nguyen, P., Hoffmann, M., Chang, H. J., Zambelli, M., Mealier, A. L., Damianou, A., Metta, G., Prescott, T. J., Demiris, Y., Dominey, P. F., and Verschure, P. F. (2018). DAC-h3: A Proactive Robot Cognitive Architecture to Acquire and Express Knowledge about the World and the Self. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):1005–1022.

[Mynski and Papert, 1969] Mynski, M. L. and Papert, S. A. (1969). Perceptrons: An Introduction to Computational Geometry. *MA: MIT Press, Cambridge*.

[Nagata and Watanabe, 2011] Nagata, F. and Watanabe, K. (2011). Adaptive learning with large variability of teaching signals for neural networks and its application to motion control of an industrial robot. *International Journal of Automation and Computing*, 8(1):54–61.

[Nguyen et al., 2014] Nguyen, A., Yosinski, J., and Clune, J. (2014). Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images.

[Niv et al., 2012] Niv, Y., Edlund, J. A., Dayan, P., and O'Doherty, J. P. (2012). Neural prediction errors reveal a risk-sensitive reinforcement-learning process in the human brain. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 32(2):551–62.

[O'Reilly and Frank, 2006] O'Reilly, R. C. and Frank, M. J. (2006). Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, 18(2):283–328.

[Packard and Knowlton, 2002] Packard, M. G. and Knowlton, B. J. (2002). Learning and memory functions of the Basal Ganglia. *Annual review of neuroscience*, 25:563–93.

[Pairet et al., 2019] Pairet, È., Ardón, P., Broz, F., Mistry, M., and Petillot, Y. (2019). Learning and Generalisation of Primitives Skills Towards Robust Dual-arm Manipulation.

[Parr and Russell, 1998] Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049.

[Parr, 1998] Parr, R. E. (1998). *Hierarchical control and learning for Markov decision processes*. PhD thesis, Citeseer.

[Pastor et al., 2011] Pastor, P., Righetti, L., Kalakrishnan, M., and Schaal, S. (2011). Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE.

[Perkins and Precup, 1999] Perkins, T. J. and Precup, D. (1999). Using Options for Knowledge Transfer in Reinforcement Learning TITLE2:.

[Porrill et al., 2004] Porrill, J., Dean, P., and Stone, J. V. (2004). Recurrent cerebellar architecture solves the motor-error problem. *Proceedings. Biological sciences*, 271(1541):789–96.

[Powers, 1974] Powers, W. T. (1974). *Behavior: The Control of Perception*.

[Powers, 1978] Powers, W. T. (1978). Quantitative analysis of purposive systems: Some spadework at the foundations of scientific psychology.

[Powers, 1999] Powers, W. T. (1999). An experiment with a simple method for controlling an inverted pendulum. *Journal on perceptual control theory*, 1(1):3–12.

[Powers, 2008] Powers, W. T. (2008). *Living control systems III: The fact of control*. Benchmark Publications New Canaan, CT, USA.

[Powers et al., 2011] Powers, W. T., Abbott, B., Carey, T. A., Goldstein, D. M., Mansell, W., Marken, R. S., and Taylor, M. (2011). Perceptual Control Theory-A Model for Understanding the Mechanisms and Phenomena of Control. *Perceptual Control Theory: Science & Applications: A Book of Readings*, pages 18–34.

[Prat et al., 2007] Prat, C. S., Keller, T. A., and Just, M. A. (2007). Individual differences in sentence comprehension: a functional magnetic resonance imaging investigation of syntactic and lexical processing demands. *Journal of cognitive neuroscience*, 19(12):1950–63.

[Precup et al., 2005] Precup, D., Paduraru, C., Koop, A., Sutton, R. S., and Singh, S. P. (2005). Off-policy learning with options and recognizers. In *Advances in Neural Information Processing Systems*, pages 1097–1104.

[Precup et al., 1998a] Precup, D., Sutton, R., and Singh, S. (1998a). *Theoretical results on reinforcement learning with temporally abstract options in "Machine Learning: ECML-98"*, volume 1398 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Precup et al., 1998b] Precup, D., Sutton, R. S., and Singh, S. (1998b). Multi-time models for temporally abstract planning. *Advances in neural information processing systems*, pages 1050–1056.

[Prescott et al., 1999] Prescott, T. J., Redgrave, P., and Gurney, K. (1999). Layered Control Architectures in Robots and Vertebrates. *Adaptive Behavior*, 7(1):99–127.

[Rabitz et al., 1999] Rabitz, H., Aliş, Ö. F., Shorter, J., and Shim, K. (1999). Efficient input-output model representations. *Computer Physics Communications*, 117(1):11–20.

[Rahmani et al., 2016] Rahmani, M., Ghanbari, A., and Ettefagh, M. M. (2016). Robust adaptive control of a bio-inspired robot manipulator using bat algorithm. *Expert Systems with Applications*, 56:164–176.

[Ramamurthy et al., 2006] Ramamurthy, U., Baars, B. J., S. K., D., and Franklin, S. (2006). LIDA: A Working Model of Cognition.

[Raspopovic et al., 2014] Raspopovic, S., Capogrosso, M., Petrini, F. M., Bonizzato, M., Rigosa, J., Di Pino, G., Carpaneto, J., Controzzi, M., Boretius, T., Fernandez, E., Granata, G., Oddo, C. M., Citi, L., Ciancio, A. L., Cipriani, C., Carrozza, M. C., Jensen, W., Guglielmelli, E., Stieglitz, T., Rossini, P. M., and Micera, S. (2014). Restoring natural sensory feedback in real-time bidirectional hand prostheses. *Science translational medicine*, 6(222):222ra19.

[Ravizza and Ivry, 2001] Ravizza, S. M. and Ivry, R. B. (2001). Comparison of the basal ganglia and cerebellum in shifting attention. *Journal of cognitive neuroscience*, 13(3):285–97.

[Redgrave et al., 1999] Redgrave, P., Prescott, T., and Gurney, K. (1999). The basal ganglia: a vertebrate solution to the selection problem? *Neuroscience*, 89(4):1009–1023.

[Renno-Costa et al., 2011] Renno-Costa, C., Luvizotto, A. L., Marcos, E., Duff, A., Sanchez-Fibla, M., and Verschure, P. F. M. J. (2011). Integrating neuroscience-based models towards an autonomous biomimetic Synthetic Forager. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 210–215. IEEE.

[Reyes-Ortiz et al., 2020] Reyes-Ortiz, O. J., Useche-Castelblanco, J. S., and Vargas-Fonseca, G. L. (2020). Implementation of Fuzzy PID Controller in Cascade with Anti-Windup to Real-Scale Test Equipment for Pavements. *Engineering Transactions*, 0(0).

[Robinson, 2009] Robinson, R. (2009). Exploring the "global workspace" of consciousness. *PLoS biology*, 7(3):e1000066.

[Rodrigues et al., 2009] Rodrigues, M. A., Li, Y. E., Lee, M. H., and Rowland, J. J. (2009). Robotic grasping of complex shapes: is full geometrical knowledge of the shape really necessary? *Robotica*, 13(05):499.

[Rosenblatt, 1957] Rosenblatt, F. (1957). The perceptron: A probabilistic model for information storage and organization in the brain.

[Rosenblatt and Payton, 1989] Rosenblatt, J. and Payton, D. (1989). A fine-grained alternative to the subsumption architecture for mobile robot control. In *International Joint Conference on Neural Networks*, pages 317–323 vol.2. IEEE.

[Rössert et al., 2015] Rössert, C., Dean, P., and Porrill, J. (2015). At the Edge of Chaos: How Cerebellar Granular Layer Network Dynamics Can Provide the Basis for Temporal Filters. *PLOS Computational Biology*, 11(10):e1004515.

[Roy et al., 2014] Roy, P., Ray, R., Wang, C., and Wong, W. F. (2014). ASAC: Automatic sensitivity analysis for approximate computing. In *ACM SIGPLAN Notices*, volume 49, pages 95–104, New York, New York, USA. Association for Computing Machinery.

[Sahrin et al., 2018] Sahrin, A., Purnomo, M. H., and Utomo, G. R. (2018). Cascade Control With PID-PSO Method on The Stabilizer Unit. In *The 2nd International Conference on Applied Electromagnetic Technology (AEMT) 2018*.

[Santos et al., 2018] Santos, M. S., Soares, J. P., Abreu, P. H., Araujo, H., and Santos, J. (2018). Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches [Research Frontier]. *IEEE Computational Intelligence Magazine*, 13(4):59–76.

[Schmidhuber, 2014] Schmidhuber, J. (2014). Deep learning in neural networks: An overview. *Neural Networks*, page 88.

[Schroeder and Korel, 2000] Schroeder, P. J. and Korel, B. (2000). Black-box test reduction using input-output analysis. In *Proceedings of the ACM SIGSOFT 2000 International Symposium on Software Testing and Analysis*, volume 25, pages 173–177.

[Searle, 1992] Searle, J. R. (1992). *The rediscovery of the mind*. MIT press.

[Selfridge, 1958] Selfridge, O. (1958). Pandemonium: a paradigm for learning in Mechanisation of Thought Processes. pages 513 – 526.

[Shabani et al., 2013] Shabani, H., Vahidi, B., and Ebrahimpour, M. (2013). A robust PID controller based on imperialist competitive algorithm for load-frequency control of power systems. *ISA Transactions*, 52(1):88–95.

[Shanahan, 2000] Shanahan, M. (2000). Reinventing Shakey. In *Logic-Based Artificial Intelligence*, pages 233–253. Springer US.

[Shanahan, 2010] Shanahan, M. (2010). *Embodiment and the inner life: Cognition and Consciousness in the Space of Possible Minds*. Oxford University Press.

[Shanahan and Shanahan, 1998] Shanahan, M. and Shanahan, M. (1998). A Logical Account of the Common Sense Informatic Situation for a Mobile Robot.

[Shanahan and Witkowski, 2001] Shanahan, M. and Witkowski, M. (2001). High-level robot control through logic. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1986 LNAI, pages 104–121. Springer Verlag.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[Snaider et al., 2011] Snaider, J., McCall, R., and Franklin, S. (2011). The LIDA Framework as a General Tool for AGI. 6830.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Technical report.

[Stocco and Anderson, 2008] Stocco, A. and Anderson, J. R. (2008). Endogenous control and task representation: an fMRI study in algebraic problem-solving. *Journal of cognitive neuroscience*, 20(7):1300–14.

[Stocco et al., 2010] Stocco, A., Lebiere, C., and Anderson, J. R. (2010). Conditional routing of information to the cortex: a model of the basal ganglia's role in cognitive coordination. *Psychological review*, 117(2):541–74.

[Stone and Sutton, 2001] Stone, P. and Sutton, R. S. (2001). Scaling Reinforcement Learning toward RoboCup Soccer. pages 537–544.

[Sun and Er, 2004] Sun, Y. L. and Er, M. J. (2004). Hybrid fuzzy control of robotics systems. *IEEE Transactions on Fuzzy Systems*, 12(6):755–765.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1.

[Takahashi and Asada, 1999] Takahashi, Y. and Asada, M. (1999). Behavior acquisition by multi-layered reinforcement learning. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, volume 6, pages 716–721. IEEE.

[Takahashi and Asada, 2000] Takahashi, Y. and Asada, M. (2000). Vision-guided behavior acquisition of a mobile robot by multi-layered reinforcement learning. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 395–402. IEEE.

[Takahashi et al., 2008] Takahashi, Y., Schoenbaum, G., and Niv, Y. (2008). Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in neuroscience*, 2(1):86–99.

[Taylor and Birmingham, 1948] Taylor, F. V. and Birmingham, H. P. (1948). Studies of tracking behavior. II. The acceleration pattern of quick manual corrective responses.

[Teicher et al., 2000] Teicher, M. H., Anderson, C. M., Polcari, A., Glod, C. A., Maas, L. C., and Renshaw, P. F. (2000). Functional deficits in basal ganglia of children with attention-deficit/hyperactivity disorder shown with functional magnetic resonance imaging relaxometry. *Nature medicine*, 6(4):470–3.

[Teichmann et al., 2006] Teichmann, M., Dupoux, E., Kouider, S., and Bachoud-Lévi, A.-C. (2006). The role of the striatum in processing language rules: evidence from word perception in Huntington's disease. *Journal of cognitive neuroscience*, 18(9):1555–69.

[Tham, 1995] Tham, C. K. (1995). Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 15(4):247–274.

[Togelius, 2004] Togelius, J. (2004). Evolution of a Subsumption Architecture Neurocontroller.

[Trunk, 1979] Trunk, G. V. (1979). A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):306–307.

[Uexküll, 1934] Uexküll, J. V. (1934). A Stroll Through the World of Animals and Men: A Picture Book of Invisible Worlds. In *Instinctive Behaviour*.

[Ullman et al., 1997] Ullman, M. T., Corkin, S., Coppola, M., Hickok, G., Growdon, J. H., Koroshetz, W. J., and Pinker, S. (1997). A Neural Dissociation within Language: Evidence that the Mental Dictionary Is Part of Declarative Memory, and that Grammatical Rules Are Processed by the Procedural System. *Journal of cognitive neuroscience*, 9(2):266–76.

[Uther, 2002] Uther, W. T. B. (2002). Tree Based Hierarchical Reinforcement Learning.

[Vancouver and Putka, 2000] Vancouver, J. and Putka, D. (2000). Analyzing Goal-Striving Processes and a Test of the Generalizability of Perceptual Control Theory. *Organizational behavior and human decision processes*, 82(2):334–362.

[Verschure, 2012] Verschure, P. F. (2012). Distributed Adaptive Control: A theory of the Mind, Brain, Body Nexus. *Biologically Inspired Cognitive Architectures*, 1:55–72.

[Verschure et al., 2003] Verschure, P. F. M. J., Voegtlin, T., and Douglas, R. J. (2003). Environmentally mediated synergy between perception and behaviour in mobile robots. *Nature*, 425(6958):620–4.

[Vignali et al., 2014] Vignali, R., Deori, L., and Prandini, M. (2014). Control input design: Detecting non influential inputs while satisfying a reachability specification. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, volume 19, pages 1416–1421. IFAC Secretariat.

[Wahid et al., 2019] Wahid, A., Toshev, A., Fiser, M., and Lee, T.-W. E. (2019). Long Range Neural Navigation Policies for the Real World.

[Wang, 1997] Wang, Q. J. (1997). Using genetic algorithms to optimise model parameters. *Environmental Modelling and Software*, 12(1):27–34.

[Wiener, 1948] Wiener, N. (1948). *Cybernetics or control and communication in the animal and the machine*. Technology Press.

[Wiering and Schmidhuber, 1997] Wiering, M. and Schmidhuber, J. (1997). HQ-Learning. *Adaptive Behavior*, 6(2):219–246.

[Wittgenstein, 1967] Wittgenstein, L. (1967). *Philosophische Untersuchungen. Philosophical Investigations; Translated by GEM Anscombe. Reprinted*. Blackwell.

[Yamada et al., 1998] Yamada, S., Watanabe, A., and Nakashima, M. (1998). Hybrid reinforcement learning and its application to biped robot control. pages 1071–1077.

[Yamaguchi et al., 1996] Yamaguchi, T., Masubuchi, M., Fujihara, K., and Yachida, M. (1996). Realtime reinforcement learning for a real robot in the real environment. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, volume 3, pages 1321–1328. IEEE.

[Yamazaki and Tanaka, 2007] Yamazaki, T. and Tanaka, S. (2007). The cerebellum as a liquid state machine. *Neural Networks*, 20(3):290–297.

[Yang, 2010] Yang, X. S. (2010). A new metaheuristic Bat-inspired Algorithm. In *Studies in Computational Intelligence*, volume 284, pages 65–74.

[Ying, 2006] Ying, H. (2006). Deriving analytical input-output relationship for fuzzy controllers using arbitrary input fuzzy sets and Zadeh fuzzy AND operator. *IEEE Transactions on Fuzzy Systems*, 14(5):654–662.

[Young, 2015] Young, R. (2015). Balancing Robot with Reorganisational Learning https://www.youtube.com/watch?v=QF7K6Lhx5C8.

[Young, 2020] Young, R. (2020). Robotics in the real world. In *The Interdisciplinary Handbook of Perceptual Control Theory*, pages 517–556. Elsevier.

[Zhao et al., 2019] Zhao, C., Sigaud, O., Stulp, F., and Hospedales, T. M. (2019). Investigating Generalisation in Continuous Deep Reinforcement Learning.

[Zhao and Cziko, 2001] Zhao, Y. and Cziko, G. A. (2001). Teacher Adoption of Technology: A Perceptual Control Theory Perspective. *Journal of Technology and Teacher Education*, 9(1):5–30.