WARWICK
THE UNIVERSITY OF WARWICK

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

http://wrap.warwick.ac.uk/179845

Copyright and reuse:
This thesis is made available online and is protected by original copyright.
Please scroll down to view the document itself.
Please refer to the repository record for this item for information to help you to cite it.
Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

warwick.ac.uk/lib-publications

Building Networks of Markov Decision Processes to Achieve Synchronised Behaviour from Complex Multi-Agent Systems

by

Christopher Robert Deeks

Thesis

Submitted to the University of Warwick

for the degree of Doctor of Philosophy in Mathematics and Complexity Science

Doctor of Philosophy

Centre for Complexity Science

March 2022

Contents

Figures

Acknowledgements

There are many people who have played a part in making this thesis a reality. The story starts in my old employment, with my then colleagues Bill Williams and Neil Cade, who bequeathed me the idea of using MDPs to model the decision making of autonomous system controller in the first place, and then secured the piece of work which as a new graduate research scientist first got me into all this. My then group leader Vaughan Stanger was supportive of all my attempts to continue some of more speculative ideas and pursue novel mathematical modelling for all sorts of advanced technology problems, and ultimately supporting this path into a PhD instead. Many other people from those days would have contributed indirectly, through bouncing ideas around or simply pointing out what were the better ideas or the sillier ideas. Some of you I still cross paths with, ours being a small industry; others I know have moved on or retired. But it was those early days in my career that led to this thesis, and I want your contributions to be noted.

I also owe thanks to the Centre for Complexity Science at Warwick, most especially for all the flexibility shown to accommodate someone determined to do this part time and not put his career on pause in the meantime. My primary supervisor, Robert Mackay, deserves many thanks for the helpful suggestions throughout, and indeed agreeing to supervise this PhD in the first place when I first came enquiring from outside about my interest and the myriad of topics I might like to study further. Thanks also to my second supervisor Darek Ceglarek in WMG. The time invested in the early part of this PhD familiarising myself with digital lifecycle and new thinking in automotive industrial engineering provided the focus and application area that I needed to take this PhD forward, and has influenced the end result.

Thanks are also due to various people at my current employer. Whilst not involved in the technical work, the expressions of interest and encouraging words of support have been helpful throughout. Just tolerating the fact I have this other far-from-insignificant commitment on-going in the background all these years has been a vital contribution to enabling me to get this work to where it is.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. The work presented here is my own, unless stated otherwise. No part of this work has been submitted in any previous application for any degree or professional qualification.

Parts of this thesis have been published by the author:

- ***Dynamic Control of Synchronised Automata using Networked Q-Learning Techniques***, Christopher Deeks, 2015 IMA Mathematics of Robotics Conference [full paper]
- ***Achieving Synchronisation in Swarm Robotics: Applying Networked Q-Learning to Production Line Automata***, Christopher Deeks, 2016 ANTS Conference [presentation, and extended abstract within the proceedings journal]
- ***Adapting Multi-Agent Swarm Robotics to Achieve Synchronised Behaviour from Production Line Automata***, Christopher Deeks, 2020 IMA Mathematics of Robotics Conference [full paper, included within the special edition journal Springer Proceedings in Advanced Robotics 21]

The published material is contained within Chapter 8 of this thesis. However, the presentation within contains additional content and more detailed discussion that was not included within the published material.

Parts of this thesis are based on earlier collaborative work by the author and Bill Williams:

- This material is contained within Annex C Sections C.1 and C.2, outside the main body of this thesis. The material in these sections has had minor revision by the author to fit the context of this thesis, but contains content that was originally joint work. [Note: Sections C.3-C.5 are the author's own work, see below.]
- Parts of Chapters 5.3 to 5.6 (other than 5.6.7), 6.2, 6.3 and 6.5 are the author's own work, but have previously been included in a collated technical report jointly authored

by Bill Williams (referenced herein [23] but not publicly available). Bill Williams' chapters of that earlier report have not been included herein, and the material that has been included in the sections listed has been rewritten from draft notes left over from the time of that earlier report. All sections have been revised and expanded with new material conducted as part of this PhD. A short summary paper covering a subset of the earlier material has previously been published and is referenced herein [24].

- Parts of Chapter 4, and Annex C Section C.3-C.5, are from an unpublished paper written by the author under the supervision of Bill Williams, from an earlier attempt to initiate further research into these topics. The material in Chapter 4 has been rewritten by the author (almost completely replaced in the case of Chapter 4.2), and updated and extended for this thesis. The material in Annex C Section C.3-C.5 has also had some revisions and updates in order to fit the context of this thesis.

Abstract

Complex multi-agent systems, consisting of multiple decision making agents as part of a larger collective, are particularly complex control problems, because of the potential for emergent (and potentially undesirable or even unsafe) behaviours to arise from the interaction between separate agents. When a number of autonomous systems interact with one another, be they robotic systems, intelligent software agents, or autonomous vehicles, they can become a complex multi-agent system, potentially prone to emergent behaviours.

Understanding, controlling and assuring complex multi-agent systems is a difficult challenge for the systems engineer, through all stages of the engineering lifecycle. There is often no simple mechanism to control the behaviour of the whole, and no established formal techniques for system proving. The underpinning mathematics is too far removed from the systems of years gone by for established techniques to be transferrable. This thesis takes a fresh look at this topic using the mathematical framework of complexity science and theoretical aspects of systems science to begin to tackle this knotty problem domain.

Synchronised robotics is a concept introduced for this PhD to describe systems consisting of multiple robotic appendages acting independently based on simple control logic, but unbeknownst to the individual robots also acting as part of a carefully choreographed collective system. Synchronised robotics provides an ideal application area from which to develop and explore an exemplar scenario. Production line robotics are the chosen exemplar. This thesis shows how systems science principles can be utilised to represent all kinds of complex multi-agent system, with different internal network structures between decision nodes mirroring the myriad of ways that a systems architect might choose to construct his or her system. It then proceeds to show how to generalise the Markov Decision Process (MDP) formulation to these networks, to produce models of interactive autonomy. This novel approach to systems design and proving is brought to life through application to the production line robotics exemplar, for which a mathematical model based on the processes and techniques described has been built, tested, and initial results obtained demonstrating the potential efficacy of the approach for capturing the complex behaviours displayed, providing a control mechanism with improved resilience to correcting undesirable emergent behaviours, and pointing a way towards a potential future system proving tool for multi-agent systems.

Chapter 1

1 Introduction

1.1 Motivation

This thesis has its origins in a small industrial research and development project that ran for a few short months in 2007-08 conducted primarily by the author of this thesis. That study had a short duration and a specific remit – to investigate the feasibility of using a particular class of machine learning algorithm as a mechanism for enabling an autonomous system to learn a preferred route to a specific endpoint and, more importantly, using simulation tools to try to illustrate or enumerate the span of the state space actually reached during the learning phase.

That activity finished a long time ago, concluding that it was feasible to demonstrate in a simulated environment that a virtual autonomous vehicle's breadth of movement during a learning phase could be bounded to remain within a desired region, but by weighting the key learning parameters to strongly incentivise movement within that region, in which case it was questionable whether there was any point in having the autonomous vehicle trying to 'learn' to do essentially what had to be worked out in advance. Put simply, there seemed to be little point 'learning' what could just be specified. It was the correct conclusion for the purposes of the original industrial research application, but a frustrating answer in other ways. In just those few months, the imagination of the author had been piqued. There was clearly a rich field of mathematics underpinning the whole field of autonomous decision making under uncertainty, interesting paths that could be explored in geometry and computational geometry, and once one started to think about the implications of trying to apply mathematical models of decision making to multiple autonomous vehicles operating alongside one another, and interacting with each other, it was clear there was an interesting gateway into topics such as self-organisation and emergence in complex systems.

As a young graduate research scientist I was intrigued. Unfortunately, given the frustratingly 'correct' answer to the original research question, the proposed follow-on activity was not to

be. Further research into mathematical techniques that did not have an obviously neat and provable solution that would allow the bounding of the range of achievable states was not a priority against the funding priorities of the time.

In my five years in industrial research and development, undertaking numerous projects covering a wide range of technologies and applications, this always felt like 'the one that got away', deserving of far greater attention but seemingly impossible to continue within the industrial context of the time, as priorities shifted elsewhere. I continued to dabble in the area in addition to my work, preparing research proposals and exploring the background to some of the various promising threads of research, but to no avail as funding streams became tighter and more focused on speedy exploitation. I had interesting work – but not this work, which was always tantalisingly there in the background.

My then employer was supportive of packaging some of my ideas up as a proposal for an industrially sponsored PhD, and I had the advantage that a term of the original project funding was that results had to published, and so the basic building blocks were already in the public domain. Other elements, such as the underpinning mathematics and how I thought they could be generalised to multiple interacting systems, had been beyond the scope of the original activity anyway, and had been investigated subsequently to that in my own time. There was all the material needed to embark on a PhD taking a deeper look into these topics and exploring some of these research avenues previously only looked on from afar and speculated about.

It was a long road firstly to start, and latterly to complete, this PhD thesis. Whilst it was originally hoped this study could be more formally linked to my professional interests (particularly concerning advanced software architectures and autonomous agents), the fast-moving world of industrial engineering compared to the multi-year research commitment of a PhD made this impractical. It quickly became clear it would be far easier to manage these two threads by separating them, and learning by chance of the Complexity Science DTC at the University of Warwick presented an excellent opportunity to pursue another route, whilst fortuitously also being a return to my 'home' university from my first mathematics degree. I am grateful for the flexibility shown given my unusual route to PhD, with the deferred start, and then my need for an extended period of part-time study, all to fit around a career I did not wish to put on hold. Career moves in the meantime have drawn me away from the world of full-time industrial research and development into systems engineering, and more recently into systems architecting.

The elongated period this PhD has taken has enriched this work. I recognise now, many years after starting on this PhD, what a disparate range of fields my personal, academic and professional interests have spanned. These disparate interests have intersected and contributed to this study in their different ways:

- I was a mathematics graduate first, with my skills in this area refreshed and extended within the Complexity Science DTC, and this has been supplemented with practical experience of undertaking applied mathematical research in both industrial and academic contexts.
- I had a childhood interest in planes, trains and automobiles, and well as in robots, and so in retrospect there was an inevitability about me ending up spending part of my career researching machine learning and autonomous vehicles (amongst many other technology areas).
- In more recent years, as a systems engineer and more recently as a systems architect, I now can appreciate the engineering domain in a whole different light, and naturally I have a particular focus on complex avionics systems with increasing levels of computational complexity arising from ever more sophisticated algorithms and software defined techniques for system control.
- For the purposes of this PhD I began to develop a number of application areas, but it was preferable to prioritise one that was connected to Warwick research interests, and the Warwick Manufacturing Group (WMG) introducing me to industrial engineering and digital manufacturing has been invaluable. This assistance has motivated the direction of this work and the form of the thesis ultimately produced.

In a way I would not have anticipated at the outset of this PhD, never mind at the outset of the original motivating project all those years ago, elements and themes from all of these disparate fields have percolated through to this research, which I hope has given it a richness of ideas but also a sense of authenticity that comes from the fact the author has been inspired by real world experience.

With such a rich base of technical material and experiences to choose from, one of the hardest parts of this PhD, and of compiling this thesis, has been to select from over a decade's worth of experience in different fields the elements that best form a coherent narrative and make novel contributions to their fields, maintaining an emphasis on my core interest of mathematical modelling of interacting systems throughout. A decision was made during this research to

focus on developing the mathematical models and applying them to an industrial case study. The material presented in the main body of this thesis aims to tell a coherent story from the motivating technical challenges, through the systems science perspective that then motivates the reintroduction of the mathematical framework first developed and published in my earlier work. This is then generalised, advanced, and taken further in new work and applied to the industrial exemplar, thereby demonstrating the novel approach to synchronised robotics that I had long believed could be developed out of these fields. Results from simulation and experimentation conducted and published as part of this PhD are included and analysed, and put in context as proof of concept.

Other research conducted as part of this work but which is not core to the main argument is included in the form of annexes to this thesis. This work has not been fully developed due to the need to have a primary focus to the thesis, but enough work has been done to capture some of these ideas that I would like it to be recorded somewhere for posterity.

Overall, I want this thesis to achieve its primary objective to show that enough original thinking has been conducted by myself, and some valuable results obtained which add knowledge to their fields, that merit a PhD. But I also want it to whet the reader's appetite with glimpses of how much more I could have undertaken, and to show that, even though I am now at the end of an already elongated period of study, there are always more interesting threads still to follow.

1.2 Research Scope

1.2.1 Autonomous and Learning Systems

The emergence in recent decades of increasingly advanced and capable autonomous systems, including robotic systems, is a phenomenon familiar to essentially everyone in the developed world. Early examples of robotic systems include systems like production line robots, capable of undertaking individual activities as part of a production line. Exploratory probes are another widely known example, initially for space exploration, but more recently in terrestrial settings such as oceanic observation. Other applications areas range from bomb disposal robots through to domestic aids for children.

In parallel to the emergence of robotics systems has been the emergence of software-defined systems. Software has advanced exponentially from its beginnings as a few simple lines of logic to provide an automatic control mechanism for a system to a major driver of system

complexity in its own right. Instead of merely codifying a number of procedural steps for a machine to undertake without needing to be directly under the control of a human operator, it is now quite standard for machines to have much programming logic within them undertaking ever more complex sequences of operations faster than a human operator ever could.

Inevitably, the two fields intersected. Modern techniques in software engineering unlock vastly greater opportunities for robotic systems than could have been achieved by earlier generations of systems. Combining the ability to have vehicles, or robotic systems, or in theory any kind of physical and/or mechanical piece of technology operating with advanced software controlling part or all of its operation, produces automated or autonomous systems.

Note that for the purposes of this thesis, the question of what differentiates automation from autonomy does not need to be explored further. Autonomous systems are taken hereafter to be a general reference to systems which select their actions based on their environment and circumstances in which they find themselves. The principle that there is a need to understand and influence the behaviours of these systems arises regardless of the computational complexity of the system.

Automation and autonomy in systems are becoming increasingly prominent in many parts of industry. Throughout the emergence of robotic and autonomous systems, for all the excitement and consequent investment into the field driven by the expected benefits of such technology, so there has also been scepticism, distrust or a lack of acceptance by some of the new technology. Increasing use of machinery within industry is associated with reduced need for manual labour, causing scepticism of the benefits of the technology amongst some in the labour movement. Safety concerns also abound; indeed this is the major barrier to the role out of fully autonomous vehicles, something enthusiastically projected by automotive companies on a regular basis over the past decade but yet to become a consumer reality. The problem with the introduction of autonomous systems into roles featuring ever greater interaction with humans is partly one of societal readiness and willingness, but also one of predictability and assurance of how the systems will behave and what their impact will be.

1.2.2 Control, Assurance and Complexity

This domain challenges many aspects of the engineering process; specifically, in order to certify a new system for use, it is necessary to provide evidence of the likely behaviour of this system in all circumstances under which it can realistically find itself. This is already an

expensive and time-consuming process; it will only become more so with the increasing complexity of new systems. Despite the considerable effort and expense invested in systems design and validation activities, there is always a risk of defective products, poor design features or otherwise undesirable outcomes being obtained. These may just cause inefficiencies, but in safety critical systems they could have far more serious consequences.

Complex systems, consisting of many interacting parts, present particular challenges for control and assurance, because of the risk of emergent properties or behaviours arising from the interaction of these parts. Typical systems engineering processes decompose a system into subsystems, and the requirements and any performance measures or criteria for that system into functional and performance requirements relevant to the individual subsystems. Once the subsystems have been designed, produced, tested and proven to meet the requirements and performance criteria for each subsystem, they are then ready to be integrated together into the parent system, which should then satisfy all of its functional and performance requirements. Or so goes the theory. Much of the risk within engineering projects comes from integration risk, where the integration of separate system elements simply does not work as expected, with elements incompatible with each other or conflicting in some way, or functioning but just not producing the effects or performance anticipated due to the way that the parts interact with each other. This pattern can reoccur in subsystem development and in wider system integration activities, 'above and below' the system in question in the logical hierarchy.

Applied to autonomous systems, there are engineering risks associated not just with ensuring that any control software causes appropriate and correct decisions to be made and actions undertaken by the system itself, and in its key subsystems, but also with the integration of that autonomous system into the wider environment, including with any other collaborating autonomous systems. If there is already difficulty in demonstrating that a control mechanism within one autonomous system produces acceptable behaviour, then there is greater difficulty in demonstrating that multiple such systems can interact with each other in a way that is acceptable for all.

1.2.3 The Mathematics of Decision Making

The mathematics of machine learning already exists as a research area, covering a wide range of techniques and applications. Many threads of research into particular problems have been undertaken in different application fields and with different mathematical content. For the purposes of this thesis and its particular focus on interacting autonomous systems, the field is

narrowed specifically to learning agents. A mechanical system that takes measurements of its environment, and then applies some internal logic or computation to determine what action to take next, naturally leads to a mathematical formulation of the decision process based on learning agents at key decision making nodes.

The major blocker for acceptance of learning systems in real world applications has been establishing the confidence that the machine will learn acceptable behaviours that do not prove to be inefficient or ineffective for achieving whatever objectives the system is designed for, or worse are counterproductive or even unsafe. If there is a mathematical model for the decision process, formalising states that can be recognised by a system and actions that can be undertaken, then the question is whether it is possible to quantify the span of possible paths in state space; that is, to determine limits or bounds for the subset of the state space that a system could find itself within from a given starting point.

This is typically a very difficult problem to solve analytically. The 'control' part of the problem can be addressed with techniques from machine learning and other data-driven methods, but these do not in general lead to analytical (or even tractable) solutions to the 'assurance' problem. Put simply, machine learning or emerging artificial intelligence techniques might provide a mechanism for allowing an autonomous system to make decisions, but the 'assurance' part of the problem typically requires a high degree of confidence that the resulting behaviour can be understood in advance. This drives thinking towards assurance arguments based on modelling and simulation, but understanding the mathematical underpinnings remains important.

1.2.4 Precursor Work

It must be declared that some initial work was conducted by the author prior to commencement of this PhD. This was narrowly bounded, but the context still needs to be explained.

The precursor work was primarily concerned with modelling autonomous systems using Markov Decision Processes or MDPs (considering these as multiple parallel Markov chains, with an agent that selects actions which influence which chain is gone down – a generalisation of the basic Markov chain concept). The application area was autonomous platform navigation, and the objective was to explore how to determine whether the reachable span of the physical state space could be shown to be bounded from a mathematical model of the decision process. As noted in the introduction, the conclusion was that this could be demonstrated in simulation,

but to ensure the autonomous vehicle stayed within a defined ‘acceptable’ region this had to be weighted into the learning parameters [22],[23],[24].

Several of the building blocks for this PhD thesis were put in place by this earlier activity, not least identifying that MDPs provide a natural formulation for agent-based learning, with a range of algorithmic techniques available to solve them. It was also from this work that I had the insight that the design of the control mechanism could disincentivise or even prohibit action selections that could lead the system into undesirable regions of the state space. But where in that original research context this essentially neutered the argument for any further work (the issue being that if you had to specify what to do and what not to do, there was little need for a system to learn that which was already known), what occurred to me back then was that there might be an alternative way to apply these ideas. Techniques that might not be suitable for a machine learning ‘in the field’ might still provide a novel mechanism for training systems to have a degree of adaptivity after any initial learning activity has been completed.

I also undertook some research into multi-agent systems, and their potential application to various engineering problems. The specific focus of these activities is not relevant to this thesis, however outside of work I did start to link the two topics together. Viewing interacting autonomous systems as individual nodes within a multi-agent system begged the question as to whether the sort of machine learning applications I had looked at could be applied to multiple interacting systems, and from that insight the ideas that underpin this thesis were born.

1.2.5 Key Research Questions

The key questions that this PhD has addressed are:

- Can the underpinning mathematics of autonomous decision making be generalised to provide defined mathematical models of interaction between autonomous agents, and the effects of interactions between them?
- Is it possible to apply agent-based modelling approaches to interacting autonomous systems, and show that acceptable and effective patterns of collective behaviour can be learned?
- Can these developments be applied to a practical example of interacting autonomous systems, and be shown to enable systems to learn an adaptable pattern of behaviour that increases their robustness to any emergent effects?

The practical example chosen is the optimisation of robotic production lines, which is an excellent real-world example characterised by the need to reorganise the configuration (the positioning and tasking) of systems based on evolving events.

Out of scope of this thesis is the specific question of what tools or techniques (whether analytical or computational) might exist that could be used to determine 'bounds' of autonomous behaviour. Some initial work has been conducted in this area, but to make this work manageable as a single PhD this thread of research was parked in order to focus on the key research questions defined above. What work has been conducted against these other topics is contained in Annex C to this thesis for posterity.

1.3 Thesis Structure

This thesis touches on a number of subject areas across mathematics and engineering. The subject matter represents a fusion of topics from across the various stages of the author's professional and academic career to date. To put this research in its proper perspective, these topics are best presented as layers of subject matter, building up to the novel aspects that are the author's own creation.

- Chapter 1 (this chapter) is the introduction, and describes the scope of the rest of the thesis.
- Chapter 2 introduces the concept of synchronised robotics, describes where it fits within the wider development of robotic systems and how it differs from other related concepts such as collaborative or swarm robotics, and introduces a number of application areas that will be used throughout the thesis to provide illustrative examples and give context to the theoretical discussion.
- Chapter 3 discusses a theoretical framework for the study of complex interactive systems, developed from systems science and tailored by the author for interactive and synchronised autonomy.
- Chapter 4 takes in a more in depth look at some of the particular difficulties that arise when trying to design an effective and provable control mechanism for complex multi-agent systems, and some of the limitations of other approaches and proposals in the wider literature.

- Chapter 5 provides a step-by-step development of the proposed novel approach to producing models of complex multi-agent systems, and details the mathematical approach to representing complex autonomous decision making processes, namely Markov Decision Processes (MDPs).
- Chapter 6 takes an in-depth look at the application of the proposed modelling framework and solution toolset to the problem of parallel decision making processes, and demonstrates how they apply to a range of multi-agent systems, including interactive autonomous vehicles and robotic systems.
- Chapter 7 completes the mathematical development of the models for layering decision processes within complex systems, to ensure that the full framework which has been developed is captured within this thesis.
- Chapter 8 pulls these threads together, firstly by introducing the specific exemplar from industrial automation that was selected as a case study for this thesis, then by applying the mathematical models and tools developed in the previous sections to produce a proof-of-concept of the control of synchronised autonomy via machine learning, including the use of novel methods to mitigate some unplanned and potentially undesirable emergent behaviours. This chapter also presents the results from the simulation activities that have been conducted, and further discussion of what has been demonstrated and learned.
- Chapter 9 contains overall conclusions from the research presented, and a discussion of areas where future research might lead.

This thesis also includes three annexes, which document some of the other research undertaken but which is not core to the primary argument through the main body of the thesis. These annexes either add more background to the core material, or present initial steps into other research directions and/or other application areas that have been researched but not pursued to completion given the need to focus this work for a PhD.

- Annex A provides greater detail on the simulation environment and algorithms the author implemented for this research, including pseudocode for the major functions.
- Annex B details a couple of other exemplar applications, from which it was originally intended to select one to develop further within this PhD. The automotive industrial exemplar was selected due to its clear alignment with wider University of Warwick

research activity, and these alternatives I had begun to develop did not need to be taken further. The development of these models, as far as it got, is included here, in part to have presented all the work that has been undertaken for the curious reader and so that it does not get forgotten. Additionally, and importantly, it is also to show that the potential scope for the techniques developed by this work is not limited to the one application area. There is wider relevance and potential application for this research as a general tool to add to the suite of machine learning approaches in the wider domain.

- Annex C represents a path not taken with the research, but for which some introductory work was documented before choosing the path this PhD was going to take. This path not taken was to look into techniques perhaps within the field of geometry to attempt to estimate bounds around the state space reached by an autonomous system (or several interacting systems), which might be at least another whole thesis of work by itself. Some introductory thoughts on how this aspect of the research might have progressed are presented for completeness, and hopefully to seed the imagination of me or someone else to pursue this alternative path in some future activity.

Chapter 2

2 Interactive, Collaborative and Synchronised Robotics

2.1 Context

Autonomous systems, including robotic systems, are already a common feature within many areas of technology. An advance long predicted in science fiction dating back generations, perhaps most famously by Asimov [2] and his eponymous Laws of Robotics [3] in 1942, robotic systems increasingly became part of science fact from the 1980s onwards. The most familiar examples include systems such as robotic appendages and equipment used in industrial engineering, where the introduction of such systems increased the power and speed of industrial activities as such robotic equipment could undertake activities requiring greater force and/or maintaining greater speed than a team of human labourers might have been able to manage. Another early example of autonomous robotic systems was remote exploration probes. An early driver for the development of such systems was the space race of the mid twentieth century, where the difficulty and cost of maintaining a human presence in space motivated increasing use of unmanned systems. For deep space exploration, with mission times in the years or decades, or for long-duration systems that did not need to be permanently manned in order to undertake their assignment, satellites and probes became the tool of choice. Now in the early 21st century, sending robotic explorers to the surface of Mars or to the moons of the gas giants is familiar to the general public, and the use of such technology has expanded to terrestrial applications, not least underwater oceanic exploration and survey activities.

Modern systems have moved beyond robotic systems which are programmed with relatively simple control software to undertake sequences of actions without the direct involvement of a human controller, to increasingly automated and autonomous systems able to conduct internal logic (at increasing levels of abstraction and complexity) to assist them in deciding what to do.

Autonomous vehicles are an emerging feature on the technology horizon, with unmanned vehicles a reality in many fields, not least air vehicles. However, the arrival of autonomous vehicle technology has prompted thinking about how far society is willing to go with letting machines make decisions for themselves. There are questions not just around cultural acceptance, but also more fundamental engineering challenges around systems' functional performance, efficacy and safety of operation, which are hindering the wider uptake of such technology. The most frequent example in the popular press is that of autonomous or self-driving cars, which have received extensive coverage in recent years heralding the arrival of the new technology, yet the technology's mass market roll-out continues to be deferred as, amongst other commercial challenges, there is still insufficient confidence that manufacturers can continue to provide high probability guarantees of their reliable and safe performance.

2.2 Interaction, Collaboration and Synchronisation

If the problem of providing adequate guarantees of safe operation of singular autonomous vehicles continues to be a major unresolved challenge within the engineering domain, the problems arising with multiple interacting autonomous vehicles is a more difficult challenge still. As well as each individual autonomous system or robot presenting a challenge to demonstrate how to understand, control, and prove the appropriateness of its behaviour, there are additional complications arising from the interaction of the systems with each other, increasing the potential order of magnitude of difficulty.

This is a significant frustration because interactive autonomy, and within that interactive robotics, is emerging as a rich field of new technologies and potential applications. It is not just a question of being able to deploy multiple autonomous systems or robots within the same area to conduct multiple tasks in parallel, and requiring confidence that each will operate properly when conducting its activities; if there are any interactions between these systems (whether planned or unplanned) then, in addition to these existing challenges, there is further complexity from understanding, controlling and proving their interactive behaviour.

It is important to note that there varying degrees or modes of interaction that one might envisage between autonomous systems. Some working definitions are adopted within this thesis. It should be noted that these are based on working definitions proposed by the author when initially investigating these topics, to differentiate the key concepts required. Given the lack of any equivalent definitions identified within the wider literature (synchronised robotics itself

being a new term introduced by the author), these definitions have remained in use throughout this work and are now presented within this thesis as an original contribution.

The key concepts defined for the purposes of this work are:

> **Interaction:** *Two or more systems operating in such a manner that the activity of at least one of the systems will impact on the activity of at least one of the other systems in such a way that the behaviour or performance of the other is observably impacted.*
>
> **Collaboration:** *Two or more systems interacting with each other where at least one of the interacting systems involved is purposefully making control decisions in order to facilitate this interaction.*
>
> **Synchonisation:** *Two or more systems interacting with each other where none of the interacting systems involved is purposefully making control decisions in order to facilitate this interaction.*

Essentially, interaction is the umbrella title for the full range of scenarios where multiple autonomous systems or robotics systems operate in conjunction, but requiring there to be some impact on the behaviour of at least one system as a result of the others. Simply operating in the same location but without affecting each other in any observable way would not qualify as interaction. Within this domain, collaborative autonomy describes where there is some degree of awareness of the involvement of multiple systems by at least one of these systems, in order to facilitate some form of planning or coordination. The focus of the case studies explored later in this thesis will be on synchronised autonomy, which is the situation where the systems are interacting, but with no knowledge of their interactions with other systems – these interactions are observed only by the beholder. This is advantageous in that it allows operations without explicit (and potentially complicated and onerous) pre-planning and implementation as to how systems should respond to each other, but has the significant disadvantage that there is relatively little maturity of thinking about how to control and prove such systems, especially when the intent is that they interact and produce more complex collective behaviours.

2.3 Emerging Technical Challenges

During the 2000s a huge volume of work was funded by DSTL and others into a range of topics under control of robotics and autonomy. A wide range of techniques were covered. Because of the complexity arising in problems featuring multiple systems (complexity referring to the

patterns and behaviours arising from interaction that do not arise from systems in isolation) progress was generally more limited with the multiple systems problem.

Much of the work that was conducted in this area was based on game theory, adopting multi-agent system architectures. While there is merit in agent-based architectures (they are intuitive and logical on a theoretical level), they came to be associated too closely with particular software approaches and toolsets that were not yet sufficiently mature to really deliver what the more theoretically minded wanted to see.

Game theoretic approaches were seen as a likely source for solutions to theoretical multi-agent systems problems, but had two major stumbling blocks for exploitation in sectors driving some of the early investment such as the aerospace sector. Firstly, methods based on identifying trend behaviours are not good enough in the world of safety certification authorities, who require evidence of what a system will or will not do based on rigorous modelling and test evidence, not what it should do eventually based on assumptions about convergence to equilibrium. Secondly, it was never entirely clear that robotic or autonomous systems without the ability to judge context and reason are best modelled as rational agents in the game theory context anyway.

Active research interest in multi-agent systems has not gone away, but in recent years attention has shifted into new control paradigms such as swarm robotics. This is a very different approach to controlling large numbers of robotic systems, through assigning goals to a collective of systems and allowing the swarm to distribute activities between them, featuring some degree of self-organisation. Overlapping with this area has been much thinking about whether large numbers of interacting systems could be modelled using techniques from statistical mechanics or population dynamics.

Interesting though these fields are, they are not capable of addressing the problem of synchronised robotics of interest in this work. Statistical techniques are only reliable for large numbers of systems, but many interactive and synchronised robotics applications will have a number of systems merely in the single figures or tens of units – not enough to be able to apply statistical techniques, but too many to readily admit tractable analytical solutions. Swarming techniques do not necessarily face the same restriction (although performance might be impaired with too few systems in the swarm), but face the additional problem that systems within the swarm often need to be able to communicate, which is not always practical nor cost effective in some of the envisaged robotics applications, nor even possible in some cases.

Consequently, the premise of this thesis is that study of multi-agent systems and techniques to mathematically describe their behaviour needs to be revisited, as much of the good recent work is difficult or impossible to apply to some of these niche applications.

2.4 Agent-Based Learning for Synchronised Robotics

There are a number of benefits expected from adopting an agent-based approach to modelling interacting and synchronised robotic systems:

- Autonomous applications need, by definition, to have at least some devolution of control, with systems having local control over their actions. Autonomous elements within a system of systems are naturally framed as agents anyway, and thus the design of such systems immediately lends itself to agent-based approaches.
- Decentralised applications, even if not fully autonomous, may best be operated from within an agent-based structure because it provides them with an environment for conducting operations and responding to environmental stimuli without a complex structure of interfaces and lag in the sending and receipt of instructions and responses.
- If there is a desire for a highly modular architecture with system elements that can be plugged in or pulled out with the system continuing to be operable, then an agent-based architecture may be preferable to a heavily integrated top-down system architecture, as it reduces or removes the need to re-evaluate the performance of the system every time a change is made to it.
- In many applications, particularly driven by robotics, it is desirable to have a number of separate systems operating in coordination with each other. What sounds simple in practice can be difficult to implement, because one can see an exponential explosion in the number of combinations of operations that can be performed once the coordinating systems become more numerous and more capable. 2 systems with 2 modes of operation gives 4 configurations to consider, but 3 systems each with 5 modes have 125 configurations to consider – it is clear to see how this can rapidly become infeasible with dozens of interacting complex subsystems. Devolving matters to lower level authorities makes logical sense, and indeed is exactly what is seen in human and socio-economic systems, but in technological systems this is not so widespread. Agent-based approaches might help facilitate such evolution.

However there are a number of complications that arise from adopting an agent-based architecture:

- Systems design and engineering processes generally struggle with agent-based approaches. Engineers are taught to start with a definition of a system, then from that to derive definitions of its principal subsystems and then any lower level components, then to design, implement and evaluate these components, and then to begin the process of integrating the components together and verifying that the components integrated into subsystems meet the subsystem requirements, and that the subsystems integrated into the envisaged system meet the system requirements. Such methodology is deeply engrained as the way to do engineering. It is more difficult to apply with agent-based approaches, because a set of loosely coupled agents will often experience emergent behaviours and/or demonstrate properties or characteristics resulting from their interactions that are not as might have been anticipated from a structured functional decomposition of the system. As such, multi-agent systems continually frustrate efforts to apply a traditional engineering methodology.
- There is little agreement within the software standards community as to the best methods or tools to adopt for unrolling agent-based approaches in more complex, safety critical systems. In particular, as the regulatory situation stands at the moment, it is not possible to obtain safety certification for multi-agent systems because to do so it is necessary to exhaustively demonstrate a near-zero probability of deviation from defined patterns of behaviour, something which is clearly difficult to do when it is not possible to prove from a system design that there cannot be any unwanted emergent properties.
- On a more practical level, problems arise in agent-based design with identifying how best to decompose systems into simpler parts. Highly coupled subsystems may require re-design or even performance compromise if they are to be separated into distinct functional blocks, which is a non-trivial exercise.

To unlock the expected benefits of agent-based design for complex technological systems that have safety critical aspects, it is necessary to think about new and novel approaches to modelling and design, particularly when it comes to modelling the interactions between agents and generating evidence of what the overall behaviour of a complex composition of agents will be.

2.5 Example Application Areas

There are a number of domains that either feature multiple robotic systems that need to interact with one another, or where there is the potential for this to occur in the foreseeable future. Some of these will be examples of synchronised robotics, where, for practical, cost or operational reasons, the systems will not be able to coordinate with each other, or perhaps even be aware of each other.

It is proposed that many of these examples are naturally represented using an agent-based approach. Some of these are summarised below. As noted in the introduction to this thesis, a couple of potential application areas were envisaged during the original conception and proposal of this PhD, and some initial work was undertaken to develop these. Ultimately a third application aligned to WMG was identified and this superseded the first two application areas. However, in order to demonstrate the efficacy of the proposed techniques and modelling approaches, it is useful to present at a summary level all three example application areas, because they feature different attributes and characteristics that are relevant to the theoretical aspects of the discussion that follows.

Therefore, to preview the approach this thesis will be taking to the application of system design principles to the building of models of complex multi-agent systems, this section provides an initial perspective on what different agents might be present in such systems for each of the three application areas, to illustrate the range of agents and complex system structures this work will subsequently consider how to design and control. These three example application areas are to help frame the subsequent discussion and to provide context to the theoretical developments and the modelling approach.

2.5.1 Vehicular Navigation and the Logistics Management Problem

The ‘traditional’ exemplars used widely in the robotics literature often concern robot navigation, and of autonomous vehicle route finding. The introductory paper on Q-learning (which will be introduced and utilised later on in this thesis) by Watkins [67] was based on a navigation scenario, as were many of the examples within the text by Sutton and Barto [59] which served as a tutorial for the author both during the precursory work prior to and then again during this PhD [22]. Much of the literature traditionally has been focused on land-based lab robots, for understandable practical reasons when it comes to conducting experimental work, but also because two dimensional state spaces provide for a simpler state space model as a

starting point for testing and simulation. However, there is nothing theoretically preventing application to domains with three or more dimensions, and hence there is increasing applicability out in the real world environment, extending into the air and sea domains too (the author's precursor work [22],[23],[24] being an example of such an extension).

Generalising robot navigation to a multi-agent system is conceptually simple – one can imagine a number of individual systems all engaged in autonomous route finding, but in the same or overlapping regions and times. Interaction could be as simple as collision avoidance, where this could not be achieved simply by operating in different areas at different times. More complex interdependencies can be envisaged if the routing decisions are not independent of each other, for example if there was some required sequencing, or conditionality between routing decisions. For example, if A chooses one route, then B must follow it with a time delay, or alternatively B must intersect at a particular time within A's passage, or B must select a separate but complementary route. In any of these cases, B's route navigation problem is not independent of A's, yet A and B remain separate decision making agents who will need some degree of independent or local decision making.

A human context to illustrate the challenges might be provided by two individuals who are making two separate journeys in different vehicles. They are starting at different places and times, but plan to rendezvous at some mutually convenient point on the way. A simple problem with pre-planning and communication, it becomes more difficult if there are traffic difficulties and road closures, and intended routes may not be available, but the activity still needs to be completed (perhaps up against time constraints). One or both individuals may have to replan their journey, which may or may not involve changing the location or time of the proposed rendezvous. Take communication or coordination out of the equation (say due to a mobile telephone battery becoming depleted) and suddenly one individual may have to plan their new route in order to still try to maintain the last known rendezvous point uncertain now about the achievability of it given the unknown status of the other. It should be clear that there is significant scope for failure to achieve the objective, or to only achieve it with unplanned milage and delay. Magnifying the problem up with three or four or five people all making a sequence of planned rendezvouses, and it is clear how vulnerable any plan might be to unplanned interruptions and/or communication failures, and how the number of participants will not need to increase too much before the whole operation becomes infeasible.

How would this manifest itself with autonomous agents? Since the beginning of this work, there has been substantial discussion not just confined to the automotive industry but

increasingly in the wider popular discourse, about the potential for large numbers of autonomous vehicles to be deployed on the roads and public highways, for example to make doorstep deliveries. Vehicles might be able to access digital mapping data and live traffic updates in order to plan and re-plan routes based on a current task or set of tasks, constantly adapting as they move around their allocated locale. Concerns about this include, amongst other engineering aspects, the lack of certainty that they will be able to do this effectively. A large proportion of the autonomous vehicles creating a traffic jam because of overly simplistic programming leading them all to simultaneously choose a narrow side street as the best strategy to avoid delays on the main highway would benefit no-one, and may result in the opposite outcome from what the system calculated would happen when it chose that alternative route. Systems nominally independent of each other and without coordination between them might end up discovering hidden dependencies between their choices.

The logistics management example was one identified in the original conception of PhD. At the time, the context was to be smaller groups of robotic systems operating within a constrained environment such as a delivery warehouse, with a number of robots conducting the picking and packing. The relatively constrained and controlled nature of the environment, and the close proximity and timely response of technicians in the case of problems, makes this an easier scenario in which to adopt a robotic solution, and unsurprisingly this is something which now exists. But the bigger problem, of the delivery of items out in the wider world, remote from direct technical support with much more limited or vulnerable communications and no ability to constrain or control external disruptions, means the more general application remains to be achieved.

Given this general discussion of this application area, and the sort of technical challenges and considerations that will likely arise, an initial view on what agents and multi-agent system structures might arise is presented next. This is to help add context to some of the theoretical discussion in the subsequent sections when features relevant to examples like this arise.

- The systems in this context are individual vehicles or robots. The agent or controller in each case is that part of the system which decides where and when to go in order to best achieve routing or scheduling or tasking requirements placed on that system.
- For context, a higher level ‘system of systems’ layer could sit above the individual systems and be responsible for coordinating and giving instructions or altered requirements to the individual systems. However, synchronised operation occurs when

this layer does not exist, or at least cannot be interacted with for some period of time, requiring the individual systems to make decisions independently.

- The subsystems in this context would be systems onboard the vehicle that make decisions about what the vehicle needs to do in order to achieve the goal it has been given. So this might include selecting a direction (an agent making a decision to turn the drive wheels one way or the other) or selecting a speed (another agent making a decision about how much throttle to apply). More complex systems might have additional decisions at this layer, for example choosing when to issue a stop signal if an obstacle is detected (another agent making collision avoidance decisions).
- Optionally, a further layer of internal agents may exist at the level of the individual pieces of equipment within each vehicle, for example the actuators and servomotors. An agent-based software architecture might allow the control mechanisms for these elements to be agents guided by objectives flowed down from the subsystems above. The theoretical basis for this level of decision making was the subject of the precursor work [24]. This is noted here for context, as this layer of control and any detailed consideration about lower-level agent-based software implementations are outside of the scope of this thesis.
- It is possible to generalise these elements into more complex or nested layers of systems. For example, a small group of vehicles could maintain communications between their group, but not with other systems or groups of systems further away. In this situation, there might be multiple system layers, with or without an overall coordinator layer, then a layer of intercommunicating groups of vehicles each of which can be considered a system, then those groups with more than one vehicle have a further breakdown to individual vehicles, before the subsystem and component layers. Other generalisations or scaling of these conceptual building blocks can also be envisaged, although discussion within this thesis will focus on the simple case.

A visual representation of the types of decision making agents and the layers of the system they exist within is provided by Figure 1.

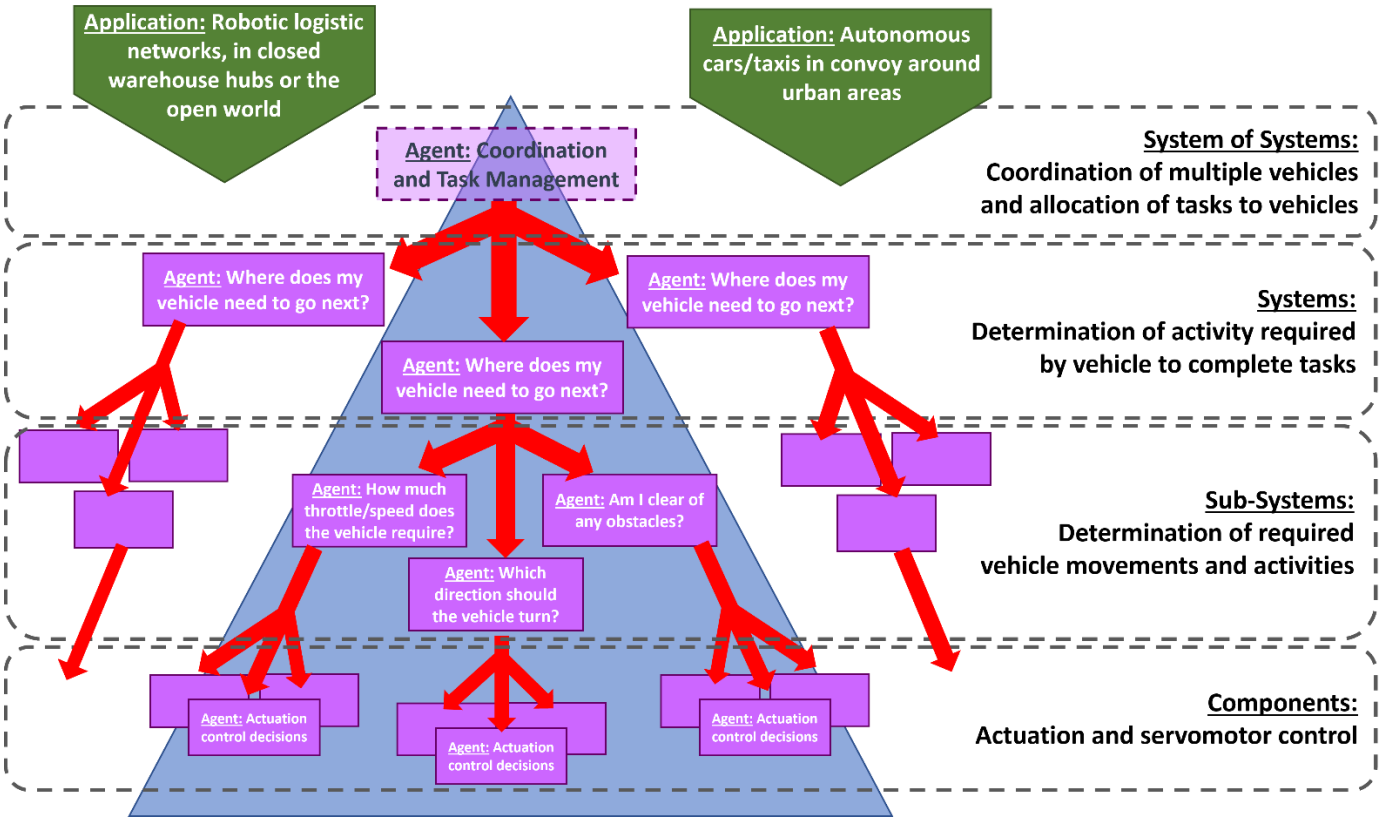

Figure 1 – Multi-Agent System Structure: Vehicular Navigation Application Area

A structured network of agents is formed, with dependencies between layers for task allocation (described hereafter as *vertical dependencies*). One branch of this network (one vehicle and its subsystems) could be said to form a multi-agent system in its own right, although this is not the conventional usage of the term multi-agent system. A more conventional multi-agent system is the incidence of multiple such branches, with or without the higher-level coordination layer. This becomes complex when there is some interaction or dependency between branches, examples of which were described previously (described hereafter as *horizontal dependencies*).

Note that the very top layer may not exist, or at least may not be continuously available, representing the absence or loss of communication and/or coordination described before. With it, horizontal dependencies may require or result in collaborative behaviour from the multiple systems involved, but without it they may instead require synchronised behaviour, as has been defined earlier in this chapter.

Annex B.2 provides some initial mathematical model building for a logistics management problem based on the principles described within this section.

2.5.2 Networked Smart Systems and the Sensor Coordination Problem

Vehicles are not the only application area where one can imagine multiple autonomous or robotic systems having to coordinate or synchronise with one another. A further range of applications is where autonomous assets (not necessarily vehicles or physically mobile systems) are part of an integrated network of autonomous systems (sometimes referred to as 'smart systems'). Non-vehicular examples include applications like communications relay points or cameras connected in a network. These are physically separate systems but they all have to work collaboratively for the whole system of systems to perform effectively. As with the vehicular examples, this could be coordinated behaviour or, in the absence of communications or even awareness between decision agents, synchronised behaviour.

The application area proposed and initially developed for this PhD is that of intelligent and integrated sensor management. The idea arises from turning a number of individual sensors into a reactive and responsive sensor network. For example, a sensor node might be a CCTV camera. A number of these might provide wide area coverage of a whole street, city block, or other public realm. These could be simple (a basic CCTV camera simply records), but future generations of cameras could be fitted with software that can reorient the camera, adjust the focal length or apply filters, to give the camera much greater functionality and versatility. But this gives rise to the question of where decisions are made about how to orient and reorient the camera, when and how to adjust focal length, and when to apply or disapply filters. It is hopefully clear how an agent-based structure might be appropriate for this.

The problem turns into a multi-agent system problem when one imagines applying a number of these smart cameras in a network, to optimise surveillance coverage of a particular area. For example, the objective could be the protection of a public asset, or a piece of critical national infrastructure. It might be desirable, if a suspicious person or object of interest is detected, to be able to monitor them as they move around, which might involve handing over the task from one camera to the next as the monitored object moves from one field of view to another. If the cameras can move, there might be decisions to make about when within a given time period is best to hand over, since one or both cameras might have to move to point in a particular direction to make the handover, which could compromise the monitoring of other objects. If one camera is already maintaining surveillance of one object, should it reorient to provide seamless monitoring of another that has just entered its field of view? If it is applying a filter to clearly identify the object owing to a clash of background colours, should it leave this filter

on when tracking the incoming new object, or will this cause it to lose the original object of interest? Are there other assets that could take on one or other of the two activities?

The problem can be generalised further with the use of different sensor types, for example a mixture of cameras operating in the visual range, with other sensors like radars, acoustic or vibration detectors, or thermometers. Radars on vehicles are used to detect the proximity of other vehicles and obstacles. Could these systems be networked with cameras to identify the nature of the other vehicles and obstructions in an area, and determine which are hazardous? Applications for this can be envisaged in disaster recovery and with the emergency services.

An alternative application area with similar principles is that of communications systems. The author has previously researched the use of agent-based techniques for controlling frequency allocations across a network of closely-located wireless router hubs, to enable local decisions about frequency shifting to minimise local interference without extensive installation testing and mitigation [21], motivated by and extending earlier work by Bullimore and Briggs [14]. Since this work contributed to a previous academic degree it is not repeated here. However, it is noted that this application area could be extended significantly further than the narrow scope of that work, by considering frequency and communication link reallocation across a wider array of devices and antennas to optimise network coverage, stability, performance, or a combination of such metrics through greater delegation of decision making to local systems.

As with the previous section on vehicular navigation, it is now possible to identify the types of agents and the types of multi-agent network structures that might arise within a network of smart systems. The basic patterns are similar to the previous example, and are described below and represented visually in Figure 2.

- The systems in this context are the individual cameras, sensors or communications devices. The agent or controller in each case is that part of the system which decides what that system needs to do in order to achieve the requirements placed on it.
- Optionally, there might be a system of systems layer with a coordinating agent sitting above these individual systems. This will not be practical or possible in all situations.
- The subsystems in this context would be systems within the camera, sensor or communications device that makes decisions about what to do in order to achieve the goal which that system has been given. This might include selecting an orientation (an agent making a decision to turn the camera one way or another), selecting a focal length,

or applying or disapplying a filter. In the communications context, this could be a choice of emitting frequency or bandwidth, or of the time interval within which to emit.

- Optionally, a further layer of internal agents may exist at the level of the individual pieces of equipment within each subsystem, for example the actuators and emitters.
- It is possible to generalise these elements into more complex or nested layers of systems. It is also possible to imagine these as mobile systems installed onboard autonomous vehicles, for example if deployed in crisis or disaster management scenarios. The agent structures for the smart systems then become nested structures within a multi-vehicle structure as described in the previous example. However, the discussion within the main body of this thesis focuses on the simpler case.

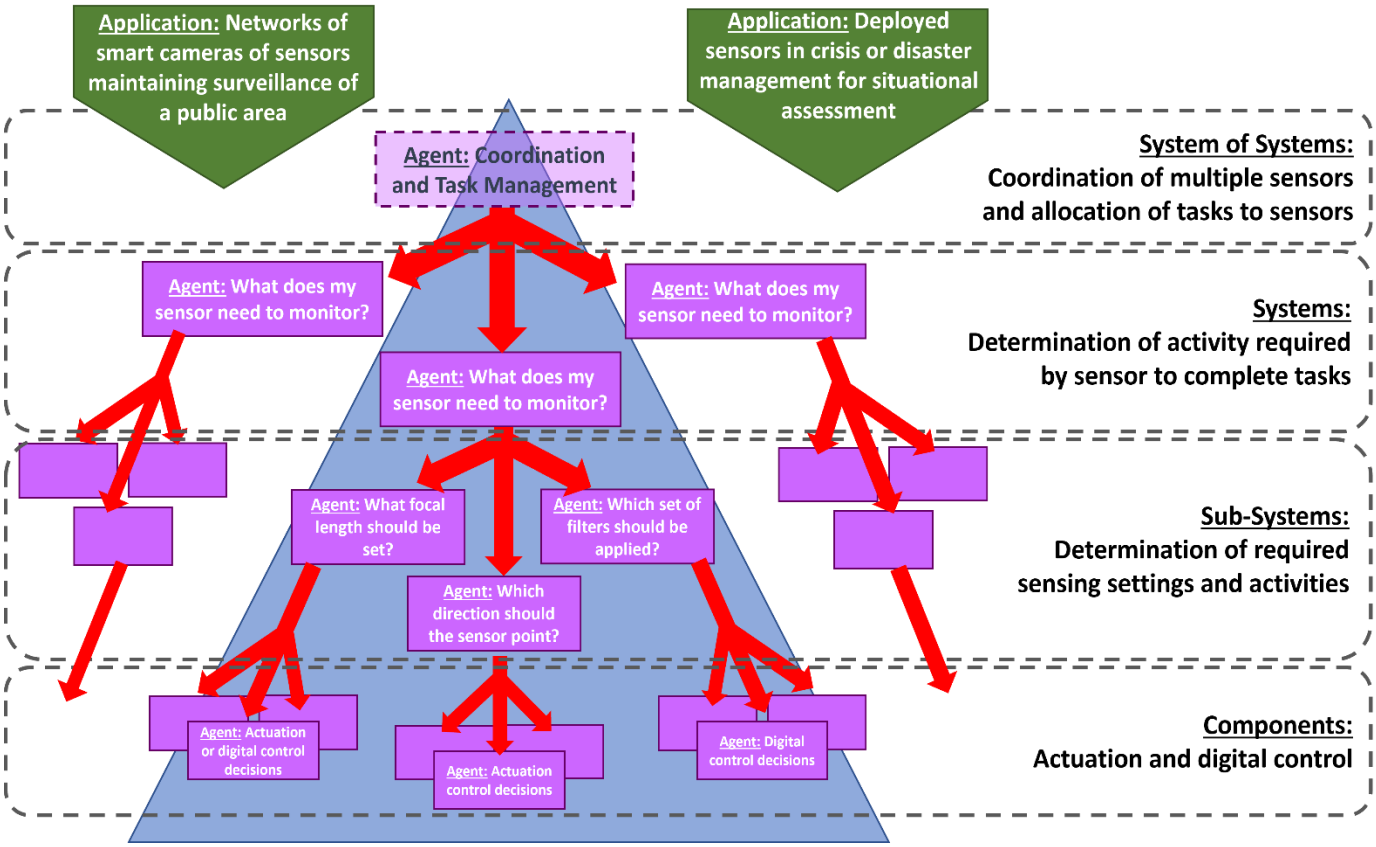

Figure 2 – Multi-Agent System Structure: Networked Smart Systems Application Area

As with the previous application area, vertical and horizontal dependencies can be seen. The absence or unavailability of the top layer of the decision tree combined with a lack of ability to communicate or coordinate at the system layer, would turn this scenario into a synchronised autonomous systems problem. Annex B.3 provides some initial mathematical model building for a sensor coordination problem based on the principles described within this section.

2.5.3 Synchronised Robotics and the Automated Production Line Problem

The third application area identified is that of synchronised production line automata. A modern factory environment, particularly for large-scale mechanised assembly, might feature a number of robotic appendages for positioning, adjusting, measuring, welding, finishing, or simply for holding in position items on the production line. For example, two robots might hold two items in position whilst a third robot welds them together.

This is a familiar real-world example of synchronised robotics, but the principles will extend to more speculative or futuristic applications which the author has seen proposed, such as for automated construction sites with robots replacing manual labour, or the NASA proposals that manned exploration of Mars will realistically require habitation facilities to be assembled or constructed by robots on the planet's surface prior to the arrival of any human explorers.

In general, this application area can be described as being when a number of physically separate fixed assets have individual tasks to complete, and the collective of them together have to produce a specified output engineered to a particular standard in the shortest possible time. There is therefore a non-trivial problem (the 'ramp up' problem) for configuring all of the robotic assets to be synchronised or sequenced in order to conduct their tasks efficiently. Overall organisational efficiency is achieved by minimising ramp up time that ramp up takes.

As the number of assets and the number of modes and functions of each asset increases this becomes a complex problem, increasing the temptation to adopt an agent-oriented approach and devolve much of the task planning to lower levels of control.

Adding the further complication of perturbation (that is, not just identifying a suitable and stable configuration for each asset, but placing them in a dynamic environment subject to change), necessitating that the system configuration be adaptable in order to maintain performance, then an agent-based architecture becomes a natural solution concept which may provide some robustness and adaptability to perturbation.

The basic pattern for a multi-agent system structure emerges, with vertical and horizontal dependencies, and the flexibility to imagine the same basic design pattern being extended with more levels or nested within other larger multi-agent structures as part of a larger more complex deployment. The primary agents and their structure are described below and represented graphically in Figure 3.

- The systems in this context are the individual robotic appendages that conduct tasks on the production line. The agent or controller in each case is that part of the system which decides what that robot needs to do in order to achieve the requirements placed on it.
- Optionally, there might be a system of systems layer with a coordinating agent sitting above these individual systems. This will not be practical or possible in all situations.
- The subsystems in this context would be systems within each robotic appendage that make decisions about adjusting the position and orientation of the joints of that appendage. Typically there will be multiple joints in each appendage. There may be additional agents for specific functions such as determining when to turn the welder on or off, or when to spray paint or polish, or when to activate a saw or a laser, depending on the function of that appendage.
- Optionally, a further layer of internal agents may exist at the level of the individual pieces of equipment within each robot, such as the actuators and functional tools.

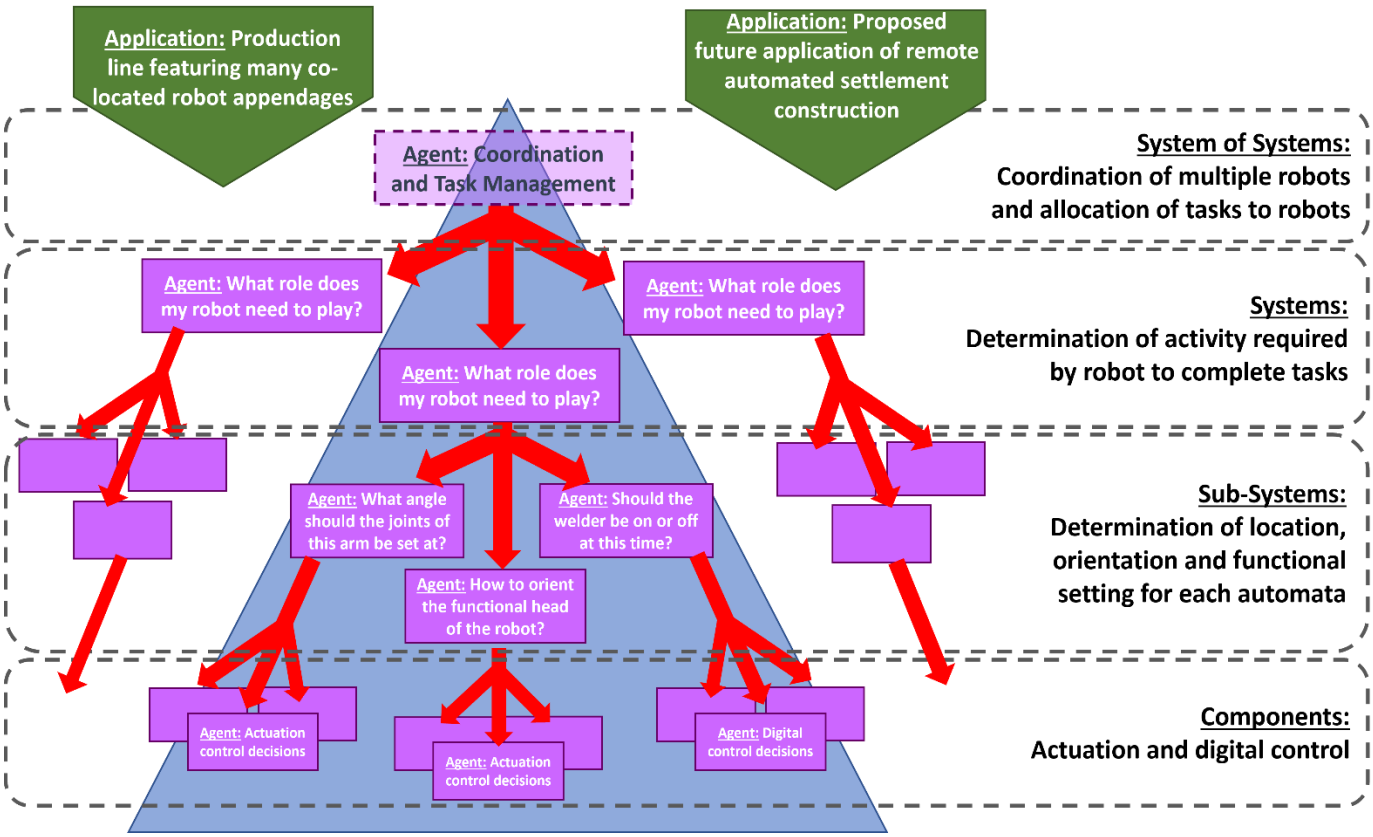

Figure 3 – Multi-Agent System Structure: Synchronised Robotics Application Area

This application area provides the primary exemplar for development within this thesis and will be discussed further in later sections, principally in Chapter 8.

Chapter 3

3 A Systems Science Perspective on Synchronised Robotics

3.1 Context

Complex systems are generally defined as systems that consist of many separate and distinct yet interconnected components, where both the system as a whole and the constituent components each display behaviour that has some kind of observable impact on the wider world attributable to them, but where the behaviour of the whole is not just a composition of the behaviours of the constituent parts. Connecting the components together, even if in an intangible way, can lead to behaviour that arises from the interaction of these components that is different from the behaviour expected from the components operating in isolation. Furthermore, the arrangement of these constituent parts and the connections and linkages between them can result in different behaviours being observed at different points or levels within the system.

Sometimes this is referred to as emergent behaviour, although this term can also be used more broadly for some similar but slightly different phenomena.

When it comes to technology, humans are far more used to understanding and controlling machines as singular entities. Complexity, however, can arise from the design choices and organisational structures behind systems, man-made or otherwise, and complexity in systems should be expected to increase with technological ambition. Technology developers will increasingly have to consider more deeply how they engineer systems, to understand how their design choices influence the behaviour of the resulting system.

To develop mathematical models and techniques for complex systems engineering challenges, it is important to begin from the conceptual underpinnings of complex systems science. Concepts such as complexity, emergent behaviours and complex and multi-agent systems are

sometimes referred to in the wider engineering literature, but often without a consistent baseline for what these concepts are and why they have become relevant. For this thesis, with a key focus on control methods for multi-agent systems, it is important to establish a suitable conceptual baseline, in order to build the mathematical model on solid foundations.

This chapter begins with an initial discussion of what is meant by complex systems and systems behaviour within this work, and defines what is meant by emergence in this context. Multi-agent systems are defined as a class of system which overlaps with complex systems, neither being a proper subset of the other. Organisational or structural paradigms for designing future multi-agent systems are discussed, as are other system design considerations such as observability. The impact of these factors on the multi-agent system control problem is discussed. The overall intention of this chapter is to properly baseline the key concepts and design assumptions that will underpin the detailed developments in subsequent chapters.

3.2 Complex Systems and System Behaviour

3.2.1 The Problem Domain

Apart from the very simplest and essentially trivial examples, when an engineer refers to a system they are typically referring to a collection of components that have been organised or combined together in a particular way, usually to fulfil a specified function or set of functions. More complex systems are organisations or compositions of other systems, typically to fulfil a 'bigger' function than the simpler systems could have achieved by themselves. A particularly complex system could have many such levels of systems in a hierarchy, naturally giving rise to subsystems, systems and supersystems depending on the level within the hierarchy that the engineer's system of concern is to be found.

Indeed this is not just a peculiarity of man-made technical systems. As humans, we are familiar with large complex systems across all walks of life. Consider for example the ecosystem, or the economy, or large organisations, all of which can provide familiar examples with which to illustrate the core problem. There is a whole which consists of many individual parts. This whole has some kind of tangible presence and observable impact on the wider world, as do the individual parts (implicitly assuming some sort of boundary between what is and is not part of the system, although not necessarily assuming that it is clearly defined or distinct). However, we also recognise that the actions and behaviours of the whole are not simply the actions and

behaviours of the individual parts 'magnified up'; the arrangement of these parts and the connections and linkages between them can result in different behaviours being observed at different points or levels within the system. For example, we are familiar with freak tornados that occur despite relatively stable local air mass, and with financial institutions that have invested so heavily in products designed to reduce systemic risk they end up massively increasing it, and with organisational improvement initiatives intended to reduce bureaucracy and improve efficiency in particular functional areas that have precisely the opposite effect over the organisation as a whole.

As humans, we are familiar with all of the examples above, and yet when it comes to technology we are far more used to understanding and controlling machines as singular entities. It is not just that we personally design and build machines – after all, we humans design and build large organisations and, though perhaps less tangibly, we design and build our economy, yet these display complex behaviours. Complexity and emergence cannot simply be dismissed as random or unpredictable natural phenomena that we can never really account for. They are a direct consequence of the design choices and organisational structures behind systems, man-made or otherwise. The reason they are coming to the fore as significant challenges to the systems engineer is simply because until relatively recent times man-made machines were rarely complicated enough to produce the circumstances under which such phenomena could occur. But compared to other human constructs such as civil society or the economic system, it was just a matter of time before our ambitions for our technology progressed sufficiently far that engineers have had to start to consider more deeply how they were designing and organising systems, in order to understand how these choices impacted on the subsequently observed system behaviour.

Autonomous systems pose a particular challenge. Not only does the sort of function to which one might wish to apply an autonomous system potentially imply a relatively complicated system, with a large number of interconnected components and operational requirements increasingly the likelihood of complex behaviour occurring, but, by definition, autonomous systems have some degree of decision making capability at least partially independent of a human operator (remotely-operated or pre-programmed systems are not regarded as being autonomous in this context). Any degree of devolved decision making capability means that the systems engineer responsible for designing a particular system not only has to consider the behaviour and expected impact on the whole system from what an individual component is expected to do, but also from all the things that it might conceivably choose to do, which clearly

can be a significantly greater challenge. With both the technical complexity of and the autonomy granted to systems increasing, so the potential for the system to display complex behaviours also increases, perhaps significantly.

In systems that contain multiple autonomous elements, there needs to be a mechanism that enables both the monitoring and the management of these elements to deliver the desired overall system behaviour. Whilst machines may be able to concurrently observe and manage a great many more elements than a human, it is desirable to avoid unnecessary complexity in this observation or management, and so it may be appropriate to do this at the overall system behavioural level. Furthermore, it should also be possible to constrain the behaviour of a complex autonomous system through the design of the individual system elements such that the behaviour of the overall system can be controlled.

3.2.2 Systems

Traditional control theory might define a system as being those aspects under the influence of an operator when trying to complete a task, presumably (though not necessarily) the *direct* influence of the operator. This implies a clear division between the internal aspects of a system which the operator does control and the external aspects which they do not. These external aspects may impact on the operation of the system, whether in predictable or random ways, or seen as environmental perturbation or noise.

For complex systems, although the essence of this traditional perspective remains, some of the implied limitations need to be relaxed. The precursor work adopted a definition of a system previously introduced by Thoms [63], which remains relevant for use within this thesis:

> **System:** *A system is that body which is considered as a functioning unit, which is formed of many often diverse parts that achieves a purpose.*

This definition places no explicit boundaries on the extent of what is or is not included as part of a system. It does specify *a body*, implying tangible limitation to the system, but it is allowed to be a number of separate components, not necessarily co-located. It also specifies a *functioning unit* that *achieves a purpose*, thereby excluding seemingly random coincidences of factors that cause a particular outcome.

Purpose is normally thought of as a system property that drives a desired outcome, and this thesis will not attempt to define this any more tightly since the nature of a system's purpose will vary greatly between different types of system. Note that a system's purpose can be seen

to be analogous to the concept of a system *goal*, which is the terminology used in much modern control theory literature.

The boundaries, composition and purposes of systems are always in the eye of the beholder. When a beholder observes what they think of as a system, they bound what they are observing and infer a purpose, and their understanding of that system is against the current context. It should be understood, however, that different beholders may perceive a system in different ways – either inferring a different purpose, or considering different components to be part of the system, or both. Consequently, it is likely that in highly subjective situations there will be different determinations as to how a system is behaving, which could arise from a difference in perspective on what the system in question actually is.

3.2.3 System Behaviour

The definition of behaviour from Thoms [63] also remains relevant:

> **System Behaviour (general):** *The manner in which a thing acts under specified conditions or circumstances or in relation to other things.*

Where the 'thing' in question is a system, and where the conditions, circumstances or relations to other things referred to in the definition can be summarised as the system's *state*, this can be reworded to provide a definition of system behaviour more reminiscent of that within modern control theory. As in the precursor work [25], a slight variation of the definition is therefore adopted within this work:

> **System Behaviour (specific):** *The manner in which a system is seen to progress from one state to another, which can be quantified by one or more parameters and fits into some predefined category of system performance.*

This definition provides that behaviour is to do with actions in context, and corresponds to some property that can be observed. Consequently, the understanding and classification of behaviour may be subjective.

3.2.4 Complex Systems

When there are a number of separate and distinct systems operating individually, it is possible that the actions undertaken by some systems influence the behaviour of others, possibly despite the absence of an obvious tangible connection between them. Whether tangible or not, if there

is some implicit connection between systems then they are forming a network, hence the phrase 'network of systems'. 'System of systems' is an alternative description meaning essentially the same thing. A multi-agent system is a more specific instantiation, where the individual elements that make up the larger system are necessarily intelligent agents, with an independent decision making capability but typically without a global world view or a single point of control. Some discussion of these concepts has already been presented within this thesis in Chapter 2, with example applications described in Chapter 2.5.

Despite this multitude of labels, all of these are (or can be) examples of *complex systems*. Other labels can surely be found elsewhere in the literature. For the purposes of this thesis, the following (also derived from Thoms [63] but adapted in the precursor work [25]) provides an overarching definition of a complex system (and of complex behaviours):

> **Complex System:** *A system comprising multiple interacting parts which exhibits characteristics and behaviours which are not associated with the individual component systems acting in isolation.*

Such systems can exhibit a wide range of behaviours, which can evolve out of the interactions between individual components and with the local contextual situation in which they are operating. It is this capability which enables them to respond to different operational challenges in different ways, bringing capabilities traditional unitary or monolithic system might not have been able to bring but making the human operator's challenge more difficult.

Note that with complex multi-agent systems, purpose is a result of the individuals and the interaction of the individuals, who may not be aware of the overall system purpose but will be working towards their own local goals or purpose. Generally, such systems should not be assumed to have a well-defined purpose or goal commonly understood throughout the system (indeed, such inhomogeneity of purpose could well be the cause of any complexity).

3.2.5 Multi-Agent Systems

A variety of definitions for an *agent* and a *multi-agent system* can be found in the wider literature so, to be clear what is meant within this thesis, the definitions used in precursor work [25] is again carried forward.

Firstly, an agent in colloquial English means an entity (typically a person) responsible for making decisions on behalf of another entity. In software engineering, a software agent is a piece of software that functions as an agent for another part of the system (which could be other

software components, interfacing subsystems, or potentially the human user). For this thesis, it is not necessary to go into detailed discussion of software engineering details and standards, as different specific meanings and definitions are in use around software agents. It is sufficient to define an agent as:

> **Agent:** *An entity within a system at which decisions are made which affect the behaviour of a part of that system.*

An agent-based system is a system that contains one or more agents. The precursor work considered one agent controlling an autonomous vehicle [23],[24] building on [15],[68]; this PhD has focused on the multi-agent situation, where a system contains more than one agent. Based on the definitions from the precursor work already given, the following is naturally derived as a working definition for a multi-agent system:

> **Multi-Agent System:** *A system that consists of multiple parts each containing at least one agent making decisions that affect the behaviour of that part.*

With this general formulation, agents can be arranged without constraint within the system. This definition allows for agents to be arranged hierarchically, with a natural ordering or layering based on which part of the system may be regarded as a supersystem or subsystem to which other parts. It also allows for them to have no hierarchy whatsoever, all existing in a single system layer acting in parallel to one another. More complex arrangements with elements of hierarchy and parallelism can easily be envisaged. All of these are permissible within this general formulation. As has already been seen with the examples in Chapter 2.5, and particularly via the graphical representations of multi-agent system structure contained in Figure 1 to Figure 3, this decomposition into hierarchical and parallel is a useful design pattern which will reoccur.

As noted previously, systems, and their boundaries and purposes, are in the eye of the beholder, with inherent subjectivity about what each of these things might be in any given context. Agents have decision making responsibility for parts of a system, but if the boundaries of these parts are subjective then so might be the allocation of responsibility to agents. Fortunately however, these considerations tend to be more applicable to human systems. With technological systems, a software agent will typically have defined interfaces and functions, and so the structure of the multi-agent system should be objectively definable. It should be noted that advanced concepts such as artificially intelligent ‘self-tasking’ agents might breach this assumption and require deeper consideration, but these are beyond the scope of this thesis.

Complex and multi-agent systems are closely related phenomena, but they are not the same thing. Multi-agent systems are often complex, and they present a significant challenge to the system design process, to understanding and controlling system behaviour, and to system assurance. However, multi-agent systems could, in principle, display non-complex characteristics and behaviours, if the interactions between the agents are carefully designed and constrained and operate in a well-defined environment. Conversely, complex systems are a much broader family than just agent-based systems, covering a wide range of system concepts. Some aspects of the techniques developed in this work may be applicable to other forms of (non-agent-based) complex system, or to non-complex multi-agent systems, but the specific focus of this thesis is on complex multi-agent systems.

Figure 4 illustrates the relationship between the two concepts by way of a Venn diagram:

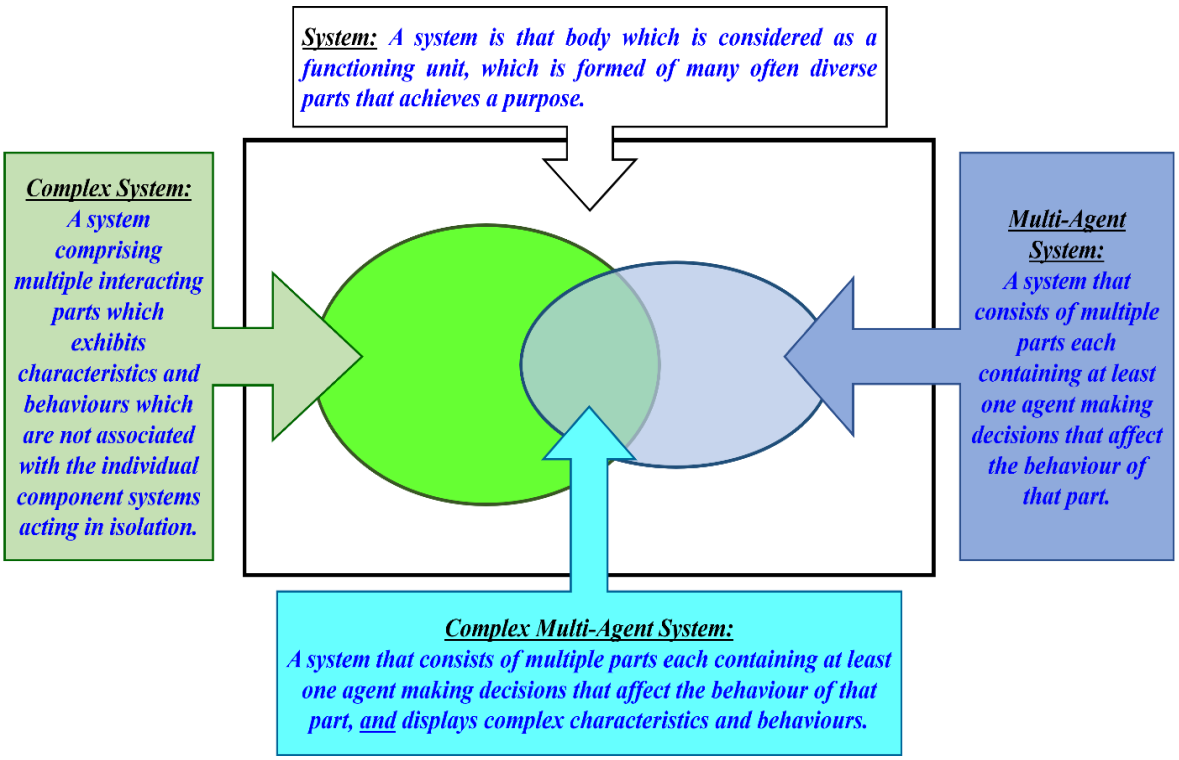

Figure 4 – The Relationship between Complex and Multi-Agent Systems

3.2.6 Limitations of this Work

The field of autonomous agents and related technologies within academia is diverse and in general has focused on delivering specific solutions to agent problems at the low level. This thesis is not going to discuss specific examples of agent technology, software standards or

techniques used in the implementation of agent-based software. Instead, this high level discussion is to provide the baseline assumptions based on principles from system science, from which a mathematical framework for multi-agent decision making can be developed.

As noted above, agents that can independently evolve are excluded from this work. Concepts from artificial intelligence of agents that are able to define their own scope and purpose might not be easily controlled, as they could potentially evolve to remove, limit or work around any mechanisms used to control them. Also excluded from scope are systems that do not have at least some degree of localised decision making capability of their own, or which have not been designed or implemented to fulfil a specified function or purpose, since any such systems (whilst potentially complex, for example many naturally occurring systems) would not make sense to be built as multi-agent systems, nor indeed would engineers be likely to be actively developing them without specified function or purpose.

3.3 The Organisation and Structure of Systems

This section considers the implications of the organisation and structure of a system, and how this can influence the observation and the control of such systems.

3.3.1 Background

A system, and an observer's relation to that system, was defined by Thoms [63] as:

> *"A system is that body which is considered as a functioning unit, which is formed of many often diverse parts, that achieves a purpose. But we need to remember that a system is always in the eye of the beholder. When we observe what we think of as a system we bound what we are observing, we infer a purpose and our understanding of that systems is against the current context. We make assumptions as to what is contained in that system and seek to recognise patterns relating those elements to each other so that we can understand their complex interrelations. And yet what we observe as systems behaviour is observed at a higher level that any individual cognisant entity that it may contain. Our expectation is that the system satisfies the purpose for which we perceive it and we will even attribute it with the flexibility to meet changes in its environment."*

The observer or the beholder may ultimately be the user of a system, but will be the engineer or designer of the system in the early system design phase. An engineer or designer would expect to have knowledge of the whole composition of the system, whereas a user may (by choice or by design) only see part of the system relevant to them. There is therefore some subjectivity about system organisation and structure of systems just as there is about system function and behaviour, and this distinction should be kept in mind when considering the implications of organisation and structure on systems.

When designing a system, the engineer has to define the intended purpose of that system and choose architectural rules or patterns that are appropriate to that purpose. A defined and presumably fixed purpose implies a fixed structure to the system, which will not change in response to the environment. Whilst this may be true of, for example, the hardware from which the computer on one's desk is constructed, it is not true of the software that is running within it. The software components are being constantly loaded, run and unloaded in response to requests from the user, from the operating system and/or from any networked devices it is connected to. This behaviour is a result of control algorithms or rules that one might recognise as reactive autonomy, and the same type of behaviour is exhibited in networks.

For more sophisticated systems capable of behavioural autonomy, the engineer has to define more than just a physical architecture. They must define an organisation, including the rules and constraints that elements must abide by, and this requires them to understand the triggers for characteristic network behaviours. To understand the operator's perspective on these systems, the engineer also needs to recognise the organisational paradigms they may have chosen, and the implications their choices could have on the observation and control of both the organisational behaviour and of the overall system in its environment.

The problem space changes further when considering systems with deliberative elements. These elements have the capability to decide how to organise their own group structure to achieve goals supporting the system purpose. To understand the operator's perspective in this case, one would need to consider how the operator might recognise the emerging organisational structures within the system, as well as the potential control mechanisms.

3.3.2 The Impact of Networks on System Behaviour

Complex behaviours arise from the interactions between independent elements of systems, and these interactions mean that elements are implicitly or explicitly forming a network of some kind. Since networks display basic reactive autonomy based on simple rules, one should expect that network behaviours might influence the behaviours of complex systems. For example, behavioural agents may learn that interacting with certain other agents gives them increased access to the information they need, thereby creating scale free networks based on their evolving goals. Deliberative entities may choose to interact with agents with similar interests, thus creating their own small world networks which can grow and evolve over time.

To understand the behaviour of any system, it is necessary to first recognise the nature and cause of the behaviour of any networks within that system, as this network structure is a part of the environment that influences what behaviour is observed. Consequently, it is important to briefly recall the types of network organisation that could be present and whose properties and characteristics might be influencing system behaviour.

3.3.2.1 Formal Networks

Formal networks are networks that have a defined structure and fixed rules. As such, they are inflexible and are applicable only to rigidly defined, unchanging problems.

In the real world, many networks will need to change and modify themselves. The example application areas described earlier in this thesis could be described using a formal network in some cases (the production line automata for example), but this would severely limit the ability to operationally and organisationally adapt in other cases (discussed further in Chapter 3.3.3 below). To allow for such adaptation, informal networks will need to be investigated instead.

3.3.2.2 Random Networks

Random networks are networks which have a defined number of nodes within which connections are made, not with any specific intent but randomly. These networks are known to demonstrate sudden changes in information flow due to what is called percolation, which occurs when the number and transfer rate of random links exceeds the volume of information that is attempting to pass through it.

Random networks can be recognised by the degree of distribution of the nodes and the clustering coefficient (the ‘goodness’ of the links between neighbours).

This kind of network structure would be appropriate to a system whose composition varies – systems within the system of systems leaving and joining during on-going operation. This is relevant to some autonomous vehicle concepts (discussed further in Chapter 3.3.3 below).

3.3.2.3 Small World Networks

Small world networks are networks that use some sort of attribute or property to guide the creation of the links between nodes, and hence nodes with these similar characteristics will seem to be clustered. As a result, information that in some way relates to the attribute or system property that is guiding the creation of system links will have a shorter path between its source and destination that it would have had in a random network.

Among the properties of a small world network are that it has a relatively small average path length and high clustering coefficient. When this is combined with a small number of longer length links, it produces a network that becomes more richly interconnected (biased towards information transfer) than random networks.

This kind of network might be relevant to systems where there are multiple systems that cooperate on particular tasks for a period of time, and therefore form a more closely coupled cluster within the wider network for the duration of that task. Whether these clusters form through reactive or deliberative autonomy would depend on the application and implementation in question.

An example of this would be the smart sensors application, where closer linkages between subsystems might be required during handover of a monitoring task from one smart camera to another – for example, seeing links forming between the subsystems of two particular cameras as they react to each other to ensure they are pointing in the correct direction and aligning focal or filter settings to ensure smooth and continuous coverage. This would manifest itself as those branches of the wider multi-agent system network forging additional denser links between agents within those branches that might not otherwise have been observed, forming a denser cluster within the larger network.

3.3.2.4 Scale Free Networks

Scale free networks grow and evolve over time; they increase not only the number of nodes but also the links between those nodes. As new nodes are added to this type of network, their preference is to be connected to nodes with a high degree of connectivity, a process known as

preferential attachment. As a result, plotting the numbers of connections per node results in a graph following a power law. It is this power law that is the key characteristic of this type of network and gives it its name of scale free.

In these networks, the points of vulnerability are the hubs with high degrees of connectivity. But because of their ability to evolve, nodes will seek to reconnect themselves to anther node with a high degree of connectivity and quickly heal the network. The internet is a scale free network that everyone is familiar with, but one should not seek to interpret intentionality from the observation of the flows within it.

3.3.3 Relevance to the Example Application Areas

Relating this discussion back to the application areas described earlier in this thesis in Chapter 2.5, it was clear that network structures could be identified with decision making agents operating in parallel but with (perhaps hidden) dependencies on each other – the multiple autonomous vehicles, or the individual smart sensors, or the individual robot appendages in each case. A hierarchical structure connecting these agents to various subsystems could also be recognised. Each of these multi-agent systems (and the multiple system layers within these multi-agent systems) can be regarded as a network, with each node the location of one of the decision making agents, and the dependencies between nodes being the edges.

Some of the applications would be more naturally represented as a fixed network. For example, the production line automata application would likely feature a fixed number of assets with defined roles, and this is unlikely to change (at least during run time). Nonetheless, failures of individual appendages might mean others could not complete their tasks. With some reactive autonomy within the system, it might be (at least theoretically) possible for some other agents in the network to adapt to this change. For example, if two robots are required to lift some pieces together for a third robot to weld, then if one robot fails the others might be capable of ceasing welding or to return their pieces to rest, as a preferable outcome to continuing to operate and achieving nothing more than seared edges of wasted metal suspended in mid-air.

The vehicular navigation application is a good example of the opposite case of a dynamic network. The number of vehicles within the network operating at any given time may vary. This could be because there are defined geographical limits to a particular system (for example, traffic control within defined city limits) and vehicles might go beyond these limits on individual journeys or re-enter, temporarily leaving and later re-joining the network. There

could also be idle time in between any peak times of operation. Off peak, a proportion of the vehicles might be idle – technically still elements within the system, but not currently in service. Depending on the implementation, one might regard these as having been off the network and subsequently re-joining. Temporary unavailability of communications could also lead to the leaving and re-joining of systems from the network, from the perspective of the rest of the network. All of these scenarios, applied to any version of the vehicular navigation application, result in the internal structure of the multi-agent system being a dynamic network. If there is the capability to 'accept' visiting vehicles from outside into the system (for example, traffic coordination on a larger scale where vehicles might be permitted to enter from another jurisdiction) this might even manifest itself as a random graph.

Reactive autonomy in this situation might be seen as, for example, a vehicle failing or otherwise becoming unable to make a planned journey, with another vehicle redirected to complete this task instead. A more advanced implementation with some degree of behavioural autonomy might see lowly utilised vehicles recognising if an uneven distribution of vehicles was emerging and choosing to move to more sparsely covered areas in anticipation of future demand.

The engineer or designer of the system has a great deal of flexibility to determine the degree of functionality that they might wish to design into the system (notwithstanding the question of whether they can get the design approved), but this will have implications on the structure of the resulting multi-agent system, including on the internal network dynamics and the required network structure. It will be an important consideration in the development of a modelling and analysis approach for multi-agent systems that flexibility be maintained to account for this range of solution concepts. Networked learning agents provide a theoretical framework that can remain relevant to all of these examples in a way that other more rigid structures might not.

3.4 Understanding Observed System Behaviour

To evaluate the behaviour of a system, one first needs to observe the system in order to:

- Categorise system behaviour into states, and be able to make judgements about current and future system states from available local or global information.

- Determine whether a system is in a desirable or undesirable state, and whether it is likely to remain in desirable or undesirable states.
- Assess whether the performance of the system will remain within an acceptable state (or within a set of acceptable states, or within an acceptable bound), and if not to identify suitable actions to take to ensure that it does so.

There are some key challenges relating to understanding observed system behaviour. This section discusses some of the key challenges related to the observability, classification and control of system behaviour that arise when trying to understand the behaviour of systems on local and global scales.

3.4.1 Observability

In multi-agent systems, the behaviour of the system is intrinsically linked to the behaviour of the individual agents. The engineer needs to be able to answer a number of questions:

- What to observe?
    - How should one abstract global properties from observing some properties of individual agents? Can these abstracted properties be used as indicators of global behaviour or health?
    - With access only to limited information, for instance the knowledge of a subset of agent states, or a subset of agent interactions and their state transitions as a function of time, then how can this data be extrapolated to give extended patterns of local behaviour and useful insights into global behaviour?
- How to observe?
    - Should the observer be internal or external to the system? (This could be related to the question of how to ensure that the presence of any supervising agents has a minimal impact on the behaviour of the agents being supervised.)
    - How is the observer to communicate with and influence the system? How does this add to the load on the resources of the system?
    - As different behaviour can be observed at different levels of granularity within a system, then what is the appropriate level for observation and analysis of behaviour? Might multiple observers at different levels in the system be required?

The observer's perspective may influence how they will perceive reports from the system. It has been argued that system elements in social systems interact with an understanding that their relative perceptions of reality are related [9], and that they may each act against this common understanding. They will have a common understanding of their environment and that they will have little understanding of how other social systems perceive their environment. This could be described as an internal system culture [63].

This viewpoint has implications for the observer of such a system that the observer may not share the same perception as the members of the system against which to interpret their observations or the reports that may be provided to them by the system. Applied to multi-agent technological systems, this means that the engineer's or user's perceptions of the reports and/or data obtained from observing different parts of a system may be interpreted against different viewpoints of what that system should be doing and how it is to be regarded against real-world context. This means the system designer will need to understand the issues involved in using local information to make global predictions and vice-versa.

3.4.2 Behaviour Classification

Mapping observations on to system behaviour leads to a new set of questions:

- Is it possible to map patterns of local agent behaviour to a pattern of global system behaviour? Is there a functional representation of this relationship?
- Using only certain features of local behaviour, is it possible to develop a classification scheme for global behaviour?
- Can a specific global behaviour pattern be related back to a specific combination of individual agent behaviours?
- Is it possible to simulate the behaviour of the whole system by using only some important constituents and representative interactions? Or to recreate a pattern from a particular set of significant features?

3.4.3 Control

Moving towards higher levels of autonomy to deal with changing operational requirements and dynamic environments will require control techniques that can be adapted to the evolving context. This prompts the following questions related to control:

- What in the system (states, value functions, communication links…) can be changed, and under what circumstances?
- Should there be a single controller or multiple distributed controllers? If there are to be multiple controllers, how would these controllers interact?
- Should the controller be external to the system, or could there be a self-regulatory mechanism within the system?
- Can global behaviour be influenced by controlling local behaviour?

It will also be important to understand how to evaluate control techniques:

- How to judge if the control techniques developed are robust?
- What techniques are most efficient (in the sense of using minimum computation to achieve good control)?

3.4.4 Relevance to the Example Application Areas

The considerations in this chapter about observation and control are very general in nature. Chosen solutions will inevitably be context specific and the reason for expressing the issues raised in this chapter as questions is because there will not be universally applicable answers.

However a few general comments can be made regarding the example application areas from Chapter 2.5. A higher level system-of-systems layer coordinating the activities of the other elements in the system was included as an optional layer of the system for each example. There are other theoretical challenges with this approach (to be discussed later in Chapter 4.3), but a more immediate problem for the presentation within this thesis is the impracticality of having or at least maintaining this layer, hence the focus on the synchronisation problem that emerges when the separate systems within the multi-agent system are still linked in a network of dependencies but are no longer able to communicate with each other. In each example, the system-level agent (that controlling the tasking of the autonomous vehicle, the smart camera, or the production line robot in each case, or the second layer of agents in the illustrations in Figure 1 to Figure 3) acts as a controller for its branch of the decision tree, but in the absence of an overall controller for all branches of the whole multi-agent system. Different structures could be envisaged for different applications.

Chapter 5 will define an approach to constructing a state machine model for such systems, and later chapters and annexes will demonstrate how one might apply this general state machine

model to each of the example application areas. Action spaces are also defined as part of the model, where an agent choosing an action causes the system to experience a state transition. State transitions are the representation of behaviour using this approach, and as the examples are all of physical phenomena these will be observable state transitions or system behaviours.

It should be noted that this is not claimed to be the only way to address the issues identified in this chapter. In particular, digital software systems might wish to consider further what an observable behaviour means in their context. Nonetheless, this thesis develops an approach to addressing the issues identified in this chapter, suitable for the example application areas identified and potentially for other problem domains too.

3.5 System Performance

The next challenge that arises is to understand what measurements are possible and which of those measures might be of use.

3.5.1 General Agent System Measures

At an individual agent level it should be possible to gain access to the agent's current status, goals, and attribute values. These values include the agent's current goal and the values that it will use in making its choices.

The engineer will have to consider how or why an observer would want to investigate a specific agent. It may be more appropriate for the system to include agents (sometimes described as master agents or sentinels) that take responsibility for a subsystem of multiple other agents, and it is from these agents that the user or operator would seek status information.

To undertake this type of measurement, the agents (whether any or all agents, or just specified agents) would need to be designed to support external interrogation.

3.5.2 Group Measures

At higher levels of system abstraction, it might be necessary to measure groups of agents rather than individual agents, as the useful measurements may not be a function of the agents themselves but of the architecture and network structures in which they reside.

To be able to observe and measure group dynamics, the system architecture needs functions to support the identification of interactions between agents. This could include functions allowing the identification of:

- The routing of messages between agents.
- Group stability and dynamics.
- Internal group interactions.
- Lines of communication to other groups.
- Volume of communication between individual agents within a group.
- Embedded dimensionality (clustering of nodes).

To understand what was happening at the group level it would then be necessary to classify these measurements. An alternative technique would be to have a 'group coordinator' allocating roles and assignments, who could provide group level status interrogation facilities. Note that this is not the same role, nor would it necessarily be conducted by the same agent, as the sentinel role mentioned previously, should that also have been adopted.

3.5.3 System Measures

There are other measures that may be useful at the system level. These measures may be similar to the types of measures that one would use for monitoring a traditional monolithic system, but in addition the engineer will also need to consider how to measure:

- The available system behaviours. (This could be by observing the number of agents and the types of agent available in the system and using this to reference a set of defined system behavioural patterns.)
- Degradation modes and effects.
- System evolution.

By monitoring the resulting effects from the interaction of the system with the context, it should be possible to identify if the agents (or groups of agents) are following any discernible pattern. This would enable the observer to predict the evolution of the system behaviour by identifying patterns from the current and previous goals. The completion of each goal at a group level could be observable if or when the system nodes (agents within groups) change.

3.5.4 Relevance to the Example Application Areas

It is not difficult to see where measures of individual or groups of agents may arise within the example application areas. Focusing specifically on the production line automata example, as the primary example developed later in this thesis, measures that currently describe the state and performance of the overall system would include aspects like current position and orientation of tools and devices, activity throughput and completion rates, and fault reporting as and when required. Some of these measures might make more sense as a group measure; for example measures concerning task throughput and completion would make most sense associated with the group of robots collectively supporting the same activity, as there should not be any difference in the measure between members of the group. Overall quality control measures such as fault rates only make sense at the overall system level. Therefore, a practical design for such a system would need to ensure that the necessary data for reporting each of these measures is available at the appropriate layer of the system hierarchy.

3.6 Addressing the Key Challenges

This section considers how constructing models of complex multi-agent systems using the mathematical basis introduced in the precursor work [23],[24] and reintroduced and further detailed in this thesis (from Chapter 5) can provide a framework for modelling and, hopefully, understanding the behaviours observed from such systems. This understanding might help address the key challenges that face the user when observing and trying to classify and control the behaviours of such systems.

3.6.1 System Observability

The questions related to observability can be classified as:

- What to observe?
    - If one has access only to limited information, for instance the knowledge of a subset of agent states, or a subset of agent interactions and their state transitions as a function of time, then how could one extrapolate the data to give extended patterns of local behaviour and useful insights into global behaviour?
    - In multi-agent systems, system behaviour is intrinsically linked to agent behaviour, so what techniques could be used to abstract global properties from observing some

properties of individual agents? Could these abstracted properties then be used as indicators of global state?

- How to observe?
  - Should the observer be internal or external to the system?
  - If the observer is to be external, how should the system communicate with the observer? How will this add to the load on the resources of the system?
  - If the observer is to be internal, how should this mechanism be implemented? Should certain agents be designated as monitoring and/or sentinel agents?

3.6.2 Patterns of Behaviour

A system can have static and dynamic patterns of behaviour.

3.6.2.1 Static Patterns

At the individual agent level, patterns can be recognised that relate to the individual agent's current state and the triggers that will cause that agent to change to another state. In systems that contain many elements, this way of observing the system quickly becomes impractical due to the size of the observation challenge.

Above this, one can recognise that the agents will interact with other agents within the system exchanging information. These interactions can be considered to link communities of agents that may or may not be working together to achieve the same goal. Where there are multiple individuals working together towards the same goal it is likely that their interactions will follow a determinable pattern which could be represented with some sort of abstract representation or label.

It may also be of use to consider the spatiality or locality of these groups of agents within the system.

3.6.2.2 Dynamic Patterns

Agent systems are dynamic systems so it is important to consider how to observe this dynamicity without overloading the observer. Within the system there will be constant changes to the internal states of the agents as a result of events in their local environment. It may be useful where there are a number of homogeneous agents to be able to understand if these agents

are in similar states (a stable system) or diverse states, which could give an indication as to the nature or local context of the agents.

An engineer will want to be able to observe changes in the interaction between different agents as well as interactions between agents working as teams or communities. In response to the environment they will want to be able to observe changes to the volume of communication between agents.

Both of these could be used to attract the observer's attention in order to get them to observe the locality of the agents to understand the change in the context, which in turn could change the way in which they interpret the system behaviour.

There may also be occasions where individual agents become excluded from the dynamic system functionality, either temporarily (when they are not required to deliver the current goal) or permanently. Where these agents are heterogeneous, the exclusion of certain agents (or types of agent) may provide an indication of the type of goals currently being undertaken by the system.

3.6.3 Observing the Outcome of the Agent/Group Behaviour

An alternative approach to observing patterns in the agents would be to seek to observe the results of their execution within the operational context. With knowledge of the agents' internal plans and group plans it might be possible to infer the evolving 'string' of goals that the agents are working towards.

It should be possible to recognise patterns in these goal sets, which may allow the observer to understand possible causes of the agents' historical behaviour and to predict possible future behaviour.

When considering the challenge of prediction, the observer is interested in being able to predict what the system may do in the short term and to be able to understand how the system behaviour may evolve in the longer term. Key to this is the ability to understand the level of uncertainty in the observation and how that uncertainty may increase in the future.

3.6.4 Relevance to the Example Application Areas

Some of the considerations described in this chapter apply to the example application areas. Because of the focus on problems where a coordination layer is not present, the synchronisation

problems considered herein clearly feature an external observer (the human user). If there were some degree of human control or a coordinating agent, then it would be necessary to consider what the key measures are that would need to be observed and what would be the best mechanisms for doing so, in order to facilitate this. This might take the form of providing distribution and grouping data and current task assignments of the vehicles within a fleet of unmanned taxis, to assist a planner when deciding where to allocate new task requests.

The production line automata example would fit naturally into the paradigm of static patterns of behaviour; once a system is configured and operating effectively ('ramp up' having been achieved) it would not normally be expected to change during operation. A number of robotic appendages all conducting their various tasks, but with regularity such that observable and repeating patterns of behaviour are apparent to the external human observer. It should also become speedily apparent when physical faults occur, because the patterns will change, where a particular appendage no longer acts the way it did before. The example of the multiple autonomous vehicles could display static patterns (for example if they were conducting fixed and repeating journeys such as daily logistics deliveries) or dynamic patterns (an autonomous taxi service would see patterns determined by external customer demand and would surely not follow a static pattern given different service requests coming in from unplanned places and times). The smart sensors application has attributes of both, with a fixed pattern of behaviour for routine monitoring and surveillance, but the potential for dynamic patterns to occur if the system has greater autonomy to, for example, prioritise tracking of certain objects of interest and to allocate and prioritise resources accordingly.

3.7 Complicated vs. Complex System Behaviours

It is important at this stage to clarify that complexity of systems does not merely mean there are too many individual elements for an observer to keep track of, or that the individual behaviours of some or all elements in the system are difficult for an observer to comprehend. Colloquially these might be described as complex, but this would be loose language. A complicated system does not equate to a complex system.

For the purposes of this thesis, complicated means that there are many components and the properties and behaviours are difficult to comprehend all at once. However, these complicated properties and behaviours may still be entirely logical and predictable, and just because there are many components making analysis difficult does not necessarily mean these components

are interacting or interfering with each other. To be complex, there has to be some 'difference' in property or behaviour caused by interaction (discussed further in the next section on emergence). Complex systems are often complicated, because a complicated system with many components and many internal dynamics is more likely to have components interacting with each other, but it is not the same as merely being complicated. Although it is counter-intuitive terminology, relatively simple systems could be complex. Only two elements are needed for there to be an interaction that changes the combined properties or behaviour, but a two-element system is unlikely to be considered complicated. Complicated and complex are therefore descriptors of different system conditions. Referring to a system as a simple system is unspecific as to whether one is referring to the opposite of complicated or the opposite of complex, because these are generally not the same thing.

What impact does this have on the systems engineer? In one sense, not a lot. Both complicated and complex systems present engineering challenges, in the sense that systems design and proving is about determining and designing in required behaviours from a system, and then determining and/or demonstrating the appropriateness or suitability of what is actually achieved. Both complicated and complex systems present a difficulty in that it may not be straightforward to do this. Software engineering in particular suffers, because the standard approach is, in essence, to test every possible string of commands from every possible state, and this can be an enormous number of combinations. There is a need for tools and methods more generally for making this process more efficient and effective.

However, complex systems present an additional difficulty beyond potentially being quite complicated. If the behaviours and properties of the systems being analysed are not as might have been predicted from an analysis of the components themselves, then an analysis of how the system is expected to behave from an understanding of what its components will do can be logically invalid, presenting a significant stumbling block to standard processes to assurance (this is discussed further in Chapter 4.4). This logical problem does not prevent the assurance of complicated systems, although the potentially significant increase in engineering time and risk implied by the need to consider everything that an increasingly large and complicated system might do is a major complication, but it does prevent the assurance of complex systems (or at least it prevents the 'near-perfect' level of assurance that traditional processes set as a target assumed to be achievable). Complexity fundamentally challenges the premise behind assurance.

3.8 Emergent Behaviour

The definition of a complex system given in Chapter 3.2.4 implicitly defined a complex behaviour as well. Emergent behaviour is a similar phenomenon, but slight variations in definition exist which can cause confusion.

For the purposes of this thesis, the following definitions for emergent phenomena (inclusive of but broader than emergent behaviour) and for emergence within a system have been adopted. It should be noted that these are original definitions from the author based on the definitions of the related concepts from earlier in this thesis.

> **Emergent Phenomena:** *Behaviours, properties or characteristics of a composition of elements that are not behaviours, properties or characteristics of the elements individually.*
>
> **Emergence:** *The interaction between entities within a system that results in the occurrence of some emergent phenomena by the system.*

The key point is that when a number of entities are combined together to make up a larger system, it is not merely the superposition of the behaviour of the elements that is observed as the behaviour of the combined system. Put simply, composition is not superposition. Some aspect of the combined system's behaviour changes from a straightforward superposition of elements due to the integration of or interaction between those elements.

A simple example can be thought of to illustrate the concept based on the autonomous vehicles example application area described previously in Chapter 2.5.1:

- A is designed to travel directly west.
- B is designed to travel directly east.
    - A and B together should pass by travelling opposite but in parallel (assuming they do not collide).
- However, as they near each other, they deviate to maintain separation distance:
    - The behaviour observed when A and B are interacting in reality is not the same as might have been expected from a simple superposition of expected behaviours of A and B.
    - This a simple instance of *behavioural emergence*.

This simple example will be seen later in this thesis to be exactly the kind of emergent behaviour that one may wish to see in synchronised robotics. It should also illustrate that emergent behaviour is not necessarily a bad thing – collision avoidance in the simple thought experiment above, and in the industrial case study that follows later in this thesis, is clearly a good thing. It is desirable for systems to adapt in this way. It is therefore desirable, if the systems in question feature some degree of autonomy, for these behaviours to be allowed, or even encouraged if an autonomous control mechanism can support this. In Chapter 8 later in this thesis, these sorts of adaptations are exactly the thing shown to be learnable, creating a positive and indeed vital instance of emergence that allows the system to continue operating as it should. The challenge is to ensure that if a system has the means to adapt its behaviour, it does so in a safe and appropriate manner.

The simple example also demonstrates an important characteristic of emergence. It is not a random occurrence. It is not a matter of stochasticity versus determinism in system theory. It is not in general something that can be dismissed as a long tail 'black swan' event. Entirely deterministic systems can display emergence if the interactions are sufficiently complex. It is a consequence of the system design, and of the control mechanisms applied to a system. As such, in any work concerned with novel methods for controlling and understanding the behaviour of complex multi-agent systems, there is little more important than properly understanding the boundaries, structures and control mechanisms one is using to design and manage the system, because it is in those choices that emergence ultimately arises, both positively and (if one is not careful) negatively.

In this chapter, the nature of emergent behaviour is briefly discussed, and how the simple definition being used within this thesis differs from some uses of the term within the wider literature. This is important to put the work that has been conducted towards this thesis in the proper context.

3.8.1 Examples of Emergent Phenomena

The simple example given above was a form of *behavioural emergence*, i.e. the observable patterns of behaviour change when additional elements are added. Emergence could be wider than that, encompassing a range of observable phenomena:

- Changes in non-functional properties of systems; for example, two systems determined to be safe and/or secure may not remain safe and/or secure when they interact.

- Run-time behaviours of modern software systems; for example, software programmes typically overflow their buffers after a certain run-time, but it can occur that n multiple parallel programmes overflow their buffers more or less often than n times the normal run-time.
- If an information system is considered to be the composition of databases, interfaces and, importantly, the processing algorithms themselves, then the output of an enquiry could be seen to be a property of the system. If for example an additional database is made available, additional information is processed and the outcome of the same enquiry is now different, this could be argued to be an emergent outcome (although looking ahead to the discussion in the next subsection about the difference between reducible and irreducible properties, this example would be seen by some to be a case of a hidden property not observable to the user, rather than pure emergence). Nonetheless, data/information fusion being a potential driver of behavioural emergence is a consideration in modern data-distributed system architectures, and is of particular concern when there is some degree of autonomy to act based on internally processed information.
- Examples in system health management arise when different combinations of system components cause different combinations of health status message to be sent, potentially changing the prioritisation of remedial actions and hence the status of the overall system.
- Ad hoc networking is prone to such phenomena; for example, the addition or removal of nodes causing routing to change (although the same caveat applies when looking ahead to the following subsection, that this would not meet the definition of emergence amongst all ways of thinking in the wider literature).

3.8.2 Reducible vs. Irreducible Emergence

It is important to note that in the wider literature there is sometimes considered to be a difference between conventional (sometimes called ‘weak’) emergence (characteristics seen to arise from the interaction of components) and ‘strong’ emergence, which distinguishes those characteristics which cannot be accounted for at the component level (in essence, distinguishing between reducible and irreducible collective behaviour). For the purposes of this thesis, it is satisfactory to consider emergent behaviour to be essentially the same

phenomenon as complex behaviours, and these labels will be used interchangeably. The example application areas referred to throughout this thesis demonstrate conventional or 'weakly' emergent behaviours under such a distinction, since they are reducible to the interactions between observable entities.

There is a view that when the unexpected phenomenon that has been observed can be deduced from a description of the elements and how they interact, then it is somehow a weaker form of emergence than if the cause of the phenomenon cannot easily be deduced.

It can be countered that 'strongly' emergent phenomena such as these must surely be reducible to something, even if it is not necessarily a simple or linear mapping between the properties of the elements and the whole, or if it is but there are some unobservable system elements or system interactions which have the effect of making an emergent phenomenon appear to irreducible, and therefore in some way a 'stronger' form of emergence.

Hitchins [35] explains how any particular observer only need consider an appropriate level or set of levels of a system. It follows that the emergence observed from a system is relative, and depends on the frame of reference of the observer. In the assurance sense, the manufacturer of small electronic chips might be concerned about unusual electrical discharges on certain configurations of motherboards but not on others, but this may be of no significance to the network administrator at all. They have different frames of reference on the system hierarchy, and consequently see different 'emergences' which may or may not have any impact on each other. Understanding the system is therefore about understanding the layers of the system and how they interact with each other.

The difference between weak and strong emergence would therefore seem to result from perceptions of what the system boundaries, purposes and structures are. Relating back to the system theoretic issues discussed earlier in this chapter, and the inherent subjectivity in what each of these things might be depending on the viewpoint of the beholder, then it may be argued that the difference between weak and strong emergence, or between reducible and irreducible properties of an integrated system, would arise in the logical structure of the system model and the functional form (mathematically speaking) of the system hierarchy.

3.8.3 The Engineering Significance of Emergence

Many engineering practices and processes are based on models of how the products and services offered will function or behave. Usually, these will be based on an understanding of

how the component parts function or behave, the process of systems engineering being the process of getting more out of the composition than one would have had out of the parts themselves. What becomes a problem is when the result of integrating a complex system produces not just the emergent properties that were wanted, expected and designed for, but unfortunate side effects that were not. These could undermine the effective of the system, and perhaps even make it unsafe.

Model-based approaches to understanding and analysing how systems behave may not guarantee that only desirable emergence can occur in a complex system, but clearly they may be able to help. The key is to have an approach that explicitly requires the engineer to consider the internal structures and interactions, and a mathematical framework for analysing and controlling these behaviours. Consequently, there is a need for modelling approaches that 'model in' complexity and hence emergence, to increase the likelihood of behaviours and characteristics being properly accounted for.

3.8.4 Relevance to the Example Application Areas

It is possible to imagine potential emergent phenomena that could arise from each of the example application areas in Chapter 2.5. Considering the fleet of autonomous taxis or delivery vehicles, an example was given previously of the potential for a number of vehicles to re-route to avoid a traffic jam, thereby causing a second traffic jam in another location by all trying to use the same avoiding route at the same time. One can also imagine vehicles re-routing autonomously from their planned journeys in order to avoid collisions with each other, as described in the thought experiment in this chapter. Combining these phenomena could lead to more complex patterns of behaviour. For example, a vehicle that makes use of a side street to avoid a collision or a traffic jam, may then find its expected opportunity to re-join its original route is now blocked by another traffic incident. The next best route to reach the original destination is now a longer route where the system had not originally intended to go nor would it have selected merely to avoid the first obstacle. This might look to an observer like the vehicle randomly going off course mid-journey, or be presumed to be a software bug or system fault, but it would not be. This would be a dynamic behaviour pattern derived from the same control rules (in this case, routing algorithms) as all the vehicle's previous movements.

This example serves to illustrate the importance of some of the abstract concepts discussed throughout this chapter. The unfamiliar observer might wrongly assume a fault or manually assume control of the vehicle or some other course of action, which would not be required if

there were confidence in the ability of the autonomous multi-agent system to adapt correctly, and that it would continue to act appropriately. For this, there needs to be the ability to collect the pertinent data from across the network about the state of the system (the system measures described previously) and the right information needs to be presented to the right observer at the right system level (for example, a passenger in the vehicle might be content just to be informed the vehicle was re-routing due to a traffic jam up ahead, whereas a system supervisor might need more detailed information to rule out a system fault).

This example illustrates many of the issues concerning observability, system measurement, and control mechanisms at different system layers within a complex multi-agent system. It demonstrates that there may be no real distinction to be made between a reducible (weak) or irreducible (strong) incidence of emergence, because different observers will have different access to system measures (i.e. what is straightforward to understand for the passenger might be harder to understand for a supervisor sitting miles away in a control room, or vice versa). All of this demonstrates that careful definition of the system or network structure, and of the states, actions and system measures, is vital to understanding how a system is behaving and to establishing confidence that any emergent effects can be controlled.

3.9 The Need to Understand Complex System Behaviours

Unfortunately, a robust solution to the acceptance challenge for systems exhibiting inherently complex or emergent behaviours does not currently exist. The discussion in this chapter has been to document the considerations and the design logic that an engineer or complex system designer will have to go through if building a multi-agent system. Choices around system definition and required functionality, internal system structure and organisation of system elements, mechanisms for monitoring, controlling and interpreting observed behaviour – all of these need to be considered by the designer but, as the discussion on emergent behaviours attempts to illustrate, this does not easily provide a general or absolute guarantee that the resulting system will not display undesirable behaviours under any circumstances.

To better understand how human observers can comprehend the behaviours they see displayed by complex systems, and how this knowledge can be used to better design and engineer future systems to ensure effective and predictable global behaviours, it is necessary to develop techniques to understand, quantify and analyse the behaviours of these systems as a whole,

which would unavoidably appear to include looking at the internal dynamics of such systems, and the modelling and control mechanisms that cause these dynamics.

The problem can be divided into three principal steps:

1. Modelling the structure and organisation of a system in a manner amenable to subsequent analysis, including the representation of system behaviour and current and future system states from available local or global information.
2. Identifying a suitable control methodology, which captures the operational purpose of the system and is capable of influencing the behaviour of the system towards its fulfilling of this purpose.
3. Representing the functional and non-functional requirements placed on the system as bounds on the behaviour of the system model, to assess whether the performance of the system will remain within these bounds, and if not to identify suitable mitigating actions to take to ensure it does so.

The original intention for this PhD was to research solutions to each of these steps and demonstrate the application of the chosen tools and methods through the whole process end to end. In reality this is too large a scope for a single PhD, so only the first stage of this work has been undertaken for this PhD and is covered within this thesis (steps 1 and 2 above, constructing a mathematical framework and showing how this can be utilised as a control mechanism to drive appropriate behaviour within a complex multi-agent system). However there will be pointers throughout to directions that the second stage of the work could take in future (addressing step 3 by proving the bounds of a system's behaviour based on these models), and further discussion of this aspect is contained in Annex C.

Figure 5 below provides a simple illustration of these principal steps, including the future stage of tackling the assurance problem. For clarity, the main topic of this thesis is the Modelling to Control stage (referred to hereafter as the MDP approach based on the use of Markov Decision Processes as the key building block – see Chapter 5) and what is left for future work is the Control to Assurance stage (described later as the IFS approach based on the proposal to investigate the application of Iterated Function Systems for this – see Annex C).

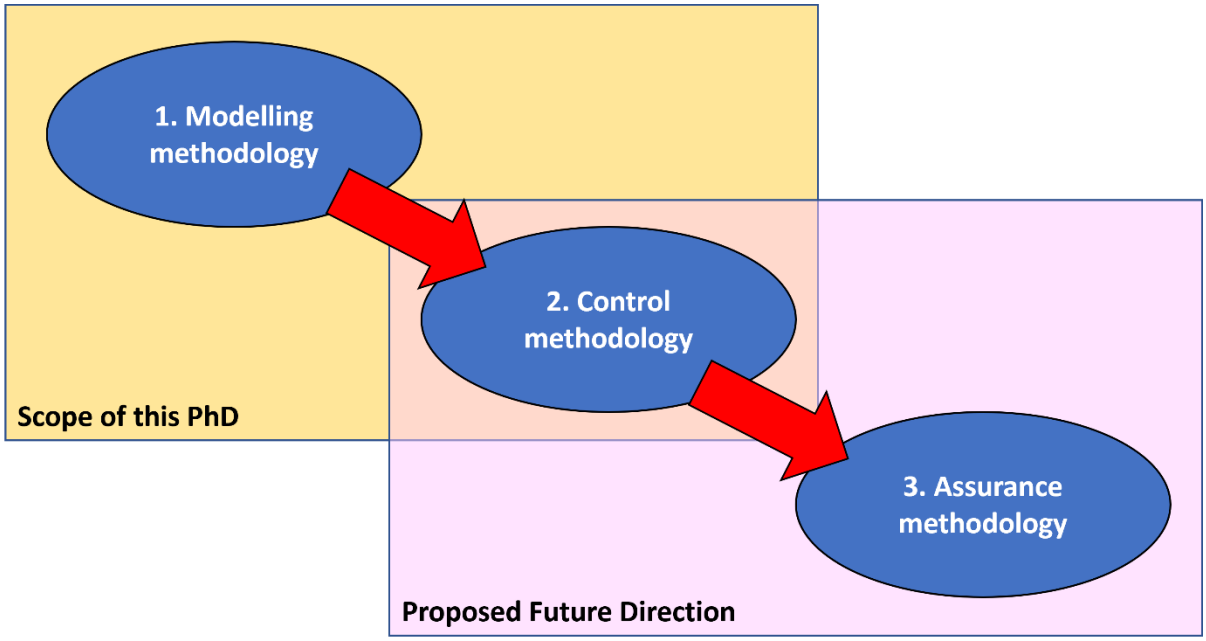

Figure 5 – The Principal Steps Proposed for Complex System Assurance

Modelling and control of complex multi-agent systems are discussed in the next two chapters of this thesis. Control is discussed in general terms in Chapter 4 as, in the context of multi-agent systems, mechanisms and design approaches for this is a major subject in its own right, and merits further discussion before proceeding with the detailed development. The mathematical modelling is then covered in Chapter 5 onwards.

Chapter 4

4 Controlling Behaviour in Multi-Agent Systems

4.1 Context

Multi-agent systems are an example of an emerging technology that presents a particularly difficult systems engineering challenge, given the propensity for such systems to produce inherently unpredictable or emergent behaviours. A real-world operational requirement that would motivate the application of multi-agent technology is likely to imply a relatively sophisticated system, with a large number of interconnected components and functions that all have some degree of local autonomy or decision making capability, some or all of which may be independent of any human operator.

The engineer responsible for designing such a system not only has to consider the expected impact on the whole system from what an individual component is expected to do, but also from all the things that it might conceivably choose to do, which can be a significantly harder problem, and do all this in a context where it may not be clear what the behaviour of the whole will be from observation and analysis of the behaviour of the parts, and vice versa.

How to control such a system will be a difficult challenge. The difficulty will partly arise from the number of agents, but also from their internal organisation and structure, and from the degree of interconnectedness between agents – properties which will depend on the system and the engineering challenge in question.

This chapter discusses the multi-agent control problem and the challenges associated with it. To motivate the discussion, control paradigms are described. Then follows further discussion on the particular challenges arising with multi-agent systems, where considerations about the predictability of observed behaviours from autonomous systems intersect with the potential for emergent behaviours arising from complex systems, creating a particularly difficult problem.

4.2 Control Paradigms

Consider as a starting point of discussion, the proposition that there are two distinct design approaches for the control of a complex multi-agent system, based on definitions from Deeks (the author) and Williams created during a previous attempt to categorise potential control solutions [25]. These represent the extremes of a spectrum of possible approaches rather than a binary choice between two specific solutions, but considering just these 'edge cases' is a useful route to highlight the general challenges. These two distinct approaches are:

> **Agent Sentinels:** *'Master agents' that monitor the behaviour of their peers and decide what actions the agents should take. The Sentinel is the agent ultimately responsible for ensuring behaviour is suitable and to correct any excesses.*
>
> **Distributed Control:** *Agents choose their own actions, based on their own (usually limited to local) knowledge. No single agent is ultimately in charge of behavioural assurance, relying on the system to either be self-correcting or, perhaps naively, to be incapable of straying outside of a range of acceptable behaviours.*

Both approaches have advantages and drawbacks. Clearly the sentinel approach would be expected to provide a greater degree of control over the individual agents, which is important for behavioural and safety assurance of any deployed systems, but too overbearing a master essentially reduces the complex system to a more conventional monolithic system, likely losing much of the functionality that was desired. Conversely, distributed control, while a popular area of research in recent years, has yet to really address the question of behavioural assurance, largely because of the high probability of the occurrence of complex or emergent behaviours that are difficult to predict and manage, making it difficult to quantify the potential limits to the exploitability of such technology.

Agents in a multi-agent system act based on their own local knowledge of the system and their local environment, and may be influenced by the behaviour of their local peers. At the time of the author's original attempt to categorise control algorithms [25], a popular area of research was to explore approaches to the control of multi-agent systems that involved agents 'brokering' with each other for some sort of utility or resource, the accumulation of which guides the agents' behaviour and thus the overall system behaviour.

To illustrate by way of example, consider the vehicular navigation application area from Chapter 2.5.1 once again. If the system-level agents controlling the individual vehicles have

no knowledge of the wider system configuration and current environment, it will be difficult for them to make optimal decisions. How is any one agent, knowing the workload it currently has and the location that it is in, supposed to plan future routes to optimise overall system performance? Does overall optimal system performance mean anything from the perspective of an agent without current knowledge about the rest of the system and its current state? From the perspective of an observer of the whole system, it might be optimal if a particular agent were to prioritise its current tasks in a particular order so that it travels to an otherwise sparsely covered part of the local area in order to relieve a growing backlog of waiting customers in that area, whilst transferring some or all of its existing assignments to another vehicle in the current location. From the perspective of the agent however, this might appear to be dumping a number of assigned jobs and travelling for no immediately obvious reason to a quiet area. The agent is unlikely to choose to do that based only its own local knowledge, but it might if there were greater information sharing about overall system state, and/or some mechanism for task brokerage between the agents to balance workload.

The example illustrates one of the problems with 'brokering' approaches, which is that the local interactions between individual agents cannot generally be assumed to 'scale up' to the global behaviour. One agent, or even a small group of agents, making decisions based on their own local area and/or sharing tasks between themselves are not guaranteed to produce the desired overall behaviours without guidance. This then leads back to a requirement for either an external controller (defeating the object of using autonomous agents), a coordinating or sentinel agent (which is the opposite of a distributed control solution), or looking for solutions that involve information sharing and resource arbitrage through as much of the network as possible (preferably globally, across the entire network).

These insights lead to the author's earlier conclusion (again see [25]) that behavioural assurance for systems lacking an external controller or sentinel essentially rely on a variation of the efficient market hypothesis, assuming that the information that flows between agents is sufficient to correctly identify in real-time the 'true' value associated with particular courses of action (sometimes called *informational efficiency*), thereby implying that with access to global information from across the network the system should be making the optimal decisions. The efficient market hypothesis itself is from economic theory concerning the pricing of assets on markets, and proposes that asset prices reflect all the information available to the markets, and therefore that the 'true' value of an asset is determined by allowing maximum access to all relevant data to all market participants and waiting for this 'true' value to emerge through

market absorption of this data, but the author previously saw an analogy between this process and the process of a local agent calculating its optimal course of action through informational efficiency with all the agents in its network (seeing the relative value of different actions an agent could select as analogous to the relative value of different assets a trader could buy or sell). There are two critically important assumptions made implicitly when relying on this hypothesis, namely that the agents themselves are not influencing the brokering process and that the so-named informational efficiency is indeed efficient (i.e. the sampling is statistically unbiased and sufficiently extensive). In essence, it is an argument that, with sufficiently plenty statistical sampling, a stochastic model will identify the correct solution, which applied to the multi-agent systems problem would mean the agents are provided with sufficient information to guide them towards efficient solutions without the need for any external direction. In relation to the example described above, the agents can only be expected to realise the true value of giving up their workload to local peers and relocating to a seemingly quieter area once they have access to full knowledge from across the network.

There are two major drawbacks with relying on informational efficiency to give confidence in a distributed control paradigm. The first is an immediate blocker for some applications, which is that it requires constant (or at least frequently available) and reliable communications across the network, so that agents can regularly receive new information. It does not necessarily mean every agent must talk to every other agent, but it does require every agent to talk to at least one other agent through which there is a route to obtain information from across the network (i.e. some agents could act as information conduits or hubs for others). As has been discussed already, communications may be impractical or even impossible in some application areas. Distributed control methods are not going to be a good option in such applications.

The second, more subtle drawback concerns whether the analogy with the efficient market hypothesis and reliance on informational efficiency is valid. There was already debate at the time of the author's earlier work about whether the hypothesis could be relied upon, given the financial crash of the late 2000s was still a recent memory at that time. It is not proposed to take this thesis into the arguments about financial risk modelling. To summarise, the concern is not strictly with the efficient market hypothesis itself, but with a naïve application that relies on the validity of the hypothesis without questioning these implicit assumptions under realistic circumstances. Put simply, referring to the two critical assumptions described above, markets might not have enough good information on which to make decisions, and the information they do have may take too long to process and for a broadly acceptable consensus value to emerge.

As such, relying on the markets to speedily or instantaneously have converged to a 'true' asset value is foolhardy. Applied to this context, even having access to all the information from across the network still does not mean agents will identify the most valuable actions to take, because the most pertinent and useful information might not be accessible to any agent in the system. Informational efficiency means that theoretically the agents will eventually converge on the best decisions based on what the collective knows, but that does not mean that this will be the optimal solution in the eyes of a beholder outside of the system itself, nor that it will even reach the decision that it does reach in a timely manner.

As a result, it is not obvious that a fully distributed control solution can be relied upon to converge to optimal behaviour solutions. This introduces significant doubt as to whether much of the earlier research that was prevalent around the time this PhD topic was conceived is applicable. Even in application areas with communication and coordination between agents, it is unclear that assurance of overall performance could be proven based on informational efficiency arguments, for the reasons given. Without reliable network-wide communication, distributed control methods are not promising, or at least not by themselves. Agents will need an ability to make local decisions without depending on other nodes in their network to cover for any periods of communications unavailability, so distributed methods might have a place as part of a suite of techniques, but are not something to base an assurance argument on.

In general, real world systems cannot be labelled as being controlled either by master or sentinel agents, or in a truly distributed manner. It is the premise of this work that these paradigms are the two ends of a spectrum of possible control solutions, a different balance of which will be appropriate for different systems. In order to evaluate and provide evidence of acceptable behaviour necessary to assure and subsequently exploit a system, a suitable hybrid approach might be necessary, which may feature some kind of supervising controller (providing the fallback of an intervention mechanism to correct excesses) whilst devolving as much of the decision making as is possible to local agents.

With reference to the example application areas referred to throughout this thesis, the requirement for a hybrid approach is already apparent. Referring again to Figure 1 through Figure 3, the system-level agents might be seen as acting more like master agents or sentinels for their individual systems. Were the optional system-of-systems coordination layer used, this might also be an appropriate control paradigm to adopt there. In the absence of this layer, a distributed approach to information sharing and arbitrage between system-level agents might be appropriate. In the absence of such communication however, either the system-level agents

start behaving as individual systems without regard for the rest of the system (likely leading to system inefficiency or ineffectiveness, and possible system failure) or an alternative is found – the synchronisation challenge that is the primary focus of this work.

4.3 Managing Complexity in Multi-Agent Systems

The characteristics of agent-based approaches that have made them an attractive enabling technology (devolved decision making to increase scalability and robustness to dynamic environments, to be adaptive and responsive to change, and to allow systems to more readily conduct multiple concurrent or synchronous operations) are the very characteristics that make such systems prone to complexity. The potency of agent-based technologies grows with increasing numbers of agents, but so does the complexity – the more agents there are in a system, the harder it becomes to identify which behaviours are resulting from which decisions, and to predict what subsequent actions might cause what subsequent behaviours, due to the influence the agents have on each other.

It is important to realise that these uncertainties in behaviour are not necessarily the consequence of fundamental stochasticity in the system. Such changes in observed behaviour could arise in a fully deterministic system that has simply grown too large and become too interconnected to effectively model all of the system dynamics. Mandating deterministic implementation, or identifying a mechanism to determine the probabilistic limits implied by a controller definition, may eliminate one source of uncertainty in system behaviour, but would not solve the underlying problem. The challenge is to consider how the design choices and control mechanisms themselves impact on behaviours, and how this knowledge might be applied to understand and better manage complex systems.

Several approaches to implementing multi-agent systems with a high degree of assurance have been considered, all of which attempt to limit the complexity of the system so that it is controllable.

Three primary approaches to managing the complexity within multi-agent systems are:

- Constraining the size and interconnectedness of the system to limit the possibility of complex behaviours occurring.
- Putting an independent mechanism in place to manage any emergent behaviour if and when it occurs.

- Designing a system in such a way that the system ‘gets through’ any disruption that occurs (i.e. acceptable trend behaviours).

A brief overview of the idea behind each of these approaches follows. They are practical solutions to some problems, but each approach has limitations and none scale to solve the underlying problem of managing complexity in a general sense.

4.3.1 Constrained Design

One approach to managing complexity is to limit the scale of system designs, so that the number of decisions that a manageable number of agents have to make is sufficiently small and take place in a well-understood environment and operational context. The interconnectedness of the agents is low, meaning that the numbers of agents that are significantly influenced by the behaviour of others is low, by whatever metric is suitable in context.

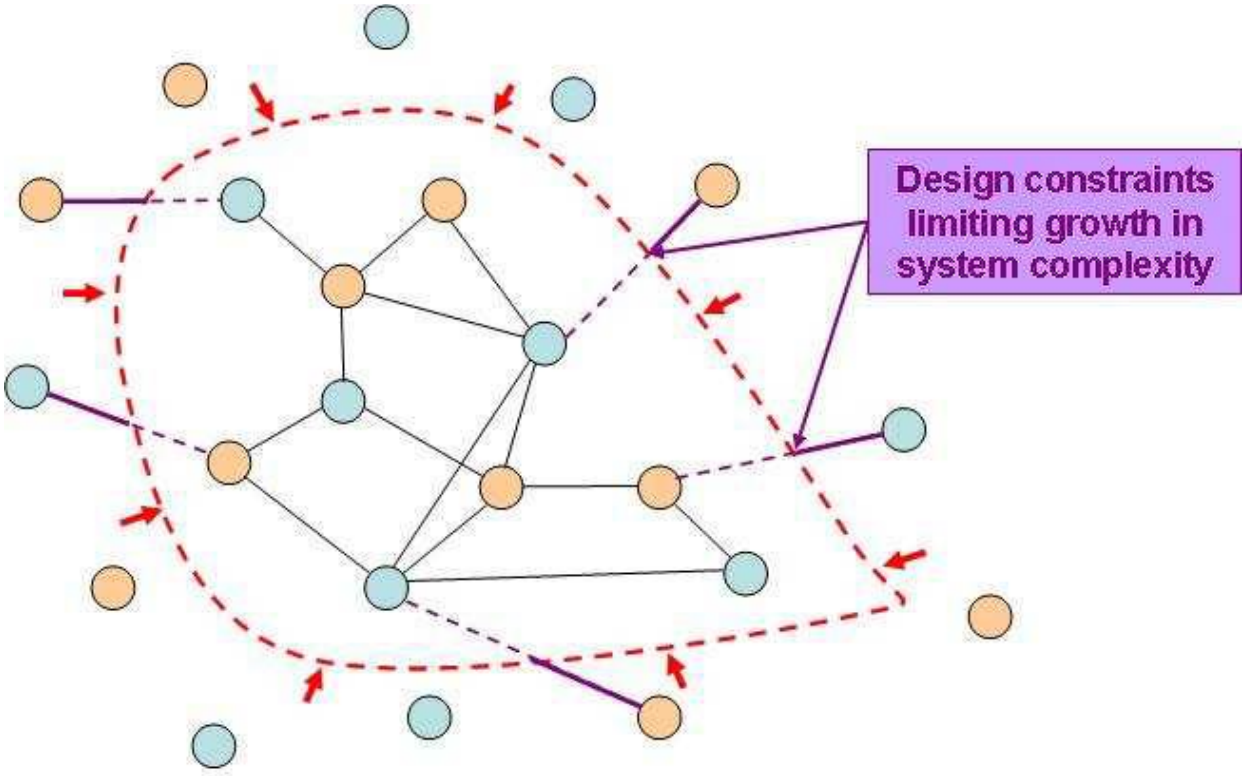

Figure 6 – Constraining Growth in System Complexity Through Design

The idea is to design the system in such a way that standard processes can be employed, to represent just the most necessary functionality of a system in a sufficiently tractable way. The

problem is that, as with any model, if it is simplified too much it is no longer representative of whatever it is that it is meant to represent. Having a small number of agents with a small number of connections is only meaningful as a model if those are indeed the characteristics of the problem at hand. This implies that there is a limit as to how much the design of a system can be constrained whilst retaining the properties that made an agent-based approach desirable in the first place.

If a tightly constrained model is sufficient as a representation of the necessary functionality of a system, then there is a further question as to whether an agent-based system paradigm was necessary to begin with. There would therefore seem to be a relatively narrow window for system concepts that genuinely benefit from adopting an agent-based approach that are still sufficiently constrained for direct analysis to be practicable.

4.3.2 Observers and Sentinels

An alternative mechanism for managing complexity in a multi-agent system is to have human or agent 'monitors' or 'sentinels' to either prescribe the limits of what agents can do or to intervene when appropriate. Prescribing too tightly what individual agents can and cannot do is not a true solution, for similar reasons as above, so if there is any possibility of unanticipated behaviours then some mechanism to respond or adapt or make changes in response to such behaviours should be considered.

The obvious answer is to have a human operator monitoring the system with the ability to overrule the system if considered necessary. There are two limitations to this solution. One is that it would appear to undermine the point in decentralising functionality to autonomous agents if they require supervision, but, more problematically, it is less and less feasible to rely on human intervention to ensure acceptable behaviour when the fundamental challenge is the increasing complexity of behaviour of systems of increasingly many agents.

A similar approach is to have so-called 'sentinel agents' monitoring and managing the other agents. Logically, this is the same as having an in-built 'decision filtering' of some kind to override unacceptable decisions, which is an alternative approach the author has seen proposed. The problem with such an approach is that it is recursive. Somewhere in the system, a decision making agent must approve or amend decisions made by other agents, or to make them on their behalf, and this is in itself a decision whose consequences could impact the network of agents in unpredictable ways. Thus a new problem is created in determining how to assure the

decision making of the sentinel agent or decision filter – a 'Who watches the watcher?' problem. Ultimately, this is moving the problem around to avoid having to solve it, and does not appear to move one closer to an approach to assuring the system as a whole.

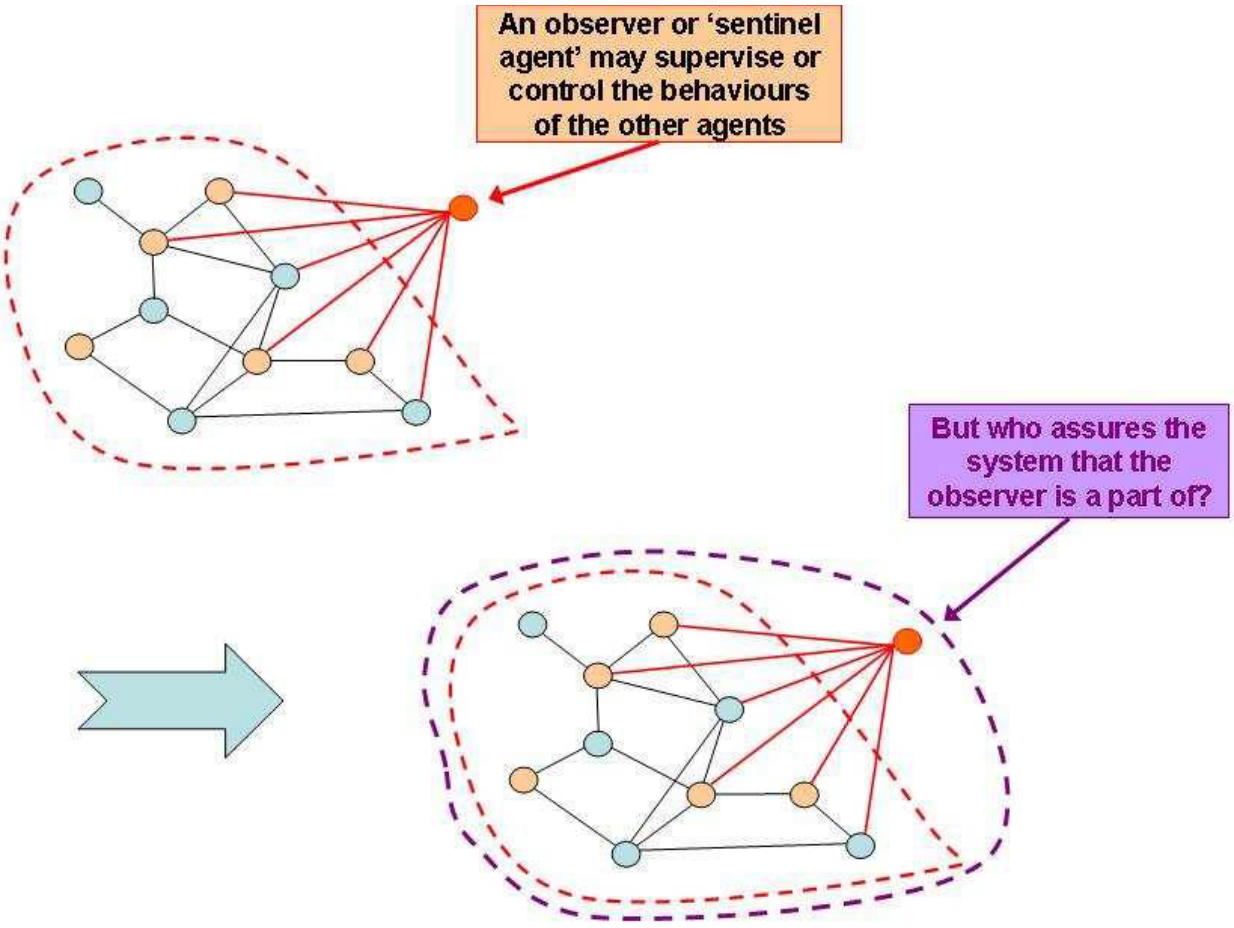

Figure 7 – The Recursive Assurance Problem of Sentinel Agents

4.3.3 Game-Theoretic Equilibrium

An alternative approach to managing the complexity inherent in multi-agent systems is to rely on results from the field of game theory. The idea is to model the interactions between agents as transactions, with the agents as brokers for utility of some kind; the overall behaviour of the system can then be described as a trend behaviour arising from a suitable game-theoretic model, such as Nash equilibrium.

There are two problems with such an approach. Firstly, requiring the agents to barter with each other would appear to constrain the types of multi-agent systems to those instantiations where

the agents are cognisant of each other's existence, which is not necessarily true in many cases of decentralised systems. Systems with processes conducted concurrently that are fundamentally linked but where bartering between subsystems is not possible, for example because there is no communication link between components, would not appear to be controllable by a game-theoretic approach.

Secondly, and the more serious problem from the point of view of assurance, is that trend solutions from game-theoretic models provide no guarantees as to the behaviour of a system throughout its run-time. Just because a system will eventually settle down into a predictable pattern does not mean it will follow this pattern throughout, leaving open the risk of some undesired behaviours along the way. Learning systems, for example, might be prone to digress from what will later be determined to be their optimal strategy during their early period of operation, whilst they are still learning this optimal strategy. Any system, learning or otherwise, that is capable of responding to environmental perturbation could make a deviation from trend behaviour in response to particular stimuli – the trend behaviour may reveal what pattern or configuration the system will ultimately return to, but nothing about the acceptability of any digressions from this pattern along the way.

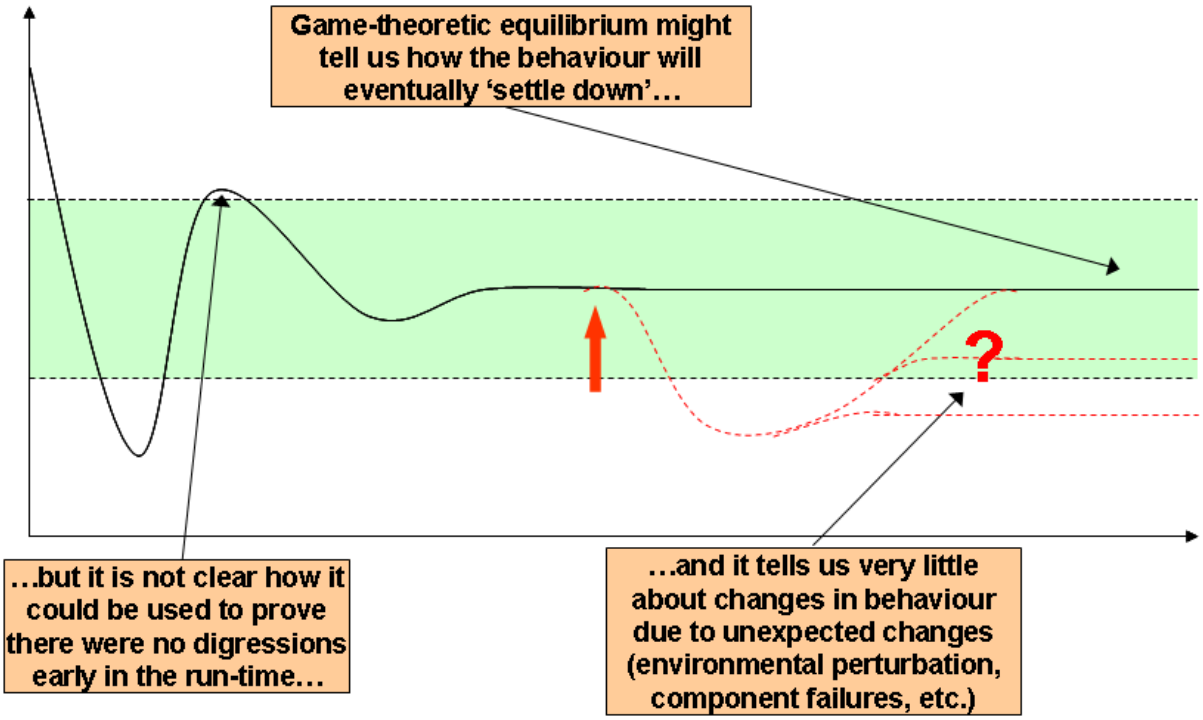

Figure 8 – The Problem with Game-Theoretic Equilibrium and Assurance

It is believed that the game-theoretic approach has similarities to the earlier work on which this PhD builds, looking at stochastic (Markovian) approaches to modelling system behaviour, so this is not an entirely inconsistent modelling paradigm. Furthermore, game-theoretic equilibrium (or other similar approaches to representing stable trend behaviours, of which many can be found in the domain of modern control theory) may provide a mechanism for demonstrating the *stability* of system behaviours within suitable bounds. However, at this time, it is not obvious that a game-theoretic approach can be used to demonstrate that a system is suitably bounded, without which further analysis is premature. Further consideration is required before such an approach could be applied on real systems.

4.4 The Assurance Challenge for Complex and Multi-Agent Systems

The assurance process is, essentially, the systematic detailing of what each component in a system is, what it does, and whether this complies with the functional and non-functional requirements placed on the system; more specifically, it is the generation of evidence for the assessment and demonstration of this compliance. Numerous techniques exist in conventional safety case analysis, such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA), which attempt to identify the likelihood and severity of occurrence and, if possible, mitigating actions.

These and other such methodologies rely implicitly on the predictability of a system's behaviour under the circumstances the system is designed for and expected to operate in. However, as a system becomes more complicated, the assurance process becomes more difficult, as components start to impact on each other's behaviour in ways that become harder to determine.

Approaching the assessment from the perspective of 'What is each component and what will it do?' is unhelpful as a basis for understanding system behaviour when, by definition, the behaviour of the whole system is different from the behaviour of its component parts. Analysing the behaviour of the whole system becomes an increasingly difficult task.

4.4.1 Faults and Failures

The concepts of *faults* and *failures* have particular meanings in the context of the behavioural assurance of systems, but a subtle difference in the relationship between them in the case of complex systems illustrates the fundamental challenge.

A *fault* occurs when a component (hardware, software, or otherwise) operates incorrectly, or does not operate at all. A fault in a component may or may not (depending on the type and severity of fault) cause a system *failure*, which occurs when a particular combination of faults prevents a system requirement from being met.

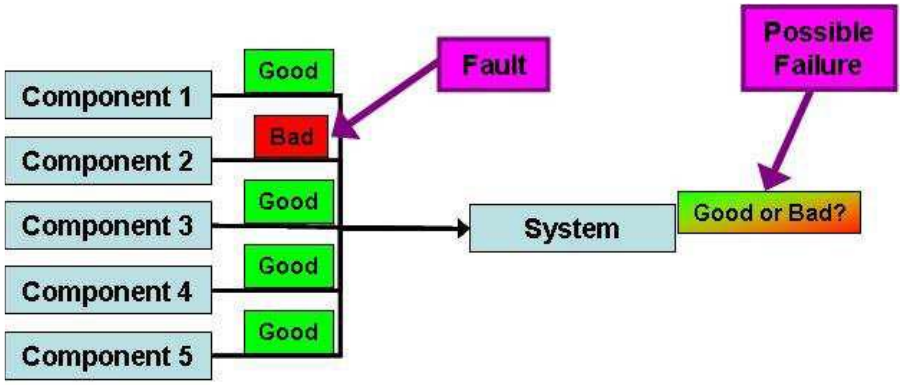

Figure 9 – Normal Operation – Component Fault and Possible System Failure

Systems will typically be tolerant of faults up to a certain threshold. In simplistic terms, a safety-critical system will have one or more system requirements which specify that threshold, in addition to requirements which specify the behaviour under 'normal' conditions. Assurance processes such as FTA and FMEA analyse how the system as a whole is expected to behave ordinarily, and how it might behave under combinations of faults, to identify those combinations of faults which might cause failures, and to understand the consequences of failures. With complex systems, the challenge is that there is no longer a direct causal link between component faults and system failures. Every component can be functioning as it is supposed to, but if the system is complex then the overall system behaviour could be different from that desired or expected. In other words, a system could suffer a failure (systems displaying behaviours contrary to requirements) despite not having suffered any faults at the component layer.

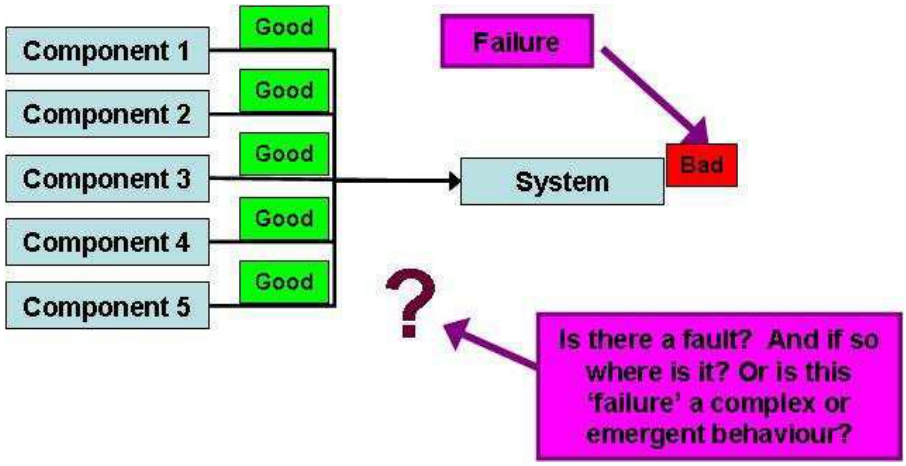

Figure 10 – Complex System Failure Despite Compliance of Individual Components

In its simplest form, the complex assurance challenge is to deal with a scenario that the 'normal' operating model does not allow for, either demonstrating that such a thing cannot happen, or understanding when it might and identifying appropriate mitigating actions.

That system behaviours do not follow logically from anticipated component behaviours fundamentally challenges the assumptions on which traditional assurance is based, which makes it difficult to escape the conclusion that the techniques themselves need to be revisited. It is perhaps unfair to entirely dismiss established assurance practices as inapplicable to complex systems, but the traditional approach is clearly not intended for and is fundamentally ill-suited to dealing with these sorts of challenges, and will require some evolution in thinking.

4.4.2 Boundedness and Stability

It is not clear how to demonstrate the acceptability of the behaviour of a system when it is not clear what that behaviour will even be. Doing so must involve understanding how the compositions of behaviours of the components can cause unusual or unexpected behaviours from the whole system. In the case of complex multi-agent systems, much of the complexity arises from the different decision making agents in the system taking courses of actions which for that particular agent appear entirely logical but can cause wider consequences through the knock-on effects on other agents (examples of which have been presented in earlier chapters). Understanding these dynamics leads towards the domain of modern control theory, a domain whose purpose is to construct more sophisticated models and approaches to the decision making and control process much broader than the traditional control systems of the past.

Two particular concepts relevant to this problem are *boundedness* and *stability*. As will become apparent later, the proposed method for conducting assurance involves identifying the limits of possible behaviour, and in some cases the long-term behavioural trend.

- *Boundedness* means that there is a quantifiable outer limit to the system's state space, beyond which its design and control specification will not allow it to go.
- *Stability* means that the behaviour converges, or that the span of the state space reachable by the complex system decreases over time. This might mean the system behaviour converges to a fixed point, but it more generally means the system 'closes in' on a stable configuration, such as a periodic orbit.
- Stability is a stronger criterion that boundedness, in that it can be proven relatively easily from the formal mathematical definitions that a stable system is necessarily bounded, but that a bounded system need not be stable (for example, entirely random movements within a fixed domain).

The difficult part, and unfortunately the most necessary part from the point of view of assurance, is determining whether a system's behaviour is bounded at all, and if so is it suitably bounded to ensure that the behaviours are within acceptable limits. If it can be shown that a system is suitably bounded, then in a general sense this could be good evidence for assurance, i.e. evidence that the system stays within suitable limitations. What the system does within these limitations may not matter, as long as those limitations are themselves acceptable.

Stricter requirements for particular types of behaviour may occur, in which case stability will also need to be shown (behaviour either fixed to a specified pattern of behaviour or, realistically, closely tracking a specified pattern of behaviour and returning to it in the event of perturbation). The focus herein is on the looser boundedness criterion; that is, how one might ascertain, one way or the other, whether a given control specification leads to suitably bounded behaviour. Some discussion of the wider problem, including assessment of boundedness and stability, is contained in Annex C.

Note that assessments of boundedness and stability will only be valid so long as the system design, and in particular the controller design, remain the same. One particular benefit of the method considered in this thesis is that updates to the control policy (in other words, changing the action to be taken in particular circumstances) will not affect the overall assessment of boundedness, due to the generation and assessment of the full span of possible behaviours. However, this would not make this approach immediately transferrable to more advanced

cognitive or adaptive systems, with the ability to change more than just the control policy but also the cognitive structure and/or the architecture of a system, which would invalidate predictions of what the reachable span of behaviours from such a system would be (hence the earlier exclusion stated in Chapter 3.2.6 that artificially intelligent systems are out of scope).

4.5 Decision Making and Control

Whenever a system component chooses the next action to take, it is making a decision based on its current state and its overall goal (or goals). This decision has to be made despite inherent uncertainty in both the environment and, potentially, in the system itself – specifically, with the outcomes of actions not known in advance and/or with the current state information not totally accurate, perhaps because of sensor limitations or an incomplete world model. The vision driving this PhD is that the mathematical apparatus for decision making will transfer to the system control problem through the creation of a suitable mathematical model for describing structures of complex networks of agents.

This section explores the analogy between decision making and control that motivates this approach, and highlights some important choices that need to be made when deciding how to solve such a problem.

4.5.1 Levels of Control

By definition, a system having a degree of autonomy implies that direct operator input is small or non-existent – at issue is how much indirect control the operator has as a result of their programming choices and the degree of autonomy granted to the autonomous system from the outset. Philosophically, one might ask how intelligent the system is, or to what extent the system is in control of its own destiny.

Broadly speaking, one can identify three levels of control, ranked in order of the degree of autonomy the system has (i.e. to what extent the system can choose actions under its own authority). This will correspond directly to the degree of flexibility the system has in the event of changes in its situation:

a) The system has an explicit sequence of operations specified.

b) The system has a decision making process with a fixed policy (so an action is specified for each situation the system is expected to encounter).

c) The system has a decision making process with an adaptive policy (so an action is chosen based on the situations that are actually encountered).

Approach (a) is clearly unable to cope with any but the most minor changes to the environment. This is the level of autonomy one might find in a conventional industrial robot, for example in a simple assembly line robot that is not capable of being responsive to the environment or to its peers. Clearly this sort of approach is far too limited to be of any great use in the future applications proposed.

Approach (b) is the level of control where, for each situation in which the system can reasonably be expected to be find itself, an 'optimal' action is defined which should lead to the eventual accomplishment of the system's objective or mission goal. However, this policy is tuned to a particular model of the environment and so any disturbance may lead to a loss of optimality or, in extreme cases, failure to achieve the goal. This remains one of the single biggest limitations to the operation of autonomous systems – if the system only knows how to respond to particular scenarios, then it will likely struggle when faced with something unexpected, potentially even when the situation appears only slightly different from that which the system has been programmed for.

Approach (c) attempts to overcome the limitations of approach (b) by allowing the system to make a constrained modification to its policy in the face of changing circumstances. This makes approach (c) attractive at all levels of decision making, from higher-level strategy down to the detailed control of system components. It is proposed that the ability to change tactics following changes to the local operating environment, mission goals or system state, whether planned or unplanned, should be right at the heart of the decision making process.

However, it is important not to forget the more conventional control problem, for which one might reasonably expect that approach (b) is an appropriate level of control for a system. Approach (c) might then describe an extension required for effective operational decision making. It is further proposed that it is not a clear distinction that approach (b) is for the control problem and approach (c) is for operational decision making. Approach (b) might be suitable for operational decision making controllers in relatively static, well-understood or controlled environments. Likewise approach (c) may be necessary for systems that are required to continue operating even after the system itself has undergone a change significant enough to affect the control problem (such as continuing to operate despite damage). The exact

breakdown of the level of control required at each layer in the hierarchy will depend entirely on the nature and purpose of the system in question.

The challenge is therefore to balance both of these perspectives throughout subsequent developments. It is noted that the principle objection to approach (c) is the lack of a sound mathematical basis from which to construct a safety case, or to predict performance with any degree of certainty when the operational policy is variable by design. The novel certification ideas from the precursor activity [23], building on initial work by Williams [68], were intended for approach (b). Adapting these techniques to provide guarantees about the boundedness of system behaviour for approach (c) would be an extension of these ideas, requiring further research. Further thoughts on this topic are contained within Annex C to this thesis.

4.5.2 A Graphical Representation of the Levels of Control

The three different levels of control can be represented graphically. The following figures represent the three approaches described above, illustrating how the degree of autonomy granted to the decision making controller relates to the problem of predictability. In each figure, a series of actions chosen are represented as a series of blue arrows, and the green strip across the graph illustrates the range of states that can be reached when following this control policy, labelled as the environmental boundary. In the latter two cases, dashed red arrows indicate where other actions that could have been selected. Note that the three approaches to control affect the range of states the system can be allowed to reach; the adaptive approach (c) in particular is a rather more complicated problem.

Figure 11 illustrates approach (a), where action selection is explicitly determined. In this case, the system might be better described as automated rather than autonomous (the latter implying the ability to choose between different actions), and the range of states the system can find itself in is entirely predictable. In fact, it is a fixed set of reachable states (barring say unanticipated mechanical failure). With such an approach to control and consequently a narrow environmental boundary, the system is inflexible and it is limited to fixed and pre-planned tasks.

With reference to the example application areas from earlier in this thesis, this level of control might be equivalent to a networked camera which had little or no scope to reorient itself or to adjust its focal length or filters. It either cannot do so, or has limited options in line with a pre-programmed rule set. In either case, the system is not that ‘smart’ and will not have the

flexibility or adaptability required for more advanced and reactive applications. Alternatively, this level of control might correspond to an unmanned vehicle which follows the same specified route on a regular cycle. It would be operating automatically, but with little or no flexibility or adaptability. This level of control would be of limited use for the sorts of autonomous operations envisaged in the earlier discussion.

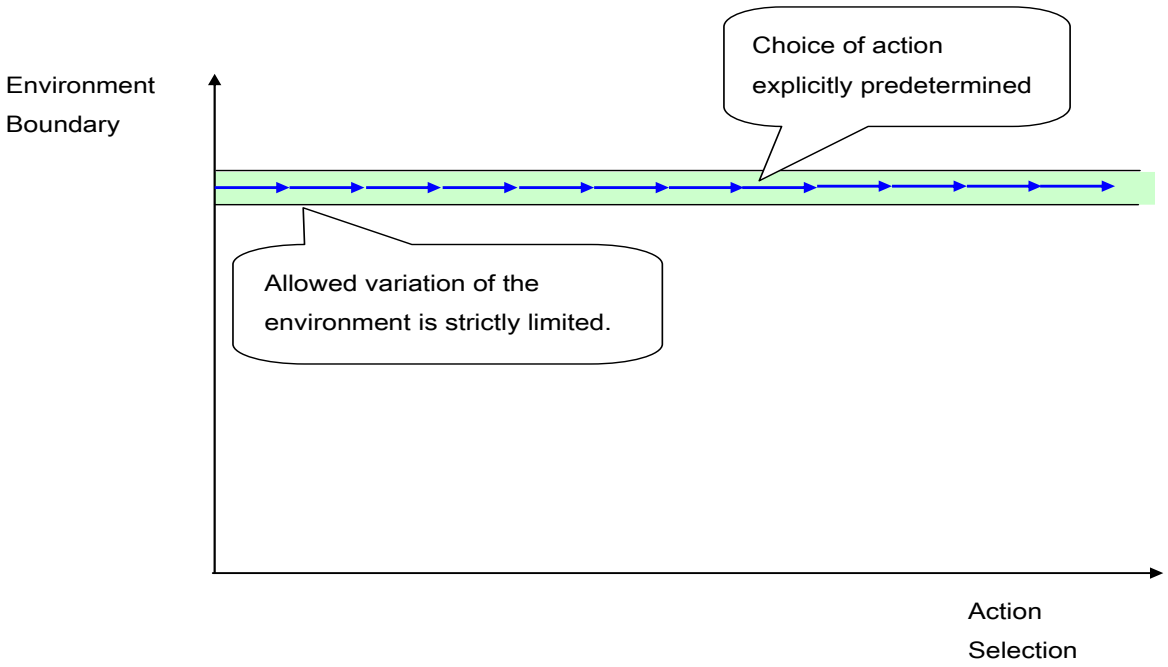

Figure 11 – Approach (a): Explicit Action Selection

Figure 12 illustrates approach (b). At each decision point the range of reachable states is larger as there are more actions available for the system to choose between, and thus more possible outcomes. Nevertheless, a fixed policy still constrains the range of reachable states for the duration of the operation.

With reference to the same example application areas, this level of control represents the smart camera or the autonomous vehicle now being granted some ability to adapt locally, but based on a fixed policy. In the case of the smart camera, this might manifest itself as having a set of strategies whereby when 'Object X' is within 'Region A' these filters should be applied and this range of camera motions are permitted, but when 'Object X' is instead within 'Region B' these alternative filters should be applied and the range of camera motion is restricted. There is therefore some local autonomy, but within a fixed policy. Alternatively, with autonomous

vehicles, this level of control might correspond to having a set of acceptable routes pre-planned, with the instruction to choose the one from this pre-approved list that, at the time of selection, has the lowest traffic density. The decision as to which one to use on any particular occasion is determined locally at the time, but it will still be predictable based on set rules.

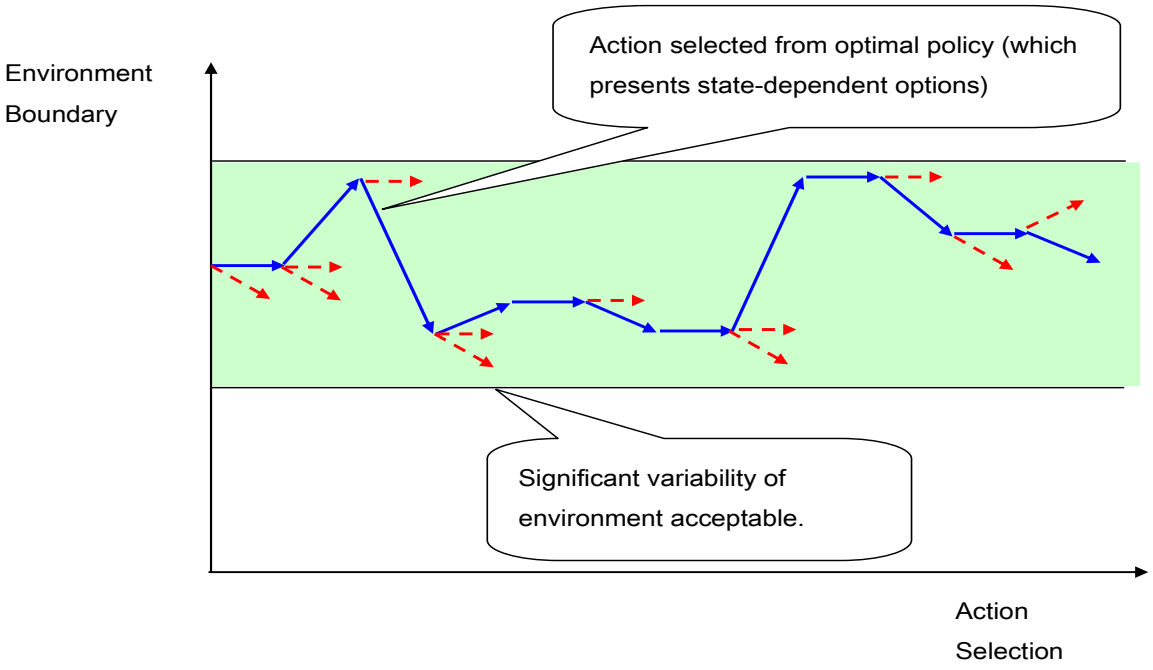

Figure 12 – Approach (b): Optimal Action Selection from Policy within Fixed Range

Figure 13 illustrates approach (c). Here, as the operating environment changes (or indeed as the system itself changes), the range of reachable states varies. The overall limits on the system behaviour (the environmental boundaries) remain, but what is currently reachable within this boundary varies. If it were acceptable to be in any state within the overall boundaries, then approach (b) might be sufficient. What Figure 13 illustrates is that approach (c) can be utilised when system behaviour needs to be refined to accommodate any uncontrolled external factors – i.e. when the system needs to be constrained to a range of states more limited than the overall boundaries, and this allowable range of states varies as the situation develops.

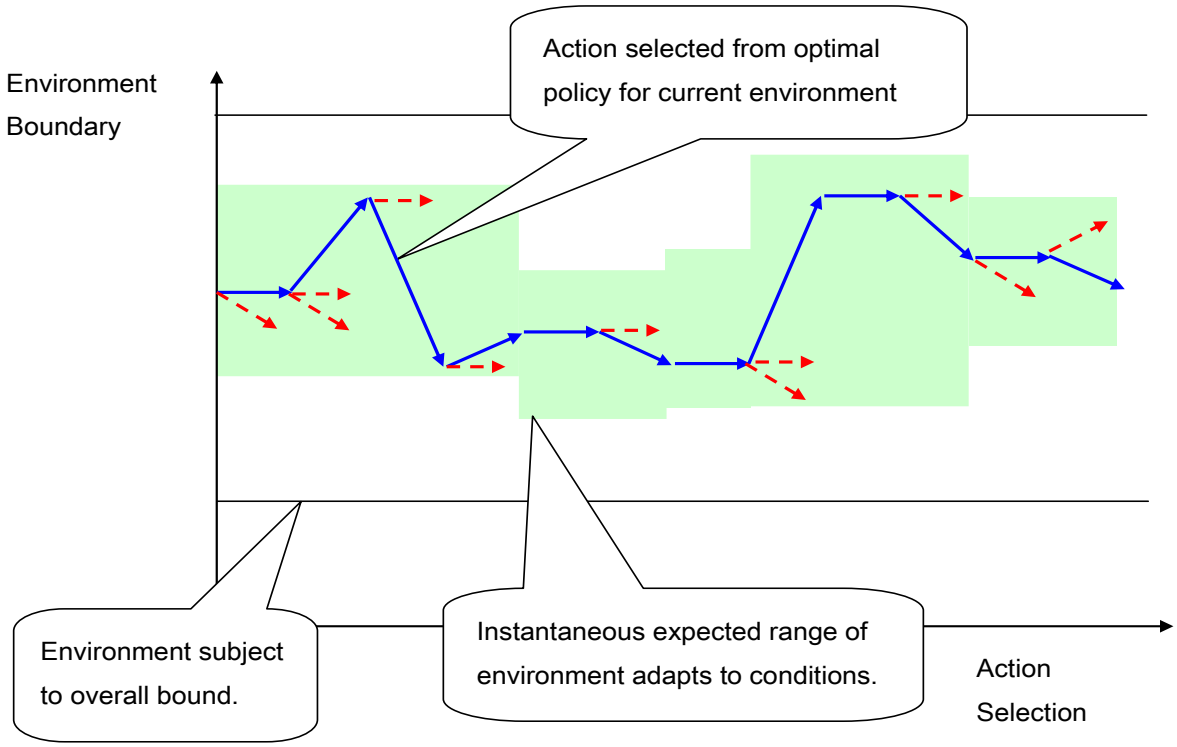

Figure 13 – Approach (c): Optimal Action Selection from Policy within Variable Range

As an example, consider an autonomous ground vehicle crossing a bridge. The overall boundary is clearly fixed by the edges of the bridge. However, before crossing the vehicle may not know what obstacles or other moving vehicles may be on the bridge, which will have to be determined as it crosses. Thus the range of states that really are safe is more limited than the overall boundaries, and varies according to what the vehicle encounters as it crosses.

Note that the reachable range of states may vary continuously, but so long as decisions are made at discrete time steps one need only consider the situation at those particular time steps (the decision points). In the future, continuous controllers may be desirable and the continuous variation in the range of permissible states will be important. Note also that there are potential complications arising from the fact that earlier action selections directly affect the range of permissible states at future times. Some form of look-ahead in the decision process is important, and this feature will be incorporated in the solution techniques discussed later in this thesis.

Chapter 5

5 A Mathematical Framework for Modelling the Behaviour of Complex Multi-Agent Systems

5.1 Context

This chapter develops a framework for constructing scalable mathematical models of complex multi-agent systems, with the flexibility to be adapted to fit a multitude of different organisational structures of systems, and which captures the fact that for each agent (represented as a node in a network) there needs to be a local decision making process captured mathematically. Drawing on techniques from the machine learning domain and applying suitable simulation tools then provides the necessary toolset to be able to observe, analyse and begin to understand the complex behaviours of multi-agent systems. It shall become apparent that the mathematical tools in question also reveal an approach to making multi-agent system performance more robust to unplanned emergent behaviours, with some degree of internally triggered adaptivity to external perturbation possible.

This specific modelling technique developed makes use the Markov Decision Process (MDP), as originally proposed in the precursor work for single autonomous system applications [22],[23],[24], building on concepts proposed by Cade [15] and Williams [68]. This chapter will begin by discussing the key modelling assumptions, before introducing MDPs, revisiting the formulation from the precursor work, but then substantially extending it to demonstrate how it may be adapted to the more challenging domain of complex multi-agent systems.

5.2 Modelling Assumptions

The starting point is to decompose a complex system into subsystems each with a simpler and more recognisable purpose, which may be formulated as goals or objectives, and then to construct a suitable mathematical model for this decomposed system model that is amenable to analysis.

It is assumed that a complex multi-agent system is a large (but finite) collection of subsystems where:

- Homogeneity of elements is not assumed; entities need not be physically similar nor do they need to have common functions or goals.
- 'Perfect' communications and shared knowledge between elements is not assumed.
- 'Self-knowledge' at each element may be imperfect.

It is not realistic for decisions to be made at one point in the system and be communicated instantaneously to all other subsystems to respond simultaneously. Therefore, it is assumed throughout that there is some degree of decentralised or local control:

- The system, and each of its decision making subsystems, is autonomous in that it has some degree of freedom of choice to select its own actions based on its currently observed state, independently (or at least semi-independently) of any human operator; each subsystem is therefore a part of the other subsystems' environments, and the reasons for the actions of one subsystem may not be apparent or comprehendible to another subsystem, for whom it is simply another source of environment perturbation.
- Each individual element could have an independent decision making capability; more generally however, there may be variable granularity of decision making authority between different subsets of elements in different contexts.
- Any cooperation between groups of elements, planned for or otherwise, is on a per-task basis and is not a global assumption; any groupings or affiliations of elements within the system should not, in general, be seen as fixed groupings or affiliations but instead be regarded as transient arrangements.

It is also assumed that the control model for each system element can be formalised as goal-driven decision process, with a finite dimensional state space. Even though the 'true' system

state may be of much greater dimensionality, the state space representations recognised by each individual element can be modelled as a finite, although possibly non-linear, projection.

5.3 Functional Architecture

Recall from Chapter 4.2 that there are two distinct design paradigms for the control of a complex multi-agent system. These can be summarised for present purposes as:

> **Centralised Control:** *A single decision point or 'master agent' monitors the behaviour of other agents and decides what actions should be taken. This 'master agent' is ultimately responsible for ensuring behaviour is appropriate and to correct any excesses.*
>
> **Decentralised or Distributed Control:** *Agents choose their own actions, based on their own (usually limited to local) knowledge. No single agent is ultimately in charge of behavioural assurance, relying on the system to either be self-correcting or, perhaps naively, to be incapable of straying outside of a range of acceptable behaviours.*

Both approaches have advantages and drawbacks, which were discussed in the previous chapter. In practical applications, a hybrid approach may be the most robust, for the reasons discussed previously. Any modelling framework should therefore be flexible and be able to accommodate both approaches, but also address the difficult case of synchronised autonomy identified previously as a gap that was not satisfactorily addressed by either control paradigm.

The approach investigated herein is to introduce a mathematical model of the interaction between agents. This model is derived from the specific decomposition of the complex system into its component parts. Devolving the decision-making into separate elements necessarily also splits the state space, the information upon which decisions are based, into separate segments, one for each decision element. The principles involved in this aspect of complex multi-agent system design will be discussed later.

A working model of the decision-making process proposed is outlined in Figure 14. The decisions taken by the system are a sequence of actions, indicated by the multi-way switch in Figure 14.

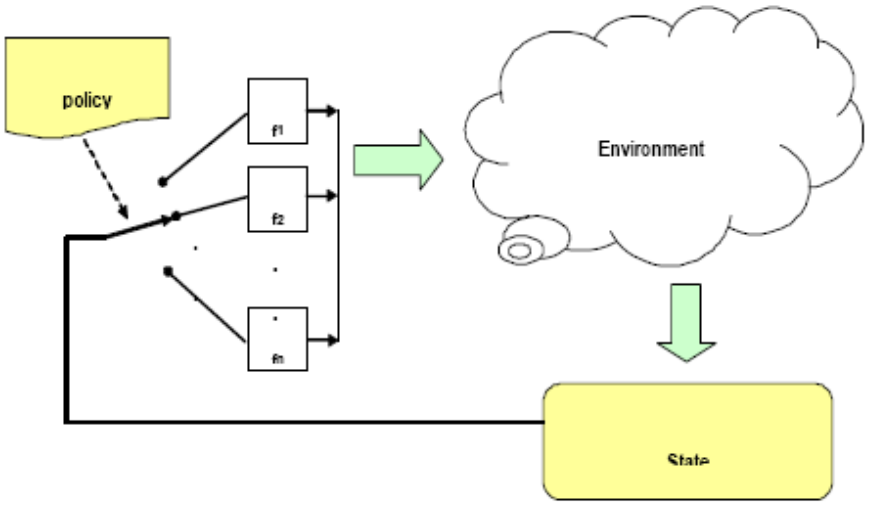

Figure 14 – The Decision Making Cycle

.

Given the current state of the system, the policy determines an action to take. This action causes the system to move within its environment (either a physical move or some other change in a system state variable) which results in the next system state (which is not necessarily a different state). This new state then informs the next action selection and, depending on the system and application in question, the cycle repeats either indefinitely or until a terminal state is reached, after which no further state transitions can occur.

The decision-making process just described is a generalised form of the Markov Decision Process (MDP) which is described more fully in Puterman [56] and more recently by Sutton and Barto [59]. This decision-making architecture can be applied to the entire spectrum of functional architectures from centralised 'sentinel' control to a fully distributed architecture.

The segmentation of the state space and the decision-control process into smaller subspaces and control problems generalises the basic control problem to any appropriate degree of granularity in the actual system architecture. The focus within this thesis is on the specific consideration of how to model and account for the interaction between agents in a multi-agent system and the affect this has on the overall system behaviour. The developments across this chapter and Chapter 6 will demonstrate how this unified approach to architectural design enables the construction of tractable models for multi-agent system behaviours.

5.4 Structural Architecture

The structural architecture of a complex system can be modelled by considering the principles which govern how it might be decomposed. Three criteria are identified which might be used to determine this:

- The *environmental scope* of each decision. The environment is partitioned, with an agent designated to act in each of these partitions. This is a form of concurrency; each agent is responsible for some part of the environment and operates on it to fulfil the system purpose, or goals. Typically, but not necessarily, each agent would have a similar control policy and capability, although its actions would be determined by the local environmental conditions.
- The *functional scope* of each decision. The functions of the system are partitioned, typically using dependency as an ordering relation. If, for example, the partitioning results in three agents, *a*, *b* and *c*, where *a* -≻ *b* and *b* -≻ *c* (to be read as *a* depends on *b* and *b* depends on *c*, i.e. information flows from *b* to *a* and from *c* to *b*) then *a* -≻ *c*, but *b* -/-≻ *a* and *c* -/-≻ *b*. This results in what engineers would recognise as a layered architecture.
- The disjoint *goals* of the system. A system may have a number of independent (or nearly independent) goals. Each agent can, therefore, be designed to satisfy a single goal (or subset of goals), rather than the complex goals of the complete system. Indeed, it should be expected that the individual agents would not even need to be aware of the goals of the complete system.

These criteria, however, are only appropriate for a 'loosely coupled' system in which each agent has a greater interaction with its environment than it does with its peers. At the other extreme of this axis of decomposition lies the tightly coupled systems such as cellular automata, where interactions with neighbours are given a similar weighting to those with the environment.

Figure 15 illustrates the sort of structures which might result in the state space from these strategies.

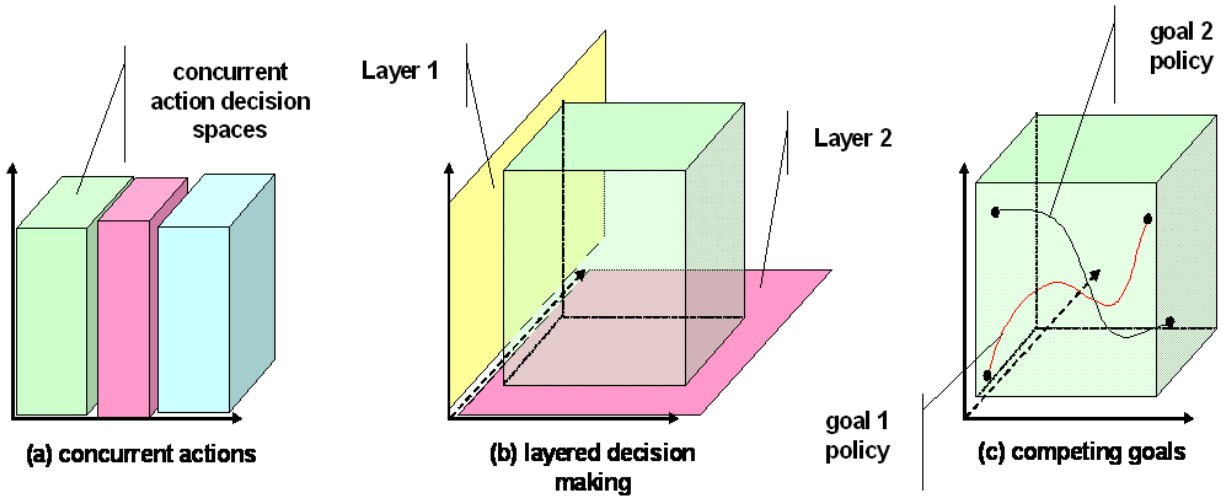

Figure 15 – State Space Structures

The essential problem for the designer is to find a representation of the system state space which adequately captures the behaviour of both the component agents and to corresponding (potentially coterminous) subsystems and to the system as a whole, and which, moreover, allows behaviour to be predicted sufficiently to assure compliance with the system performance requirements. A number of strategies have previously been proposed [23]:

- Decompose the state space into mathematical *subsets*. Each subset represents a partition of the environment in which concurrent agents operate. Typically the subsets will be nearly disjoint. The intersections represent intercommunication between agents, which will be of the form 'agent A is in state X'.
- Decompose the state space into mathematical *subspaces*. This is a specialised form of subset which has a more constrained structure. In particular, where each subset will, in general, have access to the same state variables as the original state space, in the subspace model only some of these variables are available to each agent. Communication between agents takes place through shared variables. This is the model to adopt for a layered structure.
- In a tightly coupled system one can form, around each agent, a '*local approximation*' of the global state space. This allows an agent to be designed and analysed independently from the influence of other agents, but with a well-defined mechanism to incorporate that influence later. In other words, it allows (at least to a first approximation) the local and global decision-making to be decoupled.

What will be required for a mathematical formulation of such system models is a method to decompose the state space around, and then to implement a simple control process on top of, this three axis structure. This will need to incorporate interdependencies between partitions to represent the complexity in system behaviour.

In particular, in order to be relevant to the complex multi-agent systems problem domain, the mathematical approach selected will need to be capable of modelling the difference between local and global views of the state space.

5.5 Constructing a Hierarchy of Decision Processes

The decision making process for a complex system is an interaction between various (possibly competing) requirements, and the resulting behaviours are typically the composition of many separate and distinct actions. What is required is a mathematical framework for modelling and analysing system behaviour that decomposes this complex and interrelated set of requirements and actions into a composition of simpler processes.

The first key concept behind this thesis is the idea that the controller of an autonomous system will have to be segmented into individual processes and subsystems, in order to capture mathematically how the control inputs for specific system components affect the state of other components. Such decomposition in itself is a conventional approach to modelling complex controllers. However, the number of states required for even a coarse description of a system can be large – combinatorial explosion of the state space is common as the model complexity increases, which can make the analysis of the system difficult if not impossible. Establishing a series of operational layers within the system could help, although care will still have to be taken to keep this decomposition sufficiently coarsely grained for the model to be tractable.

Although hierarchical controllers are not a new idea in control theory, the innovation in this thesis is to pursue this idea within the context of an analogy between decision making and control, and, moreover, take this forward into complex multi-agent systems. A system component that has to select a control input in a given situation is making a decision based on its current state and its environment, therefore the mathematical apparatus for decision processes should be applicable to the control problem. The decision making apparatus will be introduced in Chapter 5.6; this section is concerned with the decision hierarchy that must be constructed for the system in question.

There are likely to be many different ways of describing a hierarchical decomposition of a system, all a variation on the basic idea that a system is composed of a number of subsystems, which are in turn composed of a number of sub-subsystems, and so on down into the inner workings of the system. Layers in a hierarchy represent the 'depth' of a subsystem, or the number of distinct systems or processes that sit on top of it before one sees the output from that component. However, working with the notion of a decision hierarchy requires a slightly different perspective of hierarchical decomposition. The immediate way to think of what might be termed as hierarchical decision making would be to have strategic decisions, tactical decisions, and, at the bottom layer, the control problem, analogous to the layered perspective one might have with operational decision making. An autonomous system has to balance these layers throughout its operations, considering strategic objectives like its progress towards its overall objectives (its 'mission goals') alongside its control level problems such as the number of volts to apply to a particular mechanism. One might expect there to be interaction between different mission goals or operational constraints at the various levels of decision making. Note that this description of the system levels is just as an example; the actual number of levels in the hierarchy depends entirely on the system in question.

Figure 16 below introduces the three hierarchical dimensions that are characteristic of a decision making problem. The layers typical of hierarchical decomposition are present, described here as the layers of operational decision making (which may be regarded as strategic, tactical, and control layers). However, each decision layer can be further decomposed into series of parallel and concurrent decision processes. A parallel decision process refers to the process of choosing an action with regards to concurrent functional requirements, and a concurrent decision process refers to the process of many parallel system elements choosing actions that together support a single requirement. The key point is that one can identify one-to-many and many-to-one relationships between goals or requirements to be fulfilled and actions to be taken.

A useful way to interpret this model is to consider the three 'dimensions' of decision making as the 'layers' which partition the state space (the 'level' the decision is made at), the 'concurrency' which partitions the scope of each action, and the multiple goals which partition the requirements, consistent with the logical structure presented in Figure 15.

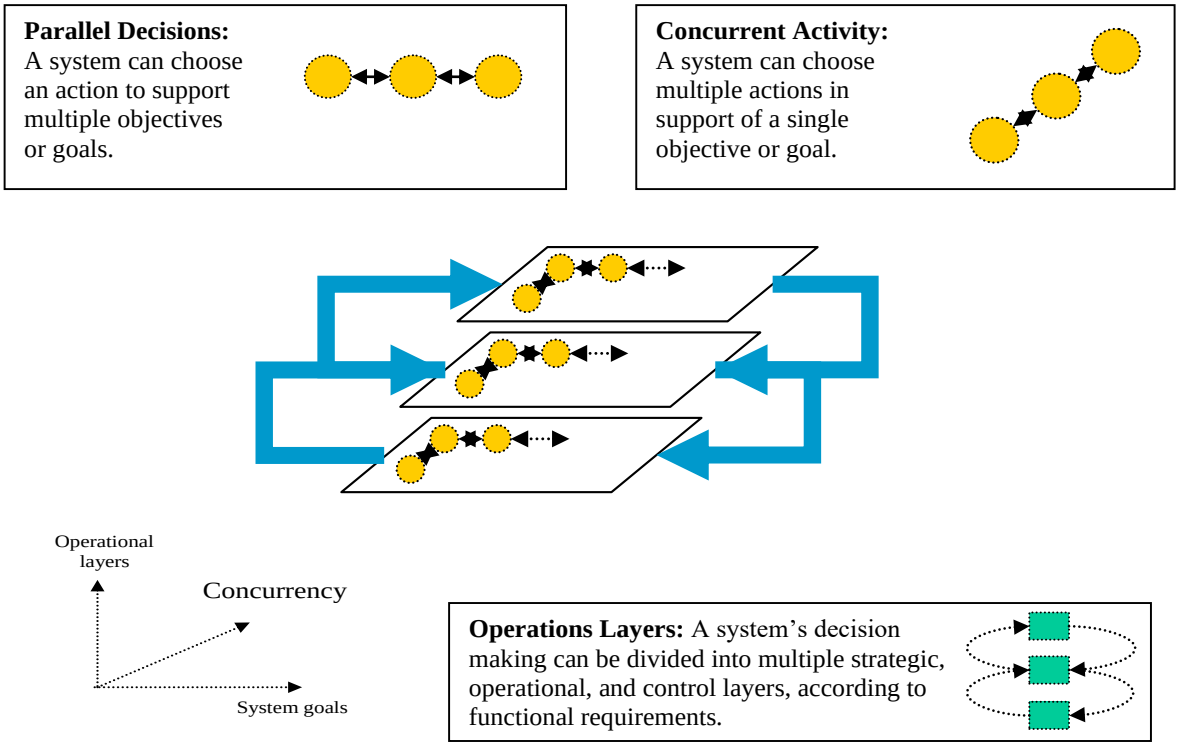

Figure 16 – Hierarchical Decomposition of Decision Making

The benefit in decomposing the decision making into these simpler processes is that they are linear, and require only relatively minor modifications to the standard model and to the typical algorithmic solution methods to enable their representation and solution as MDPs (to be defined in Chapter 5.6.2). It will then be only a small step further to compose these simple linear processes to give a many-to-many relationship between goals and actions which can then also be solved algorithmically.

Note that the approach to complex multi-agent system design being built up in this thesis has been developed for general application across a full range of system organisational constructs, featuring as many layers and agents in different hierarchical structures as the designer requires. Particular attention will be paid later to parallel systems, because these are characteristic of the synchronised autonomy challenge described in Chapter 2 and identified as the primary focus for this PhD, but at this stage of introducing the conceptual building blocks the full hierarchical framework is presented.

Referring back to the exemplar application areas described within Chapter 2.5, it is useful to consider how this theoretical construct of the dimensions of decision making aligns to those real world examples. The descriptions of the examples were based around the idea of system

layers, referring to the natural or logical ordering within systems based on information dependencies and information flow. These correspond closely to the operational decision layers, in that the impact of a decision made at the lowest component layer (actuators, servomotors, digital filters…) is a contributor to the current state and future decision making of the subsystem of which they are part, and so on up the hierarchy. Figure 1, Figure 2 and Figure 3 have been presented in such a way as to highlight groupings of agents in layers, but one can also see how there may be multiple competing requirements or objectives to be balanced within the same system level.

Before proceeding with a worked example in detail, it is important to clarify a key distinction between the two presentations despite the similar layout of the earlier figures compared to Figure 16 (and to Figure 17 and Figure 18 which follow):

- The nodes in Figure 16 represent required system actions or outcomes (with the author having previously introduced the terms parallel and concurrent in this context to differentiate the two in the precursor work [23],[24] before reintroducing them here).
- The nodes in the earlier figures in Chapter 2.5 represent agents making decisions.
- It may be the case that an agent is dedicated to each control point or decision, i.e. if a system has to send 5 control signals to 5 subsystems, it is designed and implemented with precisely 5 agents, one for each decision. It may also be the case that different agents can be incorporated to focus on decision making for separate but concurrent requirements placed on that system, potentially (notwithstanding the challenges described in Chapter 4.2) brokering some mutually acceptable solution between themselves. In general however, this appears to be a simplistic and prescriptive approach, and is not a requirement. The actual software design and implementation need not be so constrained and other factors will determine how many agents are required, probably based on other considerations like natural or practical subsystem boundaries.
- In conclusion, despite the apparent mirroring of the structure between the hierarchical decomposition of decision making captured in Figure 16 and the representations of candidate agent networks in Figure 1 to Figure 3, these are not intended to represent or imply mandatory one-to-one mappings of decisions to agents. A decision to be made might, but does need to, correspond to an agent, and a layer of decision making might, but does not need to, correspond to a group of agents at the same level of the system.

5.5.1 Parallel and Concurrent Decision Making

This section considers the decomposition of an operational level in a decision hierarchy into its component decision processes occurring at that level. The parallel decision process refers to a decision process that involves selecting one action in support of multiple goals. At each operational level, however, there is a second dimension to consider, namely the decision process that involves selecting multiple actions in support of a single goal. The exact formulation will depend on the particulars of the system, but the key feature is the distinction between having parallel goals to fulfil and concurrent actions to select at each separate operational level in the system. Complexity can arise on any one level as the influence one or more of the concurrent actions selected has on the selection of another action supporting a different goal (or combination of goals).

To support multiple goals, one might anticipate a large number of state variables to be necessary, proportionate to the number of goals as well as the complexity of the overall system. It will be important to structure the problem with the smallest practical set of state variables and possible actions in order to keep the analysis as simple as possible, and the hierarchical decomposition is then the process of apportioning the whole set of states and actions into the decision domains for which they are required.

To illustrate this approach it useful to consider a practical example that features a decision model with two interrelated action selections and two dependent system goals. There is one action to be selected in support of the two goals, and one of the goals is supported by both actions. This is the simplest a decision model can be while featuring both parallel and concurrent decisions, but in practical applications there will be as many goals and actions as required by any given system.

With reference to the production line automata application area from Chapter 2.5.3, consider a system consisting of a number of robotic appendages conducting a variety of tasks on a production or assembly line in series. There will be robots or teams of robots conducting specified tasks on items on the production or assembly line (called units hereafter for convenience). Suppose that one of the tasks is a quality control and repair activity. Units on the line reach this stage, and the requirement is that the units be checked with a scanner to ensure they are undamaged or unblemished, and if not, that they be corrected. One can imagine that this could consist of applying polish to remove any smears detected, or applying further spray paint or lacquer to any imperfections from an earlier treatment, or filling any detected

gaps with an adhesive. Each unit that arrives will have to be scanned all over to check for imperfections, and then if any are identified these will need to be fixed. For convenience, the possibility of any irredeemable units needing to be discarded is not considered.

In this scenario, the system designer might identify that there are two system goals to be achieved here:

1. To ensure that each unit is scanned all over to identify any faults or imperfections.
2. To ensure all faults or imperfections are fixed.

Suppose that units need to be lifted into the air while the sensor scans it all around. This is at least a two robot task – one robot arm to support the unit in the air, and a second fitted with the sensor that will conduct the scan. Whether the applicator is fitted to the same arm as the scanner or if there are yet more robots involved is not critical, but for simplicity here it is assumed to be fitted alongside the scanner.

The system designer might further identify that there are two actions (or sets of actions) that the system will need to determine:

1. Determining the pattern of movement that is required to fully scan the unit for imperfections in the shortest time. This could involve choosing one pattern from a set of pre-determined patterns if there is regularity to the types and shapes of units, but in more advanced concepts (for example a highly digitised production line capable of delivering bespoke variations of items that may display customisation from unit to unit) this may be something that has to be determined each time. In either case, a suitable scan pattern needs to be determined somehow.
2. Determining when to turn the applicator on and off. A speedy and efficient solution, if reliable autonomy can be achieved, would be to have faults or blemishes dealt with as soon as they are detected. In advanced concepts, this might involve a local decision in real time as to which colour of paint or type of polish to apply, depending on what imperfection has just been detected.

The system has to determine two actions to take – the relative motion of the two robot arms, and the use of the applicator. With autonomy and real-time adaptivity, these might need to be continually reassessed during the task; for example, following the discovery of a blemish, determining the appropriate treatment, pausing the scanning motion to undertake this treatment, and then seamlessly resuming the scan once the treatment is complete.

Figure 17 below illustrates this example as parallel and concurrent decision processes, where there are three elements (labelled as decision outcomes 1, 2 and 3 in the figure) resulting from these decision processes:

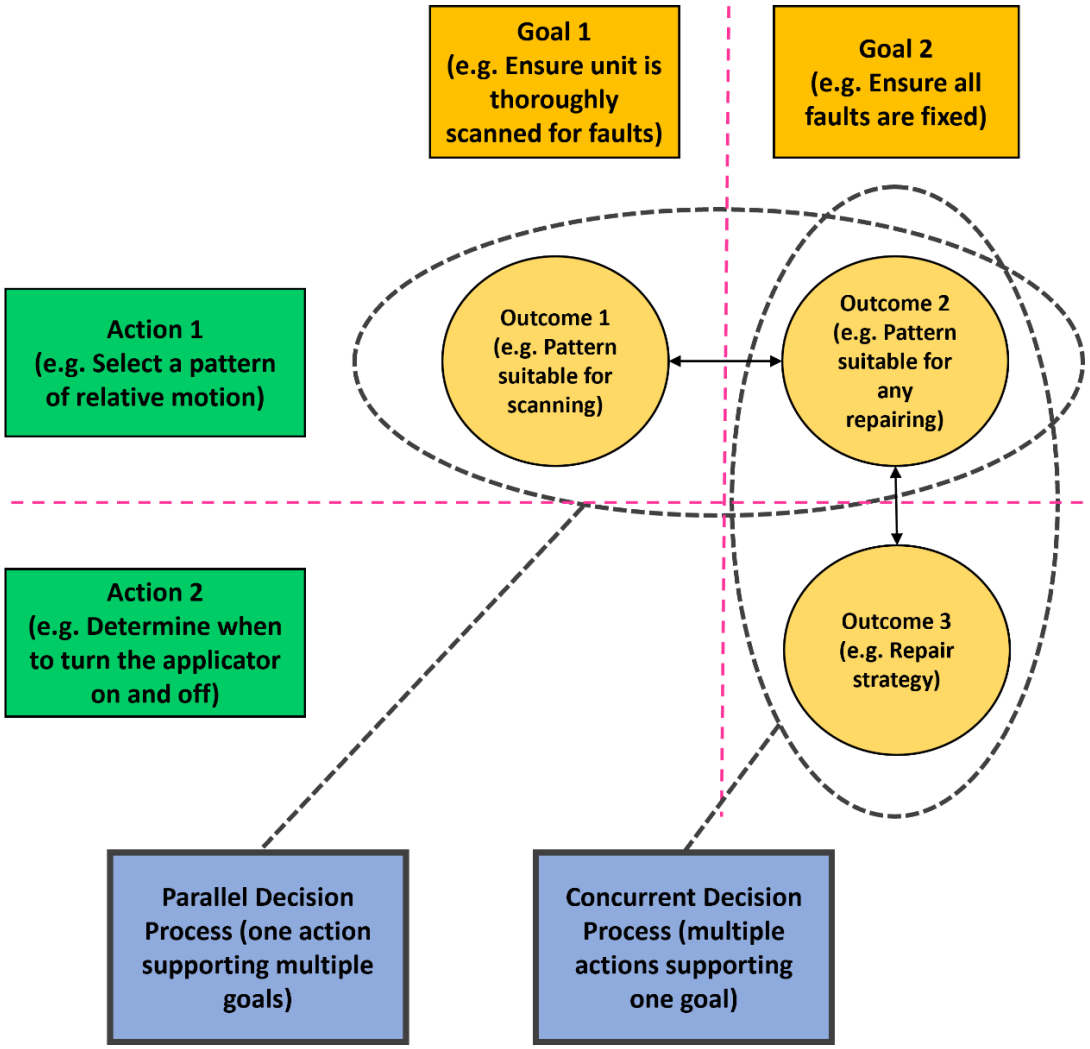

Figure 17 – A Graphical Representation of Parallel and Concurrent Decision Processes (Production Line Automata Application)

Figure 17 illustrates:

- The selection of Action 1 is a parallel decision process:
    - One action selection is made that has to simultaneously satisfy more than one goal.
    - In simple instantiations, this may be a one-off selection from a list of options; in more complex instantiations, this may be an on-going process of continually assessing and adjusting in order to maintain the best course of action.

    - For this particular example, this means that the movement pattern (i.e. the series of relative movements of the two arms to ensure the scanner can see all around the unit that is lifted into the air) has to be selected and maintained with regard to both goals – to satisfy the requirement to fully scan the unit from all sides, but also to be responsive or adaptable enough to adjust movements when necessary to conduct any necessary repairs.

- The selection of Action 1 and Action 2 simultaneously is a concurrent decision process:
    - Both actions need to be selected correctly in order to achieve one of the goals.
    - No matter how optimal the strategy is for meeting one goal (for example, a speedy scan pattern of the unit), the other element is also required to be able to complete the goal (in this case the system also needing to be able to correctly fix the fault).

In a more general formulation, if the selection of action a_i affects system progress towards multiple goals, this action selection is a parallel decision process. Not only should the outcome following the selection of action a_i reflect the progress towards all relevant goals, but the control policy that leads to the selection of a_i should reflect the dynamics and requirements of all relevant objectives. Thus the selection of action a_i can be seen as a competition or collaboration between components (depending on the context), where multiple system goals need to be balanced when selecting which action to take.

Equivalently, if the selection of two actions a_i and a_j affects the progress of a system towards a common goal, this selection is a concurrent decision process. The separate control policies that lead to the selections of a_i and a_j should both reflect the dynamics and requirements of this common objective. This generalises to higher dimensional problems with more than just two actions a_i and a_j in the logical way.

It is deliberate that action a_i chosen by the parallel decision process is coupled to the other action selection by the need to mutually support a common system goal. This coupling of parallel processes at each operational level is the essence of the complexity that can arise in hierarchical decision making. The vision for general hierarchical decision processes is that they can be decomposed into a network of such parallel goals and action selections divided across different operational layers, the exact structure dependent on the particulars of the system in question. The model above, with two goals and two coupled actions to select on a

single layer, is the simplest such form of hierarchy (besides essentially trivial cases that feature only one of the dimensions of hierarchical decision making).

Note that decision processes interacting with each other might be implemented as two separate agents within the same system assigned different tasks. Each agent is independent from the other since neither of them is under the direct control of the other. However, the actions selected by one of the agents may well change the state of the system for the other, even though the other has not done anything to prompt this. Alternatively, the action taken by the first agent may somehow change the environment, and then future actions selected by the second agent may not have the same effect as they had previously. Furthermore, the second decision process can similarly affect the first. The decision making processes of the agents are fundamentally linked despite the agents themselves being separate with distinct roles.

5.5.2 Layered Decision Making

Parallel and/or concurrent decision processes such as those above could feature on multiple layers of a decision hierarchy. Series of interlinked and concurrent goals and action selections on multiple operational levels require the system designer to also consider how the decisions made at different operational levels affect each other.

It is not difficult to think of examples of layered decision problems featuring decisions to be made and actions to be selected at distinct planning and control levels as well as, crucially, featuring the interaction between them (for example, a robotic system's overall controller, concerned with the delivery of the overall system objectives, and individual servomotors concerned only with the appropriateness of their individual positions and orientations). At either level there could be further segmentation into parallel goals and/or concurrent actions.

To illustrate this approach, it useful to further extend the example from the previous section to now incorporate the decision making interactions with a second system layer. This is the simplest a decision model can be while featuring all three of the hierarchical dimensions from Figure 16 (i.e. each of parallel, concurrent and also now layered decision making), but in practical applications there will be as many goals, actions and system layers as is required by and appropriate for any given system.

Continuing the example from before, suppose that all the decision making previously described and illustrated in Figure 17 takes place within the highest layer of the system (or at least, the highest layer of the relevant part of the system). Two decision processes result in two action

selections (or more realistically, two control signals). The system has been designed so that the detailed specification of exactly how these actions will be undertaken is delegated to a lower system level – resulting in a set of lower level decision processes and action selections.

A simple instantiation of this problem across two layers contains all the decision making previously described on a higher level planning or deliberation layer, with a lower level of control or configuration layer decisions still to be defined. The decisions to be made at this level will be about detailing how to implement the decisions that have come down from above:

- When a scan pattern is determined by the higher layer (or more specifically, when required relative positions have been provided), this will then prompt lower level decisions about how each of that arm's joints need to be adjusted to support the new plan of action. This will be the case in both arms, as they have to align to ensure the correct relative positions are maintained as required.
- The arm fitted with the scanner and applicator will have additional lower level decisions to make concerning the orientation of the function head, which will also be subject to a decision process to ensure that it remains aligned with the higher level requirement.
- Furthermore, there is also the second action determined by the higher level – when to apply or disapply the applicator. It may be that a lower level of decision making is required here not just to turn the applicator on and off when directed, but potentially to make further decisions about applicator pressure, angle, or composition – or even to choose which substance or colour to apply based on a local determination of the nature of the imperfection or blemish being treated.

It should be noted, with reference to Figure 3, that this planning layer of the decision hierarchy might be delivered by agents corresponding either to the system layer or to the system-of-systems layer of that illustrative example, and the new configuration layer of the decision hierarchy might be delivered by agents corresponding to the subsystem layer. However this is not the only way to implement this, and the layers do not need to be seen as equivalent.

Figure 18 below illustrates the full range of decision processes that have been described. Note that the two control or subsystem layers refer to separate decision processes within each of the two robotic arms, the first being the support arm that only has to deal with the movement challenge, and the second being the applicator arm that has both the movement and the additional applicator challenge. Parallel and concurrent decision processes can be seen to emerge within these lower level decision processes where dependencies exist.

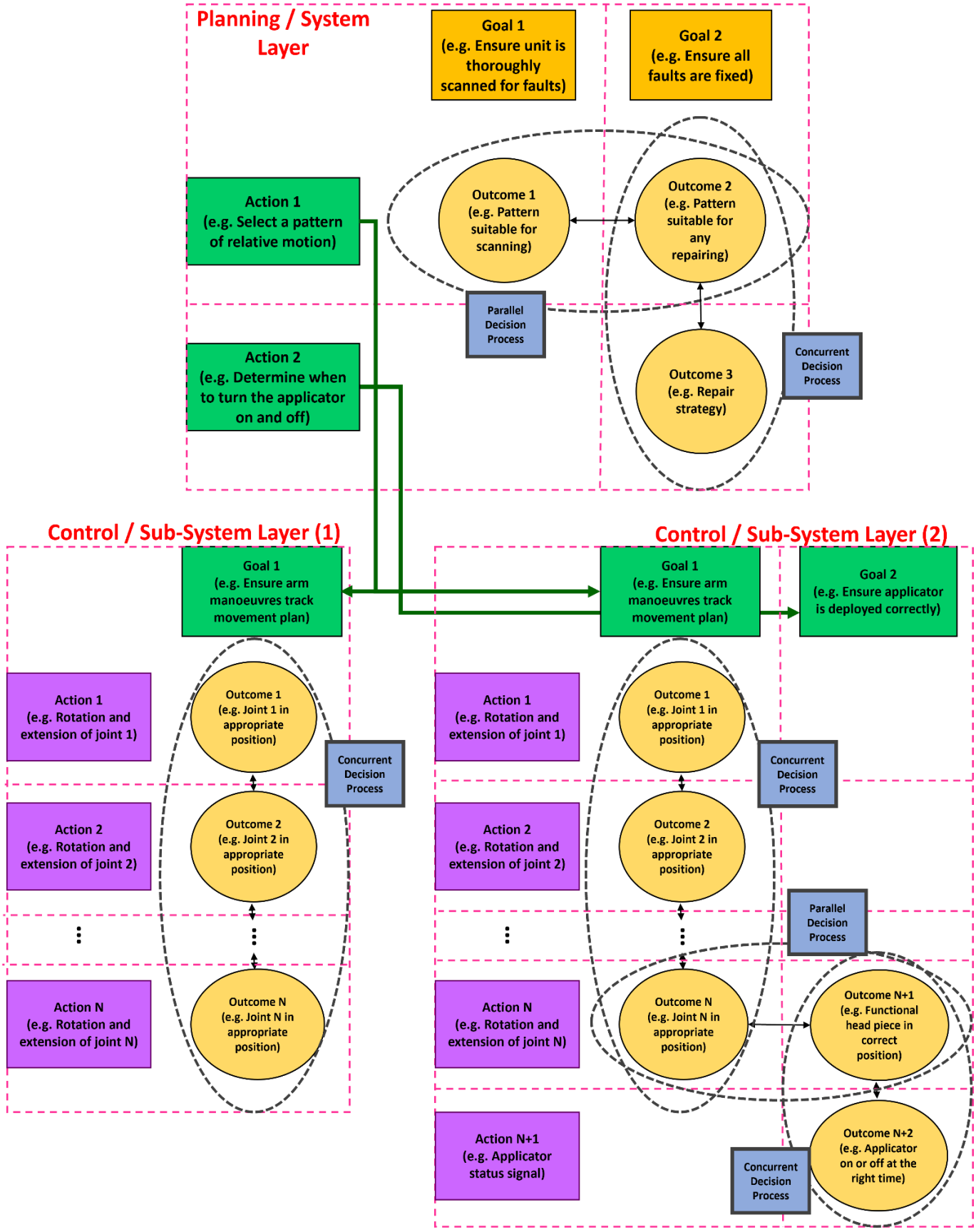

Figure 18 – A Graphical Representation of Parallel, Concurrent and Layered Decision Processes (Production Line Automata Application)

5.6 Modelling Complex Systems

5.6.1 The Mathematics of Decision Making

This section begins the development of the proposed mathematical model for representing and understanding the behaviour of complex multi-agent systems. This development is deliberately generic, focusing on modelling concepts and possibilities; since every system is different, any solution will need to be tailored to the particular structure and context of a practical system implementation.

Markov Decision Processes (MDP) methods for decision-control problems were recommended (albeit not in the context of multi-agent systems) in the author's precursor work for several reasons, including specifically their generality of application, the guarantee of convergence of solution in most ordinary circumstances, and their (relative) tractability compared with alternatives. All of these reasons remain valid for the more advanced complex multi-agent systems applications studied in this PhD, subject to adaptations and extensions which will be described herein.

MDPs provide a mathematical framework suitable for modelling decision making processes in inherently uncertain situations. That their description is context free and flexible is the reason MDPs can be applicable to many decision making and optimisation tasks in numerous research fields, and accordingly they are the key building block in this work. In the next section, MDPs are defined and some of the possible solution methods described, hierarchical variations of which will be developed and applied later in this thesis.

Chapter 5.6.2 provides a brief summary of the key MDP concepts. Chapter 5.6.3 discusses their relevance and application to learning systems. Chapter 5.6.4 introduces the Q-value formulation of the MDP which operates on state-action pairs, and provides a natural formulation for agent-based systems. Chapter 5.6.5 provides some background to the range of solution techniques available. Chapter 5.6.6 defines the solution technique selected for use herein (Q-learning). Chapter 5.6.7 then describes the novel approach introduced in this thesis for adapting the MDP formulation to apply to multi-agent systems. Chapter 5.6.8 also introduces for context the Partially Observable Markov Decision Process (the POMDP), an alternative tool that is theoretically a more complete framework for autonomous decision making, but unfortunately is considerably harder to solve, with no guarantee of convergence to a solution.

5.6.2 Markov Decision Processes

Note that across the extensive literature, Markov processes are described using various different notations. For consistency within this thesis and its associated publications, definitions and notations have been adopted that are consistent with the well-known text by Sutton and Barto [59].

The key building blocks for an MDP formulation of a decision-control problem are:

- As is the customary approach in modern control theory, the current situation, or the system *state*, is described by a (finite) set of variables $s \in S$, which summarise the most pertinent factors influencing the system.
- Working in discrete time, the change in state each time step is a *state transition*. In an ordinary Markov chain, the sequence of observed state transitions can be regarded as the behaviour of the system.
- MDPs also have defined actions $a \in A$, a (finite) set of choices available to the system at each time step that influence which state transition occurs next (not necessarily deterministically). Where Markov chains are passive, modelling behaviour rather than control, the addition of an action selection process gives systems the capability to choose between different courses of actions, or to switch between different branches of a tree of Markov chains. It is this choice that progresses a system from reactive to higher levels of autonomy.
- The system earns a numerical *reward* $R_{s,s'}^{a}$ for each state transition $s \rightarrow s'$ that follows the selection of each action a.
- Providing all obtainable rewards are finite, and providing that all state transitions are probabilistically well-defined (all probabilities between 0 and 1, with the sum over all possibilities equal to 1), MDP solution algorithms can identify (offline or online, analytically or learned) those actions to take in particular states that return the greatest expected cumulative reward (see [59]).
- The *policy* $\pi(s, a)$ is an association of action selections to current states (not necessarily deterministically). In general, it defines the probability of choosing action a when in state s. The solution to an MDP is expressed as the *optimal control policy*, which identifies those actions to take to maximise the expected return.

- A system has a *goal*, which drives behaviour by inspiring the definition of suitable rewards. Desirable behaviours supporting the system goal should be rewarded better than undesirable behaviours which do not. Maximum attainable reward should correspond to the completion of the goal (or the closest it is possible to get to it). Expected cumulative returns are used to avoid the obvious pitfalls with greedy policies that seek to maximise the immediate reward to the potential detriment of longer term performance.

Note that implicit in this model is the *Markov Property* that the optimal action can be decided on the basis solely of the current state. Also note that, as Puterman [56] shows, under mild constraints there can exist an MDP in which the policy is deterministic, i.e. in which for each state there is a single action which will be selected for optimal behaviour. It is on this basis that one can claim predictable behaviour from the decision process.

Note that different choices of action will typically result in different state transitions and thus different rewards. The decision process is that of choosing which action to select when in a particular state, based on what transitions and rewards would be expected to result from such a choice. The main advantage in using MDPs instead of traditional control methods is that this formulation can incorporate uncertainty in the model or the environment by calculating (or estimating) the expected return following each action selection.

The biggest challenges for the engineer designing an autonomous system controller are firstly to ensure that the MDP is suitably defined to reflect the impact that the different variables can have on the behaviour of the system (the fidelity of the state and action spaces S and A compared to the real world), and secondly to ensure that the goals, conveyed via the reward scheme, really do prompt the desired behaviour. The optimal control policy is optimal with respect to the defined reward scheme; whether the behaviour caused by this policy is what was desired depends on how well the goals and rewards were aligned in the first place (notwithstanding any environmental perturbation that can also be a factor in some contexts). Clearly there is a delicate balancing act in the initial design phase that will need to be considered. This will be highly specific to individual applications, with no uniform solution.

The basis for most MDP solution techniques is the Bellman value equation. The notion of the value equation first appeared in Bellman [8]. Variations of the definition can be found in the wider literature, but this thesis will adopt the formulation from Sutton and Barto [59]:

$$V^{\pi}(s) = E_{\pi}[R_t|s_t = s] = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'}\left[R^a_{s,s'} + \gamma V^{\pi}(s')\right]$$

where the value V of state s is the expected return R_t (assuming one is following and continues to follow policy π). Expected return is the expected cumulative reward defined in the logical way as the probability-weighted reward expected from each action, summed over all possible actions. Note the recursive formulation, with R_t consisting of the expected next reward $R^a_{s,s'}$ and the (possibly γ-discounted) value of the possible successor states $V^\pi(s')$, summed over all possible successor states s'. This enables many algorithmic solution methods to solve even large models in reasonable time, either searching for an explicit analytical solution for a fixed world model, or applying machine learning techniques to identify the currently best understood solution for a dynamic world model.

Note also that the values of the state transition probabilities $P^a_{s,s'}$ and the action selection policy $\pi(s,a)$ will depend on the particulars of the system in question. It will be necessary to define these for each set of states and actions, unless working with a machine learning algorithm which does not require a priori knowledge of the transition probabilities because, by definition, it learns them (either offline from observation or online from experience).

MDPs are not the same as simple Markov chains, which have neither rewards to receive nor decisions to make as to which action selection will maximise that reward. This distinction can be thought of as a Markov chain describing a path that is being followed, where an MDP is a process to select which of several paths is the best. Each possible sequence of actions one could select when in a given state produces a Markov chain, and the solution to the MDP is the Markov chain identified as giving the system the best expected overall result, calculated using the rewards.

A few other comments about MDPs are made at this stage:

1. The state is a description of the important properties of the system at a given time. They could include the system's position, velocity, temperature, bearing, or any other variable that affects system operation and impact upon the decision making. What should be a part of the state depends on the system in question and the context. However, one should be careful to include only necessary state information as, although MDPs can theoretically model vast amounts of state information, high dimensionality systems will be difficult or impossible to solve efficiently or accurately and may consume excessive computing resources in the system. Choosing an appropriate state space representation is critical to making the process practicable.

2. Any variables outside of the control of the system are part of the environment. The environment can affect the transition probabilities of a system, in that external influences on a system can affect its performance. The probability of an action causing a particular state transition is sensitive to the environment, reflecting the fact that the control input from an autonomous system's controller may be more or less effective than expected due to adverse environmental conditions. This is an aspect of decision making under uncertainty – actions may not have the consequences they are expected to have. They may not even have the consequences they had last time that action was taken when operating in a dynamic environment.

3. Markov Decision Processes are so named because they require adherence to the Markov Property. This means the system's current state contains all the information on which the next action selection is based. Ignoring the results of previous action selections which could inform the current choice may not seem like an optimal way of making decisions. However, for two reasons, assuming that the Markov Property is satisfied is rarely a problem in practice. Firstly, the use of value functions allows the system to learn from these prior action selections implicitly. Secondly, a potential workaround exists where pertinent pieces of historical information can either be stored as or derived from current state variables (in the latter case, in the way past position is derivable if current position and recent velocity are known). However, care would have to be taken to avoid prohibitive exponentiation of the dimension of the state space to ensure that only a valuable proportion of information was being stored or recovered.

4. Alternative definitions for MDPs exist. Often the difference is merely in the choice of notation or indexing convention, but in some of the literature the difference in definition is more fundamental. One common variant is the definition of the related POMDP (see Chapter 5.6.8). Another defines the MDP as a quintuple, with the fifth element the decision making epoch – the set of times at which decisions have to be made. [An alternative approach to modelling episodic patterns of behaviour is proposed later in this thesis.]

5.6.3 Using MDPs to Model Learning Behaviours

It is sometimes easy to identify which actions return the greatest expected immediate reward – in fact, these will sometimes be defined, so it is known for certain which action it will be. More useful for real applications is to maximise the expected cumulative reward over many

transitions, usually known as the expected return R. Note that, for analytical reasons (to ensure the expected return is finite), future rewards in the series are usually weighted by a discount factor $\gamma \in [0,1]$, thus giving:

$$R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where r_τ is the reward earned from the transition made at time τ. The discount factor γ has the effect of reducing the contribution of earlier rewards, weighting the expected value towards more recent rewards. This is important in a dynamic environment, where earlier rewards may no longer be representative of what would happen were that state to be reached again in the future, where giving too great a weighting to earlier rewards might give false confidence about the future. The value of γ will need to be chosen to reflect how important these earlier findings are expected to be. Weighted near 0 and they are ignored almost completely, whereas weighted near 1 rewards are treated near equally, assuming little dynamism in the surrounding environment. Where in that range the 'best' value of γ is to be found will vary from case to case and may require some experimentation to determine.

Using expected returns reveals a key characteristic of MDPs – it is not necessarily best to select the action with the greatest expected immediate reward (defined as the greedy policy). Actions with poorer expected immediate rewards may be better in the long run if they cause a transition to a state from which greater future rewards can be gained than might otherwise have been the case. This is related to the question of how much flexibility and thus potential risk an autonomous agent can take in support of its mission goal, or how suboptimal its behaviour can be before it starts operating ineffectively or unsafely.

In general, the goal of an MDP is to identify the actions to take to maximise the expected return. Therefore in practice, setting an appropriate reward scheme is the key to coaxing the right behaviour out of a system that follows the decisions made by an MDP. If the system has a goal (or goals), then the rewards should be carefully defined so that actions which take the system towards its goal give better rewards than those which do not. The expected return from a given state (also known as the value of that state) is then a measure of how good it is to be in that state with respect to the overall goal. It is a key consideration at the design and testing stages to ensure that the rewards defined for state transitions really do guide the system in the way that is intended. This means that the optimal control policy of the MDP would then correspond to a decision making mechanism for an autonomous decision maker.

A simple example to illustrate these key features of MDPs is provided below. Imagine that a cow is grazing in a field divided in two by a stream. The simplest structure for a decision problem is to give the cow a choice of whether to cross the stream and graze on the other side or to stay where she is on the current side. Rewards of -1 are incurred for a crossing, since the cow is at that point getting her feet cold and wet whilst not consuming any grass, but positive rewards of 1 and 4 are earned when on either side of the stream, depending on whether the cow is in the left or right hand side of the field which have different amounts of grass. Figure 19 below represents this simple problem. It is acknowledged that this description is artificially simple, for example there is no consideration of what might prompt the cow to make a decision or of whether the grass will run out, altering the probability of making a transition to the other side of the field. MDPs are capable of modelling these situations should this be desired.

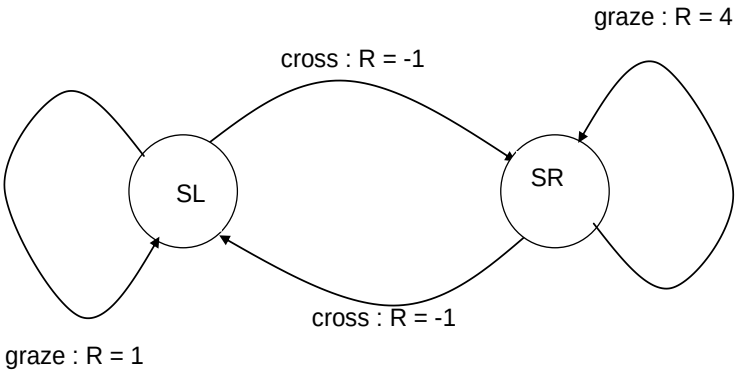

Figure 19 – MDP Illustrative Example (1)

Clearly, for this simplified description of the problem, the optimal policy solution is to stay on the right hand side SR when there, and if on the left hand side SL to make one crossing to the right hand side and then graze there (although there may be complications if one chooses to discount rewards as to when to make the crossing, but these are not considered here).

This simple model assumes the cow can at any time choose to move between the two fields. Sometimes the model may have to be more constrained. An alternative would be to enable the cow to cross into another field where the grass appears to be a lot greener but is then unable to return, say because the rickety bridge collapsed after the cow crossed it. In this case, although the grazing was better on the other side it was not the optimal policy in the long term to cross, because the cow may run out of food much sooner and be trapped.

Figure 20 shows a state model that represents this situation, where the crossing earns better immediate reward, but represents a terminal state (no further transitions are possible) and clearly demonstrates the potential fallacy in pursuing immediate reward over expected cumulative return. The optimal policy was in fact to stay put, and the safer choice of action was to not cross the bridge.

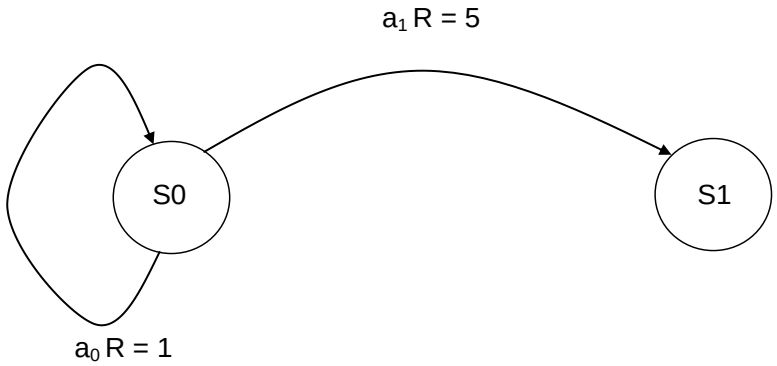

Figure 20 – MDP Illustrative Example (2)

5.6.4 The Q-Value Formulation

An alternative formulation for an MDP extends the idea of the value of a state to the value of an action from a given state. The expected return from the current state if action a is taken next is then a measure of how good that choice of action is for achieving the system goal, given the current state. Unlike state value $V^{\pi}(s)$, state-action value (known as the Q-value) $Q^{\pi}(s,a)$ can be calculated without a model of the consequences of each possible action that could have been selected, which can be difficult anyway but, when there is uncertainty in the model or the environment, could also be inaccurate or misleading.

Recalling the value equation, again using the formulation from Sutton and Barto [59]:

$$V^{\pi}(s) = E_{\pi}[R_t|s_t = s] = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'}\left[R^a_{s,s'} + \gamma V^{\pi}(s')\right]$$

$V^{\pi}(s)$ is the value of the current state s (equal to the expected return from that state) when following policy π, with $P^a_{s,s'}$ and $R^a_{s,s'}$ the corresponding transition probabilities and rewards, and with discount factor $\gamma \in [0,1]$. The above equation defines the probability-weighted expected return, considering the expected reward from the next state transition and the expected cumulative return beyond that, which is summarised as the (discounted) value of each of the

possible next states $V^{\pi}(s')$. This then gives rise to the alternative formulation using action values (Q-values) defined by the action value equation:

$$Q^{\pi}(s,a) = E_{\pi}[R_t|s_t = s, a_t = a] = \sum_{s'} P^{a}_{s,s'} \sum_{a'} \pi(s',a') \left[R^{a'}_{s,s'} + \gamma Q^{\pi}(s',a')\right]$$

The intuitive relationship between state values and state-action values can be expressed as $V^{\pi}(s) = \sum_a \pi(s,a)\, Q^{\pi}(s,a)$, which can be simply derived from the definitions as expected cumulative returns, the detail of which can be seen in [59].

The value equations show the overall value of state s under policy π to be the probability-weighted average of the values of each of the candidate actions $a \in A$ when in state s and following policy π. Whether state values $V^{\pi}(s)$ or action values $Q^{\pi}(s,a)$ are more appropriate depends on context; both are valid theoretically (provided some relatively mild analytical conditions are satisfied). Note the superscript π is often dropped as it is usually implicit that the value of a state or action is with respect to the policy being followed, but it may be more important to retain it for analysis of systems featuring multiple interacting agents; these agents would be following their own non-identical control policies, which would require consistent labelling and accurate enumeration.

5.6.5 Solving the MDP

Typical solution methods for MDPs involve value or policy iteration based on the Bellman value equation. Providing the total number of states and actions is finite, there is guaranteed to be an optimal action (or possibly a set of equally optimal actions) to take in any given state that will maximise the return from then on (provided the same analytical conditions as above are satisfied), as proven in [59]. Thus finite MDPs have a solution in the form of an optimal policy π^*, which defines which actions to take when in each state to achieve the maximum expected return $V^*(s)$ for each state $s \in S$. Formally, $V^*(s) \geq V^{\pi}(s)$ for all states $s \in S$ and for any alternative policy π (or the obvious equivalent formulation using Q-values).

Applied to this thesis, the question is whether one of these optimal policies results in predictable and appropriate behaviour. If none of them do, then the question becomes what is the next-best policy that does. A deeper question would be, if a trade-off between optimality and appropriateness is necessary, then is it even possible to achieve an operational requirement or goal without operating at an unacceptable level of risk.

The idea behind both formulations is that the expected return from a particular state is defined as a one-step recursion of the current estimates of the values of all the reachable states, amenable to iterative estimation and evaluation solution methods. One improves the estimate of the optimal policy by updating the current best estimates of the value of the previous state (or state-action pair) with the reward obtained when the transition has occurred, which updates all related values through the recursive relationship defined by the Bellman value equation.

Typical solution techniques range from dynamic programming to Monte Carlo-esque statistical sampling, and hybrids of the two. What is most useful depends on the context. Dynamic programming is useful when the full model is known (i.e. all $P^a_{s,s'}$ and $R^a_{s,s'}$ are defined, as well as $\pi(s,a)$ when appropriate) because then the value equations form a system of simultaneous linear equations (albeit possibly a large system). There will be an optimal solution to this system of value equations, and iterative dynamic programming solutions will find it relatively quickly given the potentially high dimensionality of the state space. Although applied to POMDPs (a generalisation of the MDP which will be discussed further in Chapter 5.6.8), it was dynamic programming methods that were applied in much of the early literature, including Bellman's original text [8]. Where the MDP is applied to a control problem like the hierarchical composition of subsystems (as opposed to a decision-making-under-uncertainty problem) this is not unreasonable, as in many contexts system behaviour will have been carefully specified.

However MDPs can still be solved when parts of the model are unknown, albeit sometimes only asymptotically given operational or computational constraints. A solution is guaranteed to exist so long as the relatively mild modelling requirements are satisfied (being finite, using discounts, and so on) – although it should be noted that an optimal solution may only be a transient solution (a 'currently optimal' solution) if the environment is changing. Sampling methods applied with a careful selection of sampling policy (to ensure state space exploration) will give data that might be used iteratively to improve the state or action value estimates, and similar solution algorithms will converge to a solution.

Monte Carlo sampling methods are not appropriate where safety assurance is required, because they typically require the current series of operations to complete before an unsafe action can be identified as such, by which time it is already too late to prevent it. However, compared to dynamic programming methods, Monte Carlo methods are just the other extreme of a spectrum of possible solution algorithms. Hybrid methods often give effective solutions, typically involving the use of sample data (in the form of learned data, data from a model, or observed

data) to update the values (and thus the current optimal policy) through successive policy evaluations, improving the model and/or the performance of the system as the solution algorithm progresses.

5.6.6 Q-Learning

The solution technique that will be used in this thesis is one of these hybrid solutions borrowed from the world of reinforcement learning known as a Q-learning algorithm, which calculates the optimal policy solution to an MDP from iterative updates to the action value equation (the Q-value variant of the tradition Bellman value equation) based on its practical experience and observations (i.e. rewards). It is a form of iterative policy learning where the current array of Q-values is updated according to the update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

or more intuitively:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\big(R_{t+1} + \gamma \max_a Q(s_{t+1}, a)\big)$$

where $\alpha \in (0,1)$ is the learning rate of the Q-learner and all other variables are as defined previously. α is a parameter that defines how much weight new information should have compared to the old estimate when updating value estimates. Clearly $\alpha = 0$ would correspond to no weight being given to new information at all and thus no changes to Q-values regardless of what transpires, whereas $\alpha = 1$ would correspond to responding instantly to new information with no weight given to prior knowledge at all, a purely reactive or 'knee jerk' response insensitive to learning trends in observations over time. In general, α will need to be tuned to the environment, with smaller values of α for more slowly evolving environments and greater values of α for highly dynamic environments, where the relatively fast changes to the environment reduce the value of older learning more quickly.

The solution to an MDP is the optimal control policy, which defines the optimal action (i.e. the most value-maximising action) one should take from the current state.

- During the learning phase of a reinforcement learning technique such as Q-learning, one would want the system to sample different combinations of states and actions in order to gather data about the 'true' values of these actions, with the value array (or Q-value array) serving essentially as an aggregation of all current knowledge (subject to suitable parametric weights). One may define an 'exploration' strategy or state space

sampling strategy. Given ample time, exhaustive state space exploration or simple random walks on the state space (or state-action space) will suffice, but more considered alternatives based on prior knowledge of the valuable parts of the state space to sample may help to speed learning up considerably.

- During the execution phase, it can be assumed that most or all of the time one would want the system to choose the optimal action. 'Most of the time' allows occasional sampling of supposedly non-optimal paths in order to re-sample the wider state space, essentially testing for any changes of which the system was unaware. Should the environment (or the rest of the system in the case of a multi-agent system) have changed during run time, the system may be able to detect this through sampling, iterating its value array, and observing that some state or state-action pairs have become more or less valuable over time. This will not always be possible as it will be inappropriate for some systems such as safety critical systems to purposefully take poorer decisions in this way, and thought should be given as to whether or to what extent such systems should be enabled for 'online' learning (forbidding it completely, setting an extremely low sampling rate, or allowing sampling but only within that subset of actions determined to be acceptable).

- Most of the time (as determined by a sampling strategy) for a system continuing to learn online, and all of the time for one that cannot, the action to take is the one which maximises value. The *optimal policy* is the policy which maximises the value from each state-action pair within state space S and action space A, which in the Q-value formulation will be:

$$\pi^*(S,A) = max_\pi\big(Q_\pi(S,A)\big)$$

- The *optimal action* (or a set of equally optimal actions) is that action to take under the optimal control which maximises expected return from the relevant state, which in the Q-value formulation will be:

$$a^*(s) = argmax_{a\in A}\big(\pi^*(s,A)\big)$$

Q-learning across networks is a generalisation of the standard formulation given above by having a set of parallel iterative updates to the Q-value arrays maintained at each node in the network. The author has previously introduced this and demonstrated how it can be applied to random networks and node-by-node frequency reassignment [21]; however the developments

later in this thesis will be the first time networks of Q-learners are known to have been applied in this manner within the domain of complex multi-agent systems.

Note that the definition and notation for the Q-learning technique given above are, as usual and for consistency, taken from Sutton and Barto [59]. However, since selecting Q-learning as the preferred solution technique for this work, Watkins' original thesis [67] that first introduced Q-learning has been tracked down, and it is a pleasant coincidence that the problem originally tackled (a strategy-control problem for a boat, where strategy means the navigation problem) is very similar to the vehicular navigation example discussed throughout this thesis. This was an unexpected but welcome discovery, and a validation of this choice of solution technique for decision-control problems. However, what has not been discovered in the wider literature is any prior attempt to extend the Q-learning formulation to multiple such decision making agents within a complex multi-agent system, which is the innovation in this work.

Note also that although Q-learning is defined here and is the solution approach preferred for the exemplar application (see Chapter 8), it is not the only possible solution technique, as the discussion earlier in the subsection indicates. Alternative approaches might be more suitable for calculating, maintaining and updating optimal control policies depending on the problem domain in question.

5.6.7 Generalising the MDP to Multi-Agent System Structures

The discussion thus far has concerned the MDP as a suitable mathematical formulation for the decision-control process being undertaken by a single decision maker. The focus of this thesis is to generalise this approach for more complex systems structures consisting of multiple interacting decision makers.

There are a number of modifications needed to apply this model to more complex structures of decision nodes. This section summarises some aspects that are original contributions by the author but originated in the precursor work that motivated this PhD [23],[24]. The concepts need to be re-introduced here as part of the background before proceeding from Chapter 5.7 with the original contributions conducted as part of this PhD.

Suppose one can identify m relevant state variables and n actions that need to be taken:

- A *state* becomes a *state m-tuple*:

$\underline{s} = (s_1, s_2, \ldots, s_m)$ is a tuple describing m relevant variables $s_i, i \in \{1,2,\ldots,p_i\}$, where there are $p_i < \infty$ options to describe the current state of the i^{th} variable.

- An *action* becomes an *action n-tuple*:

  $\underline{a} = (a_1, a_2, \ldots, a_n)$ is a tuple describing the n actions $a_j, j \in \{1,2,\ldots,q_j\}$ that need to be decided upon, where there are $q_j < \infty$ options for the j^{th} action.

- A *numerical reward* should be generalised to become a *reward function*:

  The combined reward function $R = f\left(R^a_{s,s'} \middle| \underline{s}, \underline{a}\right)$ defines the reward that is obtained from the whole set of actions $\underline{a}$ selected given the current state $\underline{s}$, which could be a simple sum of rewards or could be something more complicated if necessary to incentivise behaviour effectively. The exact functional form of R will be determined by the properties of the system in question.

Each state and action tuple can be treated as elements of higher dimensional state and action spaces respectively, and so one can adapt the Bellman value equation in the logical way:

$$V^{\pi}(\underline{s}) = \sum_{\underline{a}} \pi(\underline{s}, \underline{a}) \sum_{\underline{s}'} P^{\underline{a}}_{\underline{s},\underline{s}'} \left[R + \gamma V^{\pi}(\underline{s}')\right]$$

A more detailed construction of such a higher dimension Bellman value equation making use of the higher-dimensional generalisations defined in this chapter will be presented as part of a step-by-step worked example later in this thesis in Chapter 6.3.2.

Even in the simple case, the Bellman value equation actually describes a system of simultaneous equations. Generalised to higher dimensions, this could be expressed as a messy family of tensor equations, but doing so adds little practical value. Thinking algorithmically, this system is better expressed as an array (potentially a high-dimensional array), and solution algorithms can work iteratively on elements of this array just as they would on an ordinary system of simultaneous linear equations.

Note that as with all the single decision node developments discussed previously, the variation on the Bellman value function instead based on the value of actions given states $Q^{\pi}(s, a)$ can also be adapted for the complex multi-agent systems problem using tuples similar to the above. This version is useful for machine learning implementations, and it is the Q-value formulation that was adopted for the experimental work reported later in this thesis. In more general

application, the system designer will have a choice between formulations, depending on the problem in question and the appropriateness of the different solution algorithms.

5.6.8 Partial Observability and the POMDP

A well-known extension to the MDP comes about by removing the requirement to have explicit information about the current state of the system – incorporating any uncertainties in the system's knowledge of its current state. If, for example, such uncertainties arise due to physical limitations on the system's sensors, then the Partially Observable Markov Decision Process (POMDP) is an alternative model that incorporates this. Formally, the POMDP is a quintuple similar to the quadruple of the MDP, but with an additional (finite) set of observations and the set of states from the MDP replaced by a set of belief states in the POMDP. As the name suggests, a belief state is a probability distribution of the belief the system has of being in a given state, given the most recent observation and the last action taken. However, state transitions are now transformations of probability distributions, and thus the analysis and solution of a POMDP model is considerably more difficult than for a normal MDP.

Some solution methods for POMDPs do exist, but they typically require either explicit state transition models (which can be unreliable, or even unobtainable when operating in unpredictable environments), approximation to an MDP (which begs the question of why not just use a suitable MDP in the first place), or a highly involved and possibly intractable numerical solution. Thus a choice has to be made between the benefits of incorporating any uncertainties in the measurement or observation of state information by using a POMDP model, or the benefits of making the assumption that state information is sufficiently well represented without the need for explicit state transition models.

For all of these reasons, and given all the application areas discussed within this thesis relate to decision making in observable environments, it is proposed that simpler MDP models should be the preferred basis for the modelling and analysis techniques being developed on the assumption of full (or in practice, sufficient) observability of the current state. Then an appropriate state space representation and standard MDP solution techniques will solve the decision problem, with the added benefit of not having to work with specified transition models and/or potentially intractable numerical solutions. The possibility of a state space quantisation suggests that there is scope within this approach to incorporate some degree of uncertainty in state information, with the states the MDP works with themselves being just an approximation to a larger underlying (often continuous) state space.

It is recognised, however, that there are foreseeable applications where specifically including partial observability might be desirable, including multi-agent applications where systems need to work with hidden variables, i.e. state information that they cannot observed (or at least, cannot fully observe). This might arise in models where it is not possible to approximate away any uncertainty or noise in the input sensor data, for example when data is incomplete or broken up, such as might happen in some signal processing and frequency management applications. However, this is a topic beyond the scope of this thesis.

5.7 Application to Complex Multi-Agent Systems

This chapter will begin to pull together these approaches to formalising structures of systems and a mathematical formalism to overlay onto these structures, and illustrate how these might be applied to complex multi-agent systems.

Consider a generic multi-agent system with a large number of distributed system elements responsible for a variety of subtasks. In a larger system it is not difficult to imagine a situation where there are many competing demands for resource, and the system as a whole has a difficult challenge to meet all of these simultaneous goals. This scenario contains many of the characteristics of the sort of system described in the previous sections:

- **Multiple Goals:** There can be many separate and distinct requirements and goals placed on a system and it may be more practical and expedient to have them addressed individually by different agents or groups of agents (both for computation efficiency and to be responsive to locally dynamic requirements).
- **Complexity:** Applying resource to one requirement might deny this resource to other requirements, thereby creating a network of implicit dependencies between distinct tasks. This could make the system susceptible to complex or emergent behaviours.
- **Local vs. Global Perspective:** Although one would hope that the cumulative effect of all the decisions made by local agents in response to local requirements would address all the requirements on the whole system, clearly this may not be so. There are likely to be trade-offs between local and global needs. (Examples of these sorts of local/global conflicts have been given previously, for example the discussion about autonomous taxi's local need to remain with a customer-rich location but the wider system's global need to disperse assets to maintain coverage and responsiveness.)

Consider the overall structure of such a system:

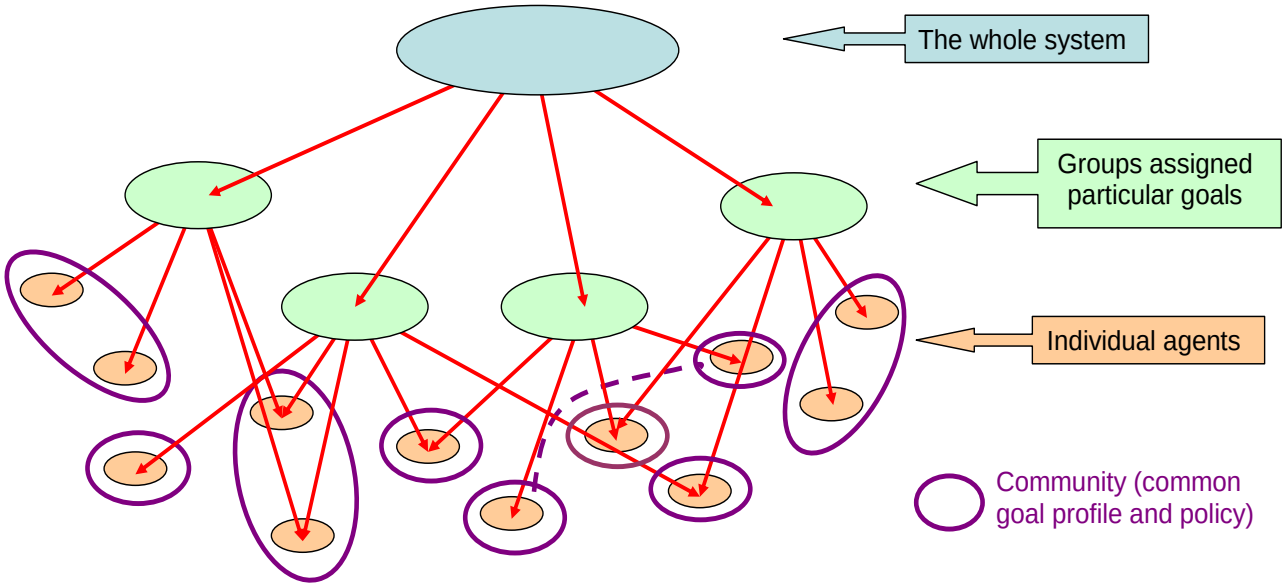

Figure 21 – A Graphical Representation of Generic Multi-Agent System Structure

The layers of 'whole system', groups, and agents are analogous to the layers of the motivating examples from Chapter 2.5, where groups are naturally defined as the 'branches' of the multi-agent network corresponding to individual vehicles, sensors, or automata in each of the three examples. The presentation here allows for groups to exist as a more general construct and not be limited to groupings imposed by physical alignment or co-location.

Communities is a new term being introduced to describe the potential for emergent groupings of agents that may not correspond to the agents' usual or formal grouping, based on these agents having common (or overlapping) goals and policies leading them towards co-dependent patterns of emergent behaviour. This will be explored further in Chapter 5.8.2.

The requirements placed on the system in the form of resource allocation requests have to be assigned to those agents capable of fulfilling them, which could involve 'percolating' the requirement through the network until an agent (or group of agents) capable of fulfilling the requirement can be tasked (not dissimilar to the approach taken in swarm robotics). However this is a specific approach which assumes that tasks and agents are, at least to some degree, interchangeable – with multiple homogeneous sub-entities capable of the same task. This will not be the case in the general formulation, where tasks may have to be undertaken by particular specified agents due to their equipment specialism, and a swarming technique is unlikely to be useful for this. Understanding which agents end up with which distribution of goals, and the

mechanism for managing this allocation, should help the system designer understand the behaviour that the different agents subsequently display.

As a basic template, to provide a baseline to show that the concepts and mathematical framework developed thus far can be practically applied to a complex multi-agent system, one can logically apply the MDP construct to this generic resource management problem on local and global levels:

- **Locally:** (the agents themselves)
    - **States:** Suppose there are *n* 'types' of resource which an agent can provide. The state of each agent could then be described by an *n*-vector, with each element describing either the quantity or proportion of resource available.
    - **Actions:** An action by an agent is the provision of resource to a task. This could be represented as a quantity or proportion of the available resource to be 'spent'.
    - **Rewards:** A simple reward scheme would be for an agent that fulfils a requirement to receive a positive reward of, say, +1 for each requirement fulfilled. Maximum attainable reward should then correspond to the fulfilment of as many requirements as the agent's resources allow.
- **Globally:** (at the system level)
    - **States:** At the system level, the pertinent pieces of state information are the number or proportion of requirements fulfilled, and the overall capacity of the system. In the simplest case, this could be represented by a 2-vector containing scalar values for these proportions. However, this does not contain any information about the resource distribution. A more detailed alternative could be a cellular automaton, with cells for groups within the system and separate 2-vectors for the state of each cell.
    - **Actions:** The action here, in the simplest formulation, would be an intervention or overrule of individual agent(s). This would be represented in the same way as at the local level (which conveniently should make the algorithmic implementation simpler).
    - **Rewards:** The system could receive rewards, say, of -2 for each requirement accepted but not fulfilled and of -1 for every 'exhausted' agent at each time step. Maximum attainable reward should then correspond to fulfilling all user

requirements as expediently as possible, but by balancing the load across agents where possible. If using the cellular automaton approach, these rewards could be for each individual cell, and the total reward for the system would be the sum of rewards over all cells.

One can envisage many possible future applications for complex multi-agent systems functionally similar to this generic template. Indeed, some of the more challenging problems that might be envisaged would arise when autonomous agents of all different kinds are part of larger deployment which includes combinations of, for example, autonomous, or least semi-autonomous, vehicles, communications systems, sensor platforms, or semi-automated operations management tools. The purpose of introducing this template is not because it is a ready-made solution for all problems, which it clearly will not be, but to build confidence that there may be recurring design patterns that can assist the system designer in the construction of control mechanisms for such heterogeneous deployments.

5.8 Local and Global Behaviours

Perhaps the biggest challenge with understanding the behaviour of systems lies in understanding how the behaviours of the individual components 'fit' as part of the larger system. With complex systems, by definition, this is much harder because of the differences between local and global behaviours. The key challenge for this work is to demonstrate that the mathematical framework for decision making (and, by extension, to system control) can account for complex phenomena, provide a basis for understanding it, and potentially control or at least constrain its consequences. It is proposed that the approach developed can fulfil these aims after some further modifications, and it is this that is explored in this section.

5.8.1 Emergence

It should not be assumed that all components of the system are assigned complementary and consistent goals, even if that is what the system designer intended to do and thinks they have done. Sometimes, it may not be possible even to assume there was any such intent, if the system has come together as an ad hoc composition of elements originally designed by different people. Consequently, it is important to understand how complex emergent behaviours are accounted for within the proposed framework. Understanding the underlying reason for the

emergent behaviours may provide a basis for understanding how to limit or constrain them, or at least to understand the impact they might have.

The key is to understand and qualify what the different goals of the individual agents are, and how they impact on the wider system. This challenge is best illustrated through a simple example, which is intended to illustrate this phenomenon in a recognisable, albeit simplified, autonomous vehicles scenario.

Imagine there are a number of deployed Unmanned Air Vehicles (UAVs) on a surveillance assignment over a particular operational area. The commander requires a redeployment of these UAVs, to concentrate on two particular locations, *x* and *y*.

- All systems west of line L are instructed to fly to location *x*.
- All systems east of line L are instructed to fly to location *y*.

Whatever the actual implementation of the systems onboard the UAVs, the decision making process can be represented as an MDP provided that states, actions and rewards can be defined in a suitable way, at whatever level of state space granularity is appropriate. For example, for the UAVs to the west of L assigned the goal of flying to location *x*:

- The state of each UAV could be its actual geographic location. This could be a rather large state space, depending on the level of granularity demanded. A simpler alternative would be a cellular lattice covering the area, with state then being the index of the currently occupied cell. Other alternative state space structures can be envisaged.
- The action taken is the direction in which the UAV chooses to move. This could be given as an actual vector, but a more tractable alternative might be, with the cellular lattice state space as an example, a discrete step to one of the neighbouring cells.
- The reward received by the system should reflect the goal the system is meant to achieve. The goal here is to travel to point *x*, so a simple reward scheme would be to give the system a negative reward, say -1, for any action which takes the system away from *x* (to incentivise travel towards *x*, as opposed to simply sitting in a holding pattern and not moving away from *x*).

The maximum attainable reward (of zero) is achieved so long as the system is continually closing on the target. If time is of concern, then applying a discount factor of less than one in the Bellman value equation will have the effect of reducing the expected return as the number of time steps increases, with the maximum attainable return achievable by following the

shortest possible route to the target (typically the direct path). There are of course many possible variations on this basic model, for example to visit waypoints, avoid obstacles, or arrive at a specified time, but these are not required for the simple example discussed here. These are all fundamentally instantiations of the basic autonomous vehicle navigation problem covered by Watkins [67], Sutton and Barto [59], the author's precursor work [23], and extensively in the wider literature.

A similar model could be applied to the other set of systems lying east of L and set the task of flying to location *y*. One might expect that this completes the task. Both sets of systems are assigned goals, and suitable onboard decision making software should be able to determine that the optimal policy to achieve these goals is to travel in a direction that reduces the distance between the system and its assigned target. Indeed, this might be exactly what does happen. But a naïve implementation of these models does leave a potential hole.

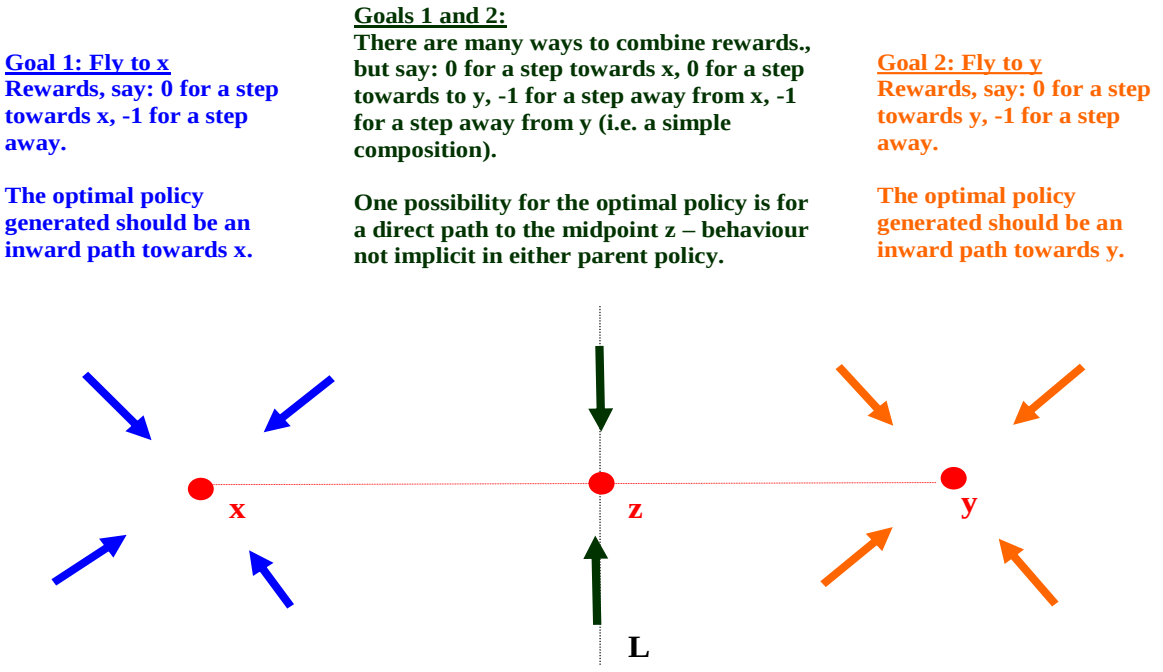

Figure 22 – An Example of Simple Emergence

Consider a system sitting on border L. All will depend on the precise implementation, but, if the original programmer did not fully think through the nuances of his or her code, then a valid interpretation of the situation is that a system located on border L is assigned both goals. In this case, the reward a system earns will be a composition of rewards relating to each of the

goals. If the combined reward were of the simplest form (adding the two rewards together) then hopefully it is clear that the optimal policy will be for a direct approach to the midpoint between *x* and *y* - the system would incur no negative rewards as it is approaching both locations simultaneously, and once it gets there the optimal course of action is to stay there, since anything else moves away from at least one of the two targets and incurs a cost.

Clearly this behaviour is not what the system was intended to do, nor presumably what it was expected to do, yet it could arise in a relatively straightforward way from a naïve implementation. If systems have multiple goals placed on them, with 'mixed' incentives flowed down to them, then this fused reward scheme generates a new policy, and the optimal policy for this system might be the best possible compromise rather than the desired outcome. The generated policy is observably different to the parents, and the observed behaviour is not implicit in the policies expected of either set of systems.

Referring back to the general discussion on the nature of emergence in Chapter 3.8.2, note that this is an example of the so-called weak emergence rather than strong emergence, because the emergent behaviour is reducible to the behaviour of the components (notwithstanding the earlier assertion that there is no substantive difference between weak and strong emergence other than perspective).

Admittedly this is an artificially simple example:

- There are fairly obvious ways of fixing this to get the behaviour intended (for example, specify explicitly whether on the line L means go left or go right).
- The operator or his commander may not be concerned if *z* and all points travelled through on the way to *z* are acceptable states, and the affected systems are not otherwise required.
- Some may not count this as complex or emergent behaviour in the commonly used sense of the word, as it is simple enough that one might reasonably have been expected to foresee it. Nevertheless, this is an emergent behaviour under many interpretations of the term, including under the working definition used within this thesis given in Chapter 3.8, and in more complicated models such situations could certainly be far less foreseeable and hence cause more significant problems.

This example is intended to illustrate that unintended, emergent behaviour can result even from simple compositions of MDPs, and thus it is not hard to imagine it occurring in more complex

setups where it was not obvious to foresee, not clear why it happened, and could easily cause deviations into undesirable states. Key to the proposed approach to understanding complex system behaviour is to understand the distribution of goals and actions throughout the constituent parts of a complex system, or the *goal profile*. This is vital because, assuming one adopts a goal-driven approach to behaviour, goal distribution determines reward distribution, which determines policy and hence behaviour. Understanding how the goals and rewards are distributed should explain why the behaviours are as they are.

More formally:

- System A is part of a subset of entities that have a goal G_A.
    - There would have to be a reward scheme that incentivises actions that support or progress goal G_A.
    - States, actions and rewards combine to produce a policy π_A that maximises the expected return by progressing System A towards goal G_A.
- System B is part of a subset of entities that have a goal G_B.
    - There would have to be a reward scheme that incentivises actions that support or progress goal G_B
    - States, actions and rewards combine to produce a policy π_B that maximises the expected return by progressing System B towards goal G_B.
- System C is part of both subsets, and has to achieve both goals G_A and G_B.
    - There would have to be a reward scheme that incentivises actions that support or progress both goals G_A and G_B (for example by using some kind of combined reward function, see Chapter 5.6.7).
    - A composition of states, actions and rewards combine to produce a policy π_C, but note that there is no reason to assume this will be the same as either of its parent policies π_A or π_B.
    - Consequently System C, similar to both System A and System B, could have a notably different policy from either System A or System B.
    - Since policy drives behaviour, it should not be unexpected if System C's behaviour is observably different from the behaviour observed from either of System A or System B.

The purpose of this apparent digression is to demonstrate another property of MDPs that make them suitable for this task – emergent behaviour can appear not through magic but as a logical consequence of the hierarchical MDP architecture already developed. It might be premature to declare this as a definitive answer for how to understand emergence, but this is a start towards understanding what it is about system behaviour that can cause complexity, which should help with understanding and responding to it.

5.8.2 Communities

How does connecting the distribution of different types of goals and rewards to different combinations of agents within a system help? If one continues to assume goal-based decision making by individual agents, that the number of such agents is finite, and, critically, that there is no assumption of homogeneity (systems will not all have the same role; they may not even be the same types of system), then it is proposed that:

- If the system can be divided into subsets of components (unitary subsets if necessary) with common goal profiles, and hence common policies, this information could be used to identify where there is a possibility of emergence, and where there might be a need to reassess behavioural predictions and possibly intervene (i.e. an early-warning system).
- Even better, if each subset with a common goal profile and a common policy can be represented as an MDP, it should be possible to directly apply the hierarchical MDP framework to model the behaviour of such systems (i.e. a predictive tool, which could mature into a control mechanism, although that is beyond the current scope).

The number of possible groups of agents with common goal profiles is potentially large (there are 2^n possible groupings of n agents, meaning an exponentially increasing number of possibilities). However, with a finite number of agents, the number of possible subsets of agents within the system, including the empty set and the whole set that need to be included for analytical reasons, will also be finite, albeit large (for the reader familiar with set theory, what is being describing is the *power set* of the set of components).

It is assumed that when goals are assigned to the system, they are only assigned to particular agents within the system (which will obviously depend entirely on the context). If there are multiple goals placed on a system, then some of these goals will be passed down to the relevant subset of agents. Looking at the problem the other way around, one ends up with a collection

of agents with different goals. If agents have the same combination of goals placed on them, or a common goal profile, they form a *community* (where a component with a unique goal profile is defined as a unitary community).

Figure 23 below provides a graphical representation of goal assignment and community formation within a collection of agents.

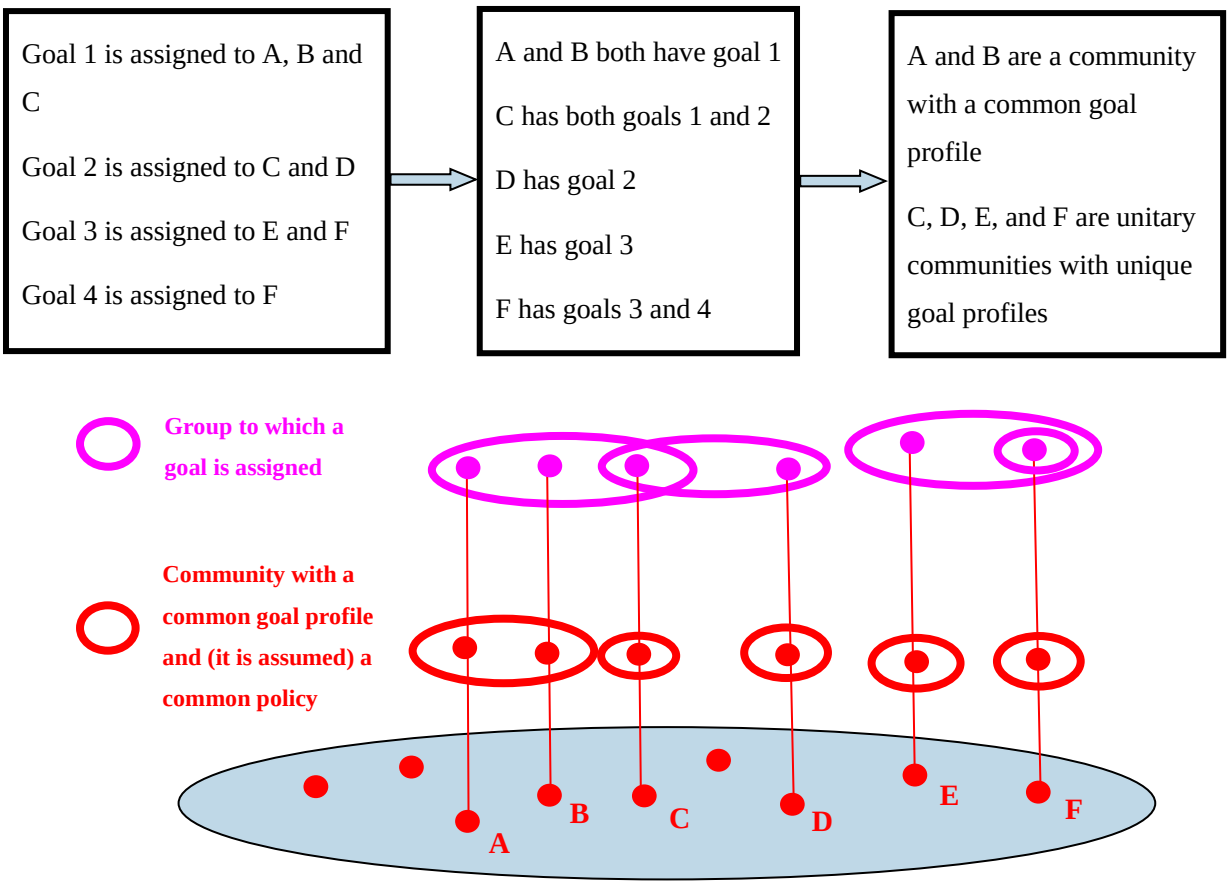

Figure 23 – A Graphical Representation of Agent Communities

How does the grouping of agents into communities of components help? Goals will have to be passed to agents by some sort of indexing mechanism. Exactly what this means will be context specific; however, there will be some sort of logical association of goals and rewards to systems (and not necessarily an *a priori* prescribed association – there is no theoretical reason preventing agents changing their group memberships, if this is permissible within any other system or environmental constraints). If this indexing can be reversed then, provided the empty and whole communities are included in the model, what will have been constructed is a

topological space. Working from a formal mathematical basis should make it easier down the line to apply analytical techniques to the observed behavioural data, such as applying metrics for the magnitude or separation of sets, or other potentially useful statistical tools for classifying behaviours, or for measuring the boundedness of behaviour.

However, the most important reason is that identifying where goals and rewards have been composed indicates where an observer of a multi-agent system should not assume optimal policies are as they might have been presumed to be. This information is vital if one is to understand observed behaviours and to know what might be possible if wishing to influence subsequent behaviour. It might also be a useful tool for behavioural assurance to be able to identify where a system is vulnerable to emergent behaviour, potentially allowing the prediction of the possible consequences of any emergent behaviour, but at least helping to predict the bounds of any such behaviour.

5.8.3 Decomposing a Complex Multi-Agent System into Communities

For completeness, this section provides more of the mathematical justification for investigating the topological communities model.

The domain of interest of a system is defined by variables such as 'those elements involved' and the 'physical' and 'logical' problem domains. Exactly what these variables are depends entirely upon the type of system and the operational context in question. The 'position' of an element within the domain could reflect its geographical position, but will also represent its current mission state, or its identity, or any other pertinent information that describes its state with respect to the variables of interest. Note that the 'boundary' of the domain may also evolve with time.

The domain of interest is *covered* (using this phrase in the analytical sense from topological theory) by a set of 'missions' or, in the common MDP terminology, goals. Each mission represents an objective to be achieved within that subset of the domain.

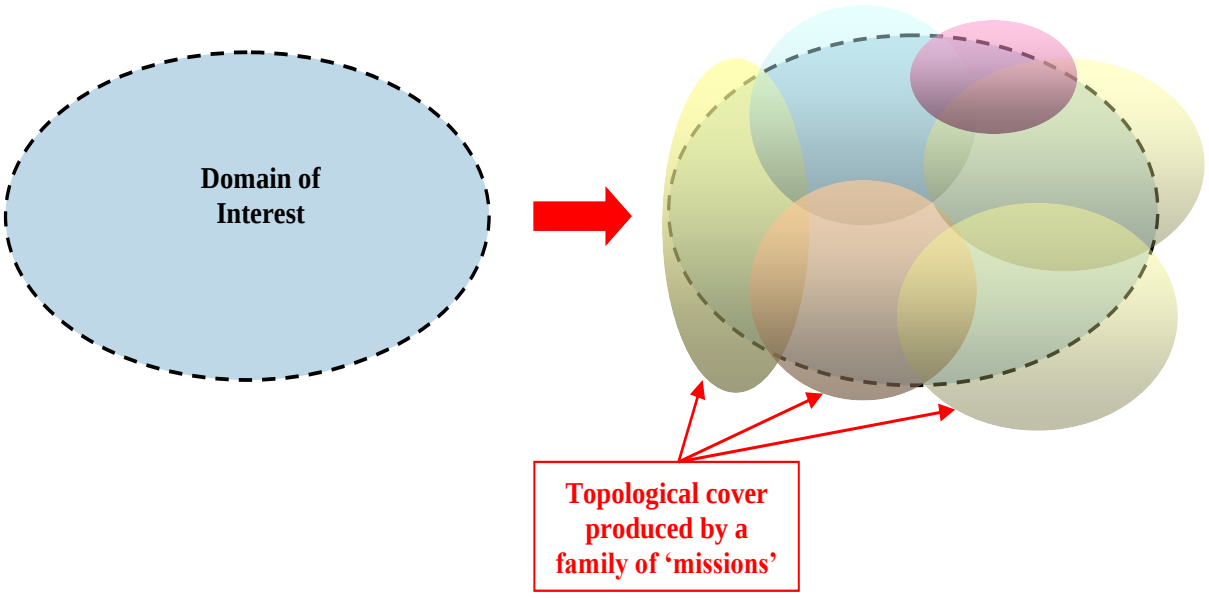

Figure 24 – Topological Basis for Agent Communities

Missions may overlap within the domain (which does not just mean in the geographical sense), and each element may be assigned to multiple missions. This has already been described within the general discussion of various application areas within Chapter 2.5 and within the developed example of the coupled decision making of agents within production line automata in Chapter 5.5, illustrated via Figure 17 and Figure 18. Certain goals end up being shared between different agents in physically separate parts of the system, for example the agents responsible for manoeuvring the two robot arms concurrently are in separate physical systems but are operating towards the same goal. They are forming a community within the wider system, which is a different association that does not correlate with their physical installation or their place within the decision control hierarchy.

To develop this further:

- Each 'mission' set is implemented as an idealised MDP with a number of state variables $\{x_1, x_2, x_3, \dots\}$.
- Each variable assumes a value determined, at any given instant, by its 'location' in the domain.
- This process is repeated until the desired 'resolution' is achieved.
- At some point, the 'idealised MDP' should become a realisation of a decision layer in a physical system element.

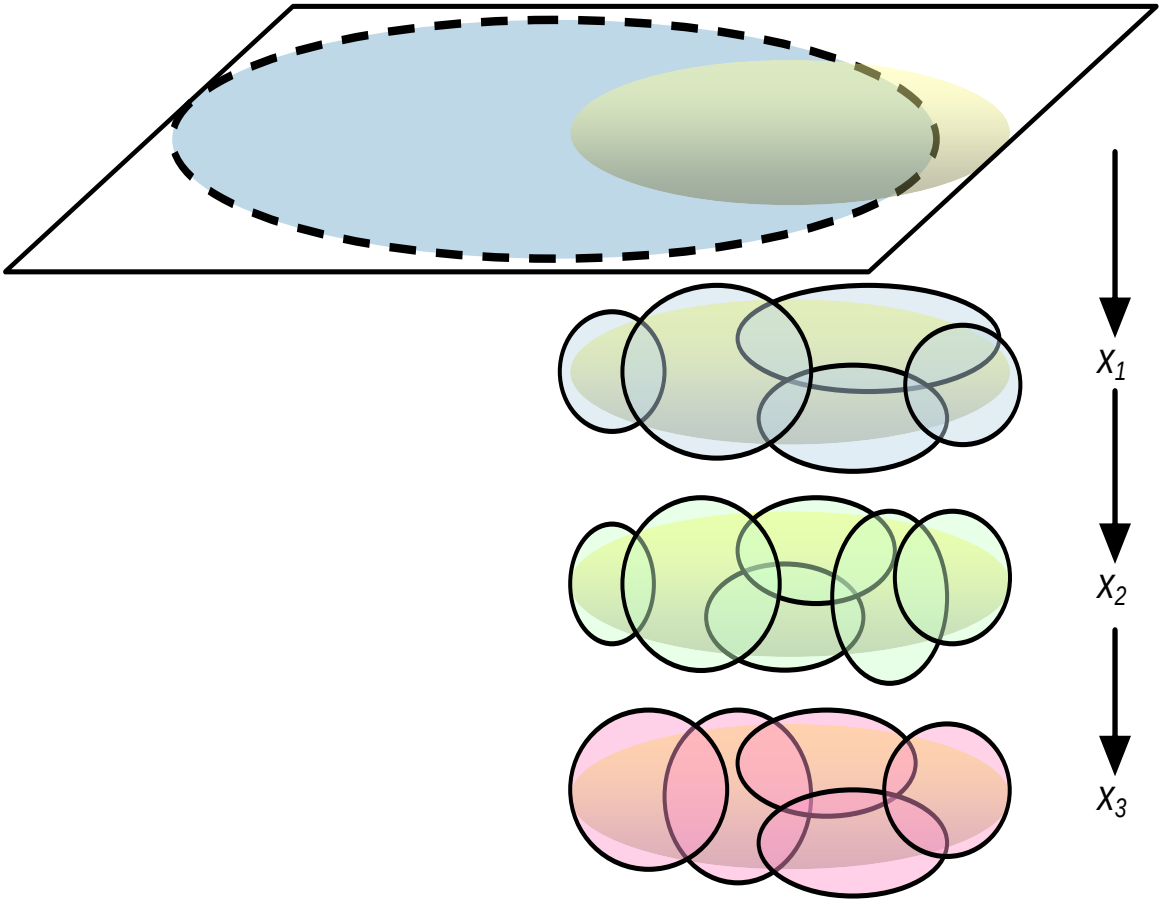

Figure 25 – Increasing Resolution of the Topological Model

This process can, potentially, provide an extremely finely grained representation of the state of all the agents within the system, and of how this determines the policies of these agents and consequently the behaviours that emerge. An additional benefit from the communities model is that, combined with some aspects of the hierarchical MDP model already developed in the earlier chapters, many of the characteristics of the different organisational paradigms discussed earlier in this thesis in Chapter 3.3 would be reflected in the mathematical model one would use for the decision making processes.

The purpose of this discussion is to bring the argument back around to the consideration of the layers and dimensions of decision making within complex multi-agent systems described in Chapter 5.5. The choice of MDPs for the precursor work was motivated by their practical attributes and amenability to iterative solution, which is appropriate for many autonomous systems applications. The motivation for researching these topics subsequently was that MDPs might have been more suitable a modelling paradigm than ever originally appreciated at the

time of the precursor work, if a deeper link could be established between hierarchies of MDPs linking communities of agents within a multi-agent system, and the topological structure. This realisation suggests a more rigorous analytical approach to the vexed question of how to prove complex multi-agent system behaviour might not be fanciful. These aspects unfortunately fall into the second half of the proposed research and thus became beyond the scope of this PhD. However some of the anticipated next steps are described with Chapter 9.2.5 amongst the conclusions and recommendations from this thesis, and pointers to where it was expected this work would lead are to be found within Annex C.

Finally, it should be noted that whilst this approach to community detection is presented in order to complete the description of the mathematical background established within this PhD, which is believed to be a novel approach to the multi-agent system domain, it is not claimed that this is the only possible mechanism for community detection within networks. The innovation in this PhD is to link the formation of communities of agents within networks with the rigorous approach to modelling the decision control hierarchy, however analytical approaches to community detection is not without precedent. Newman's work on modularity maximisation [49] is one existing example which takes a different approach from that proposed here, focusing on identifying nodes in graphs with a number of edges (potentially representing agent's communal associations?) that have spiked above equivalent nodes in random graphs, suggesting some communal event might have occurred. Whether techniques such as these can be linked to or otherwise support the proposed approach is an area deserving of future research.

Chapter 6

6 Parallel Decision Making and its Application to Interactive Autonomous Systems

6.1 Context

When a system controller has to make a decision, it is often not a singular decision. If a system has multiple simultaneous goals, the consequences of a decision may affect the system's progress towards more than one of these goals. Conflict can then arise when decisions that are beneficial towards one goal are detrimental towards another.

The development of a hierarchical decision progress begins with the simpler formulation introduced in Chapter 5.5 and named the parallel decision process. The concept is that two or more separate decision processes take place simultaneously, the desired action from each process influencing the others. These processes could either arise from distinct system components with a resource conflict, or they could arise in a single system that has multiple mission goals that cannot be achieved simultaneously, requiring that a choice be made between them.

A simple example might serve to clarify this. Consider the control architecture of an industrial robot arm with joints at 'shoulder', 'elbow' and 'wrist' in a scenario where obstacles constrain the movement. (This is a simplification of the example from Chapter 5.5.1, leaving out the 'applicator' tasks therein for simplicity here.)

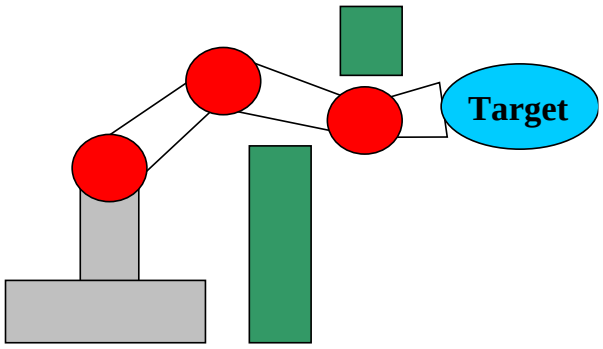

Figure 26 – A Visualisation of a Robotic Arm

Unless any obstacles are placed in particularly difficult positions, there is no single correct solution for the angles required at all three joints. This suggests that two architectures are possible: a top-down approach in which a central controller performs all calculations and dictates the control actions to be applied; and a bottom-up approach in which top-level 'goals' are satisfied by independent controllers at each joint. For example, the 'wrist' controller, knowing the dimension of the 'hand' and the location of the target, might define a range of acceptable locations for the joint. This is propagated back to the 'elbow' controller, which determines an acceptable range for the elbow joint, and so on.

This bottom-up approach is an example of a parallel decision process where each controller makes its own decision about its position, based on the overall goals and the behaviour of its partners. Even in this simple scenario it is easy to speculate on how different performance goals may compete. Certain solutions, for example, might cause greater wear to occur at a particular joint, but result in lower overall power consumption. The optimal control policy for all three controllers must find an acceptable balance of the relevant factors.

One may notice the contrast between the top-down and bottom-up approach, where the former requires one controller to select multiple joint positions contributing to a single overall state, while the latter features multiple independent controllers considering the expected range of possible positions when choosing their own individual positions. Referring to the hierarchical dimensions discussion in Chapter 5.5, here one can see that the two different approaches to the control problem correspond to different dimensions of the decision hierarchy. Discussion here focuses on the bottom-up approach to control, corresponding to the parallel perspective on decision making (one action towards multiple goals), because this approach is the one clearly relevant to problems featuring multiple interacting systems. This chapter presents a detailed worked example showing how a parallel decision process can be modelled with parallel MDPs,

and develops the necessary mathematics to enable solution of this with relatively minor modifications to standard MDP solution techniques.

It should be noted from the outset that there is no theoretical limit to the number of parallel decision processes. The problem will be with the inherent complexity that will arise in larger models, which could potentially increase exponentially with the number of decision processes – at worst, one could end up with an autonomous control equivalent to the n-body problem. A careful construction and segmentation of the control model should limit these difficulties but, to illustrate the core principles, in this initial treatment the focus is on the simplest two-process problem.

It should also be noted that the parallel decision process with two goals is not an instance of game theory, as the autonomous system acts in response to its state and environment rather than to a second player, but it may help to think in terms of simple games (note though they are not generally going to be zero-sum games, where the net gain or loss to the all decision makers balances out, nor are they likely to be symmetric games, where decision makers have mirrored goals). The worked example developed in this section has obvious relevance to the world of autonomous vehicles, namely having a goal to reach a specified target location without being intercepted. However, the inspiration for this example actually came about by considering the game theory analogy. The example is developed from a problem typical of many simple computer games: get to the exit without getting caught by the monster. It can be seen that the two problems are structurally and logically similar, and not dissimilar from the many analogous problems in the real world, dynamic obstacle avoidance the most immediate and thus the motivating example behind this section. [Note that despite providing the inspiration for the worked example that follows, the game theory approach itself has not been considered in any further detail, for the reasons given in Chapter 4.3.3.]

The following chapters detail the two decision processes that occur concurrently throughout this task and construct the MDP that will model this decision problem. Note that whilst the example developed in this section is of a parallel decision problem at the strategic or planning level of decision making (in the language of Chapter 5.5), a similar approach could be applied to the parallel decomposition on any other decision control layer, an obvious possibility being to have two separate and distinct components of a system in conflict over some limited or constrained system resource at a lower level control or execution layer, such as the positioning of the robot arm joints in the example illustrated by Figure 26.

6.2 A Worked Example: Dynamic Obstacle Avoidance

An example of parallel decision making is provided by the familiar autonomous systems problem of dynamic obstacle avoidance. Suppose an autonomous vehicle has a goal to reach some specified target location, and that there is another entity in the local area blocking its path.

The autonomous vehicle has two mission goals – to reach its target location and to avoid a collision with the other entity. The optimal policy for achieving the former could cause conflict with the latter when the other entity is on the desired path to the target location. The system controller has to make a navigational decision that respects both mission goals.

In Figure 27 below, observe a simplified representation of this problem. The target location is in a passageway or channel, with the (red) autonomous vehicle initially at one end. It has a mission goal to reach the (green) target location, and the optimal policy for that goal in isolation is trivial. However, the presence of a second entity in the passageway is a problem – without prior knowledge of the control policy being followed by this other entity its movements may be unpredictable, and it becomes a dynamic obstacle that the first vehicle must navigate around.

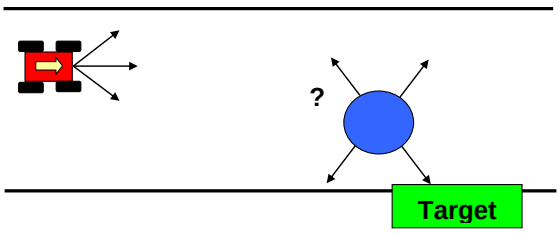

Figure 27 – Dynamic Obstacle Avoidance

A navigational decision made at any one time by the controller of the autonomous vehicle has to contend with the fact that the controller does not know what the relative position of the two systems is likely to be after it has made its move, because it does not know in advance what the probabilities of particular moves are for the other entity.

Since each interacting entity is a part of the environment for the other (since neither is under the direct control of the other) this sort of problem can be seen as a Level 2 or Level 3 control problem (referring to the levels of control as described in Chapter 4.5), depending on the degree of sophistication and flexibility required, and is suitable for an MDP-based solution.

6.2.1 Decision Process One: Reach a Target

The first step is to introduce the two goals of the parallel decision process individually. Firstly, imagine an autonomous vehicle has to reach a target location X. For this simple example, one could imagine this target is the only bridge across a river which the vehicle is required to cross, or a location where the vehicle needs to hold position in order to receive a signal. In the computer game context that inspired this example, the aim could be to get the player's avatar to the exit and on to the next level. Rather than tackle the continuous control problem, for computational simplicity one can model the environment as being a local area (for example a room) divided into a grid, and model the movement of a system as steps between cells.

For this simple introduction, a relatively course grid will be used. A typical grid dividing up a room is illustrated below, with cell X representing the target point the autonomous system is trying to reach, the exit from the room in this case. An arbitrary cell in the grid will be the starting position – any cell will do, except the cell containing point X (which would result in a rather trivial example). In Figure 28 below, all the other cells have been labelled with a state number which uniquely describes each grid position. The target cell is therefore cell 24, and in the language of Markov chains and MDPs this is a terminal state since no further state transitions will occur following a transition into cell 24.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	**X**
25	26	27	28	29	30
31	32	33	34	35	36

Figure 28 – Discrete Enumerated State Space

An MDP model for this situation should have the autonomous system presented with a set of actions in the form of moves between cells, with the ultimate goal of reaching the exit – cell 24 in this case. At each time step, the state of the system is simply the cell number the system

is currently in and a simple set of actions to choose from would be a choice of moves of up, down, left and right – to keep the problem simple, diagonal moves, jumps, and stationary moves (standing still) are not available as options, although these or other moves could be included if so required. If the system is up against a wall, a move into the wall leaves the state unchanged (so a move downwards from cell 34 leads to cell 34).

Transition probabilities are fortunately trivial in this model – each action leads to one particular state with probability 1 and any other state with probability 0. A system starting in cell 9 and choosing right will definitely end up in cell 10 with no chance of ending up anywhere else. Although the transitions are trivial in this case, it is only because this is an artificially simple example for illustrative purposes. In general, actions could cause one of several different possible state transitions, with corresponding transition probabilities between 0 and 1, and the state space dynamics are not necessarily certain (i.e. the transition model can be non-deterministic). An example of this will be provided by the second constituent part of the decision process.

The final component of an MDP that has to be defined is its reward scheme. For each state-action pair (that is, for each current cell number and selected next action), the designer will need to define the numerical reward that corresponds to how well the autonomous system is doing regarding achieving its goal, remembering that the expected cumulative reward (expected return) should be maximised by selecting actions supporting and ultimately achieving the goal. There are of course many ways to define the rewards, and the choice of reward scheme affects the performance of the decision process.

For this model, a simple reward scheme would be to reward each move with the negative of the minimum number of moves away from the exit which the move leaves the system requiring. Thus a move into cell 24 has reward 0, a move into cells 18, 23 or 30 has reward -1, a move into cells 12, 17, 22, 29 or 36 has reward -2, and so on through to a move into cell 1 having reward -8, being the furthest (8 steps away) from the exit. Then the maximum (least negative) return is obtained by getting to the exit in the fewest steps possible, and any series of actions that gets the autonomous system to cell 24 with minimum cost is an equally optimal policy.

For example, the optimal policy starting from cell 9 is for a series of 5 moves, 3 moves right and 2 moves downs which gives a return of -10 by the time the system reaches the exit. From the reward map (see Figure 29 below), it is clear that any combination of these first five moves is equally optimal for this model – they all take 5 steps with a return of -10, whereas any other

series of moves to cell 24 takes more than 5 steps and gives a poorer (more negative) return than -10. Note though that this is a particular property of this simplified example and would not generally be true. One of the possible optimal routes is highlighted in Figure 29 below:

-8	-7	-6	-5	-4	-3
-7	-6	**-5**	**-4**	-3	-2
-6	-5	-4	**-3**	**-2**	**-1**
-5	-4	-3	-2	-1	**0**
-6	-5	-4	-3	-2	-1
-7	-6	-5	-4	-3	-2

Figure 29 – The Reward Map (with an optimal path highlighted)

Each state (in this case each cell) has an associated value, which is the expected return from that state. This is obtained by using the Bellman value equation as defined in Chapter 5.6.2:

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'}\left[R^a_{s,s'} + \gamma V^{\pi}(s')\right]$$

where here states $s, s' \in S = \{1,2\ldots.36\}$, actions $a \in \{up, down, left, right\}$, a transition from s to s' following action a has transition probability $P^a_{s,s'}$ (which here is 0 for all bar one of the s' for which it is 1) and a corresponding reward $R^a_{s,s'}$, with discount factor $\gamma \in [0,1]$ and the probability of choosing action a when in state s defined by the policy π such that $Pr(a|s) = \pi(s,a)$. Note that terminal states have a value of zero by definition (as there will be no further actions or transitions of any kind from a terminal state, and hence no future rewards), so $V^{\pi}(24) = 0$.

As an example, assuming that one is equally likely to select each of the four possible actions, so $\pi(s,a) = Pr(a|s) = \frac{1}{4}$ for each a, then the value equation for state 18 will be:

$$V^{\pi}(18) = E_{\pi}\{R_t|s_t = s\} = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'}\left[R^a_{s,s'} + \gamma V^{\pi}(s')\right]$$

$$= \frac{1}{3}[V^{\pi}(12) + V^{\pi}(17) - 5]$$

since $R_{18,12}^{up} = R_{18,17}^{left} = -2$, $R_{18,18}^{right} = -1$ and $R_{18,24}^{down} = 0$ from the reward scheme defined above, with $\gamma = 1$ for simplicity and $V^{\pi}(24) = 0$ by definition. Repeating for each of the other states will produce a set of 36 simultaneous linear equations for the state values (effectively 35, since the terminal state's equation is trivial). The solution could be computed directly, admitting a set of state values as the solution to the simultaneous value equations. When the model is not artificially simple like this example, the solution can be found by estimating an initial value for each state (apart from the terminal state, which has to have value zero) and using dynamic programming techniques to search for a solution to the system of equations. This would be the technique of choice as the dimensionality of the state space increases and direct calculation becomes impractical.

Problems like this have been well explored in the MDP literature and a full solution does not need to be re-produced here. It is sufficient to note that with the framework for calculating the value functions established, the solution to the decision problem is the optimal policy which determines those actions which cause transitions to the highest-valued reachable states. In this example, this means the policy determines the direction that is expected to take the autonomous system to the state with the greatest (or least negative) value amongst the reachable states (which in this case means one of those cells immediately next to the current cell). It is hopefully clear that optimal (or equally optimal) paths such as that highlighted in the figure above will be the result, as one would expect. Hopefully, it is also clear that the rather obvious answer arises because of the simplicity of the problem tackled here; such methods will also solve far more complicated decision problems in a similar way.

6.2.2 Decision Process Two: Avoid Another Entity

The navigation task described in the previous subsection is by itself quite straightforward. In practice, it can be complicated by not allowing the autonomous system full knowledge of its state, putting obstacles in its way, or other such encumbrances. The focus here is on parallel decision making, so the complication introduced next is a second agent that the first system must try to avoid. This situation is analogous to, for example, avoiding a hostile entity like the 'monster' in a computer game, or real world examples such as evading a missile intercept, or evading detection, or navigating around an uncooperative or erratic vehicle.

Parallel decision processes means that at any given time step the autonomous system has decisions to make in each process simultaneously. If a second decision process took place over

a similar time interval but chose actions at different points in that interval, the choice of action at any one time could be based on only the one process, and any direct conflicts between the two processes' choices of actions are circumvented. Since the ultimate aim would be to have continuous decision making, where simultaneity is unavoidable, common time steps and simultaneous decision making are assumed throughout.

Consider the second decision process in isolation, temporarily ignoring the navigation task. Pursuing the computer game analogy, one can imagine the system is being chased by a monster (or more generally, a second agent outside of the first agent's control). This can be framed as a decision process with a goal to avoid the monster.

In this problem, states are not best thought of as fixed positions relative to a grid, because as the monster moves around the values of specific positions vary. A position that at one time is a long way from the monster with a relatively high value associated with it could, at some later time, contain the monster and be the worst position possible. For simplicity, the assumption is made that the system can see the whole of the surrounding area and knows the position of the monster (i.e., full observability of the state space). As such, one option for the state space representation is to define the system state as being the number of steps away from the monster that the system is at that time, using a state space representation that is relative to the other entity. Note that as the total number of states has to be finite for standard solution methods to be useful, there needs to be a cut-off point, a state that means the monster is a certain distance or more away. However, it is not necessary to consider this here, since when the two decision processes are combined it will be apparent that a maximum separation is imposed on the model anyway by the choice of state space geometry.

The choice of actions is again a choice of where to move next (up, down, left or right), and the reward associated with each move depends on how much closer or farther away from the monster that move leaves the system. Thus one can define:

- States $s \in \{S0, S1, S2, \dots\} =$ { In the same cell as the monster, 1 move away from the monster, 2 moves away from the monster…} – these states are relative positions.
- Actions $a \in \{up, down, left, right\}$ – to keep the number of variables down (and for consistency with the navigation problem), diagonal moves, jumps, and stationary moves (standing still) are forbidden.

The Markov property is obeyed because one's relative position to the monster at previous times is irrelevant to where it is now, at least in simple homogeneous environments, with no

complications arising from local geography or other constraints imposed on movement. Thus this is an MDP, with rewards associated to actions set to encourage the system to keep its distance from the monster – fulfilling the goal of not encountering the monster.

An immediate concern with the reward scheme is to specify what precisely the system is expected to achieve. Not encountering the monster can be interpreted as either 'move as far away from the monster as possible' or 'stay a safe distance from the monster'. A reward scheme that merely punishes the system when it gets too close to the monster but does not differentiate between distances greater than some threshold would likely produce different behaviour than a reward scheme which gives greater rewards the further away from the monster the system gets. Indeed in the latter case, the optimal policy would clearly be to run as far away from the monster as possible, which may not be realistic. Looking ahead to when the two decision processes are combined, one will notice that the 'run-away' option is not appropriate here. What is required is a reward scheme to guide the system to follow the 'stay a safe distance from the monster' principle, a model for which is developed below.

It is important that the monster is moving. A stationary monster could be modelled with only minor changes to the model for the first decision process by, say, defining a zero transition probability for any move into the cell containing the monster, which defeats the object of having an on-going decision process. The aim here is to introduce an unpredictable element to the control problem – where the system can select an action, but the transition probabilities are no longer simply 0 or 1 as in the simple navigation case as the position the system will end up in relative to the monster after making a move is not known in advance.

This problem is illustrated in Figure 30 and Figure 31 below. The system is in a cell two steps away from the cell containing the monster. Two steps difference places the system in state S2. Note that a state could be regarded not as a single cell but as a collection of cells; state S2 is thus a union of eight cells. Figure 30 illustrates the two agents starting positions, the cells marked respectively green and red for the system and the monster.

The system now faces a choice of possible actions. The policy it follows has yet to be defined, but assume that for some reason a move to the right is selected. Intuitively, a move in any direction other than down will take the system to state S3, whereas down (towards the monster) would take the system to state S1. The problem is that the choice of action does not specify the outcome state by itself – the movement of the monster also influences it.

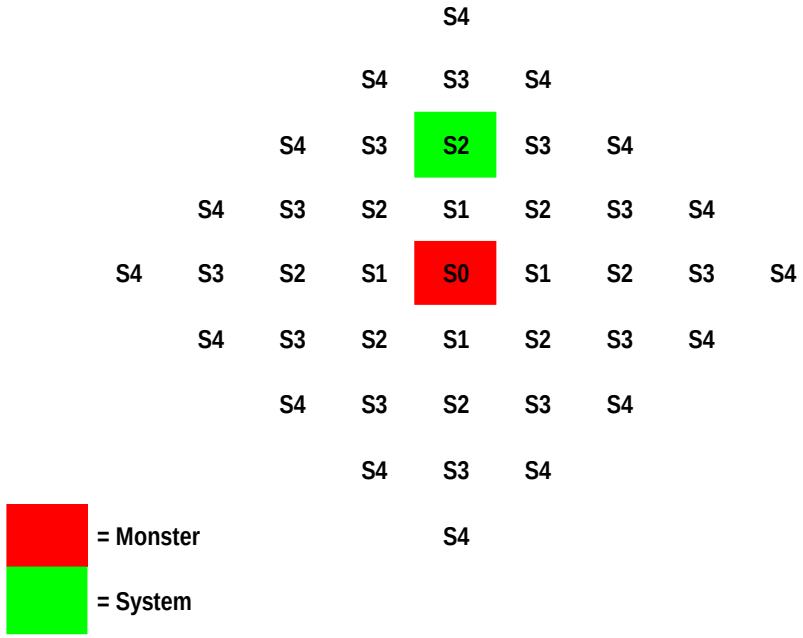

Figure 30 – Initial System State

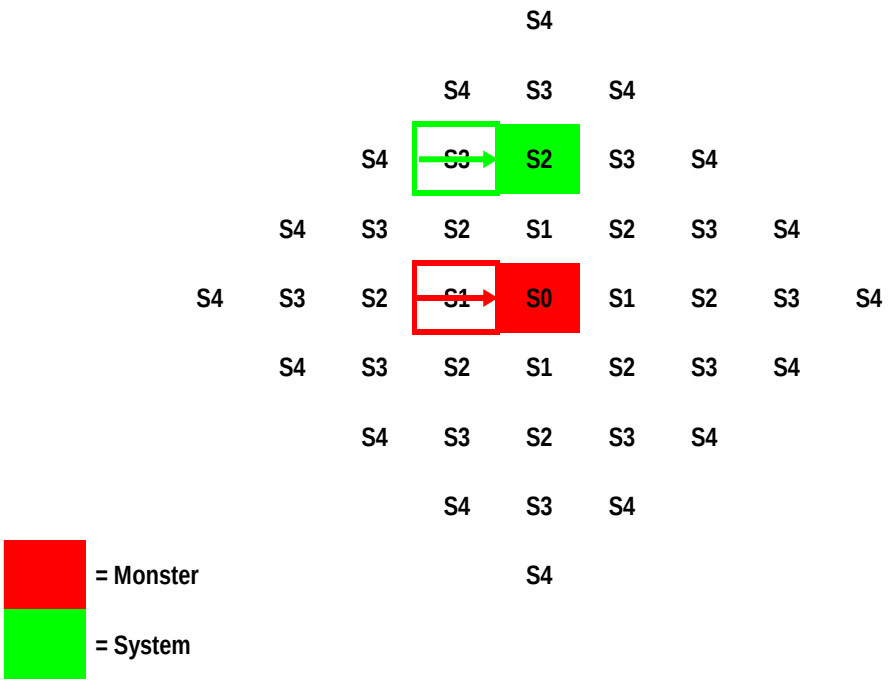

Figure 31 – System State After a Move Right From Both Agents

As Figure 31 illustrates, the move right does not place the system in S3, as might have been expected. If the monster moves to the right as well, it will still be only two steps away and hence the system is still in S2. Because the state is relative it changes with the movement of both entities. Thus the system will receive the reward for a transition from state S2 to itself, rather than the reward for a transition from S2 to S3 that might otherwise have been expected. Considering all four possible moves the monster could have made, a move to the right by the system may have looked like a move to S3, but in fact that was never going to be the case – it was a move to either S2 or S4 depending on the move made by the monster. With the additional assumption that the movement of the monster is unpredictable, this has become a stochastic decision process, with different transitions resulting as the environment changes around the system.

This model is a simple illustration of the problems that can arise when operating in dynamic environments. One solution is to specify a model for the environment – essentially a 'best guess' type of solution. An alternative is to use reinforcement learning techniques, so the autonomous system can learn for itself which are the best actions to take. If there is a defined policy behind the movement of the monster (a control policy unknown to or hidden from the system), then a good learning process should eventually identify it and the system can act as if it had known its opponent's policy to begin with. The benefit of this approach is that it does not require prior modelling information. The downside is that the system must have a learning phase (or at least a training phase) before it can operate optimally.

For the purposes of this current example, consider the simpler but potentially less realistic approach of assuming a known model for the control process of the other agent. Possibilities include assuming that the monster matches the movement of the system, or that it always heads directly towards the cell the system was just in. Alternatively, one could define a 'dumb' monster, just as likely to move in one direction as in any other. There are of course other possibilities where the monster can move in all directions but has preferences for certain directions, producing a probability distribution over possible actions. Whichever policy is chosen as most appropriate to a particular context, the sum of the probabilities of choosing the individual actions must be equal to 1, since the monster must choose one and only one of the possible options.

Once a policy and the resulting transition probabilities are defined, corresponding rewards have to be defined as well. As with any other MDP, the rewards should be chosen in such a way as to guide the autonomous system towards the desired goal, and the maximum return should be

gained from fulfilling the goal or, as in this case, the poorest returns received when failing (i.e. running into the monster). As discussed above, the goal here can be interpreted as 'stay a safe distance from the monster', and the choice of reward scheme should reflect this. One possibility for a reward scheme which does not reward running unnecessarily far away (and hence should not induce this behaviour) is presented below.

Transition into state:	S0	S1	S2	S3, S4….
Reward:	-10	-5	-1	0

Figure 32 – Example Reward Scheme

Following this reward scheme (or something similar), and using the defined states, actions and suitable transition probabilities, makes this a complete Markov Decision Process. As with the navigation problem, it is not useful to solve it explicitly here, but hopefully it is clear how such problems can be tackled. The resulting optimal policy solution defines the actions the system should take from each state in order to maximise its expected return and thus achieve its goal.

6.2.3 The Parallel Decision Process

The two decision processes outlined above can now be combined into a single problem. There will be two goals – to get out of the room and to avoid the monster. Clearly however the decision making becomes more complicated when a conflict arises between the goals of the two processes, such as if the monster is blocking the system's route to the exit.

This situation is illustrated in Figure 33, Figure 34 and Figure 35 below. In each figure, green represents the cell containing the system and red represents the cell containing the monster that the system has to evade. Note the way the state information (specifically the relative separation information) changes between time steps.

This is a convenient point to introduce the convention that states will be represented by *n*-tuples depending on the number of pertinent bits of information needed to describe the state of the system (as described in Chapter 5.6.7). In this case, this results in state pairs containing two pieces of pertinent state information – the cell number and the relative position to the monster. Thus, for example, the top left most cell of the grid in Figure 33 is state (1, S6) because it is cell number one (in accordance with the navigation problem) but its state relative to the monster is S6, representing 6 cells separation (in accordance with the evasion problem). Note of course

that this is merely one option for a state space representation; there are alternatives. Choosing an appropriate representation is a key part of the problem for the system designer (as discussed throughout Chapter 5).

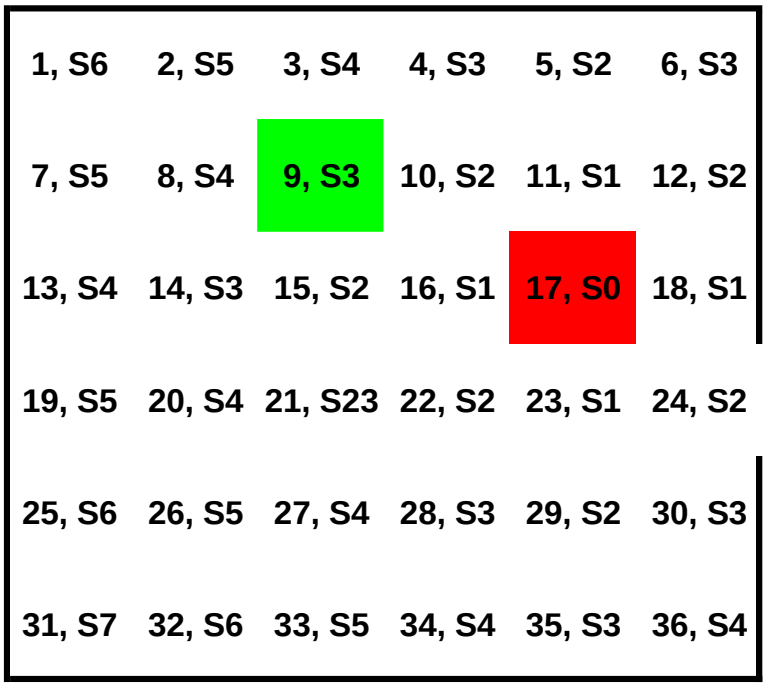

Figure 33 – Starting Positions of the Two Agents

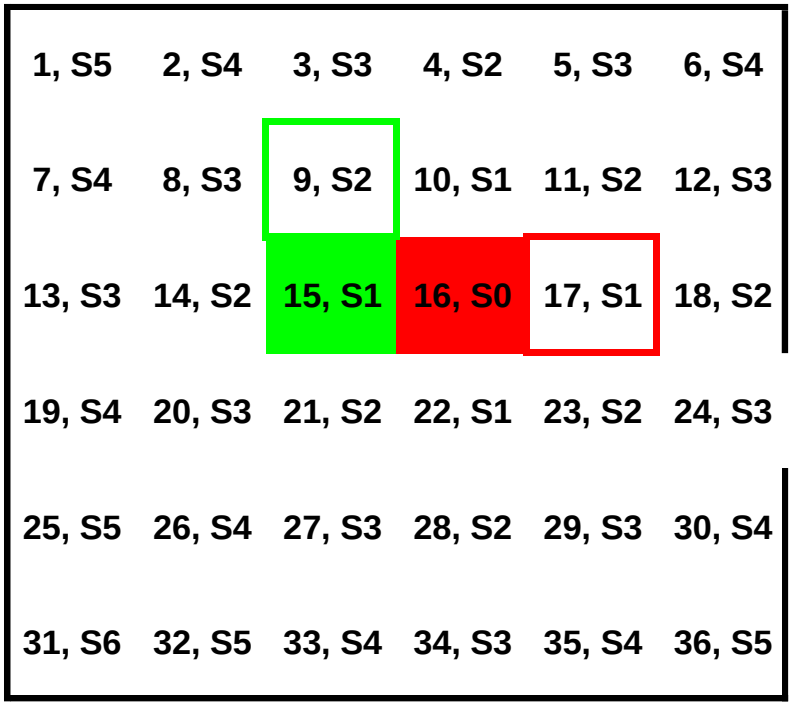

Figure 34 – Positions of the Two Agents After One Time Step

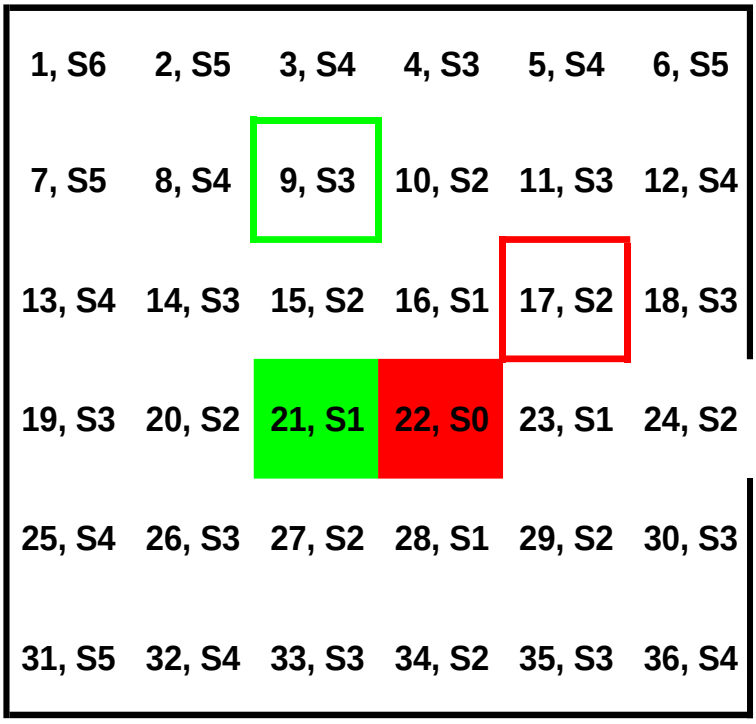

Figure 35 – Positions of the Two Agents After Two Time Steps

Note that action *n*-tuples will also be used in later developments, when different types of action are required simultaneously (such as a change in direction and velocity simultaneously), but for this illustrative example only a single action needs to be chosen (a move up, right, left, or down) because both individual decision processes were deliberately constructed with the same set of actions to choose between.

After two time steps, the system has moved from state (9, S3) to (15, S1) to (21, S1). With regard to the first process (navigating out of the room), the moves from cell 9 to cell 15 and from cell 15 to cell 21 have rewards of -4 and -3 respectively, since cell 15 is 4 steps and cell 21 is 3 steps from the target cell by the exit. The return at this stage is -7, and were the system operating in isolation three moves to the right would result in the system escaping from the room in optimal time (5 steps, with a return of -10). Note that in this example, for simplicity, it is assumed that the return is a simple undiscounted sum of rewards (i.e. $\gamma = 1$) even though this is not analytically required for problems with unconstrained run times.

However, rewards and returns can also be calculated with regard to the second decision process (dodge the monster). The first move from S3 to S1 and the second move from S1 to itself have rewards -5 and -5 (using the reward scheme proposed in the previous section), and so a return of -10 (with the same caveat about discounting as above). The moves taken thus far have produced a poorer return with respect to the second decision process, and continuing to follow

the optimal policy for the first process with a move to the right could be disastrous, as is clear from the last figure.

Clearly there is a question to answer as to which reward scheme should be used to generate a policy and thus guide the autonomous system's decision making. Or put another way, there is a question as to what one should do when the supposed optimal policy is no longer optimal due to a conflict between mission goals.

6.3 The Mathematics of Parallel Decision Processes

6.3.1 Combined Reward Functions

One answer to the question posed above could be to follow the policy corresponding to the mission goal considered to be the more important. For example, the policy could be to head directly towards the exit regardless of the position of the monster, and if you are caught then so be it. However, this effectively reduces the problem to a single decision process with a stochastic element to the transition rewards corresponding to the probability of getting caught on route. To be a truly interactive decision process, choosing one process as more important and side-lining the other cannot be allowed – both processes should be equally important. As such, neither reward scheme should be preferred over the other, and the policy that governs the interacting decision processes must be generated from a reward scheme that respects the goals of both processes. What is required is a combined reward function for an action and the corresponding transition between state pairs.

Any candidate for a combined reward function must reflect how that choice of action and the resulting transition have contributed to the system's progress towards each of the goals it has been set. Some kind of average reward would seem to be the obvious solution. However, in the same way that the behaviour of a system with a single decision to make is sensitive to the numerical values given to the rewards, so the behaviour of a system with parallel decisions to make will be sensitive to the choice of reward function used, whether this be a simple arithmetic average or otherwise. There is no definitive answer to what the appropriate reward function should be beyond the general statement that it must reflect the reward schemes and/or mission goals of each decision process.

Note that in the general case, where a system may have to choose several distinct actions (and where one might be using action *n*-tuples), a reward would have to be defined for each set or

combination of actions that could be chosen when in each state. In the most general context of all, one will require a function that determines the reward following the selection of each possible action *n*-tuple and for each corresponding possible transition between state *m*-tuples. In practice, the state and action representations should be chosen to be as simple as possible, and in this example the basic concept is illustrated with by a single action selection and a transition between state pairs.

The problem that could arise with taking an average reward is that averages are potentially less sensitive to extreme reward values, particularly in the case of extremely costly actions. Diluting the cost of extremely bad actions in one decision process when it coincides with what would otherwise be a good choice of action in the other decision process could be a serious shortcoming. In the example developed here, a move simultaneously towards the monster and the exit may not register as significantly different to another move if the cost of getting closer to the monster is balanced out by the reward for getting closer to the exit.

One crude solution is to set the extremely costly actions with such a high cost they still outweigh the other possible actions even when averaged with the best rewards in the other processes. For example, the reward for encountering the monster in the evasion process is -10, costlier than any of the rewards possible in process one. This reward could be much harsher, say -100 or -1000, but this risks guiding the system not just to never go there but to never go anywhere near there – solution techniques based on iteratively improving value estimates and policies are sensitive to extreme values. Earning an extreme reward value early in the algorithm run-time could discourage the system from actions it might otherwise have considered acceptable, at least during the training/learning stage before an optimal policy has been identified but possibly permanently if the rewards set were so extreme it altered what was identified as an optimal policy. The question of how to set the combined reward function is related to the issue of how much risk a system can be permitted to take, and thus to the broader issue of safety assurance for an MDP-based control algorithm. It should also be clear that setting rewards of $\pm\infty$ for particularly desirable or undesirable actions cannot be allowed.

In general, there will often be a trade-off between setting rewards that easily distinguish particularly good or bad actions and potentially biasing the system so heavily for or against these actions that it changes, inhibits or even cripples the system's operation. The sensitivity of one set of rewards to the other is a critical factor affecting the performance of the parallel decision process, particularly the scaling of the rewards of the two (or more) processes relative to each other. Ironically, it is conceivable that it might sometimes be safer overall to permit

some slightly riskier behaviour in the short term, because not doing so might risk the system ending up immobilised and unable to do anything at all – the appropriate safety criteria, as usual, depend entirely on the type of autonomous system in question and its operational requirements and environmental context.

An alternative solution is to consider using more sophisticated averages than the simple arithmetic average; selecting a reward function that behaves in a manner suitable for the operational requirements of the system. Given that the reward schemes for the two decision processes in the example developed in this chapter are similar in terms of the relative magnitudes of the individual rewards (and are thus more vulnerable to underestimating the risk when extreme rewards are diluted), and given that all the rewards used are negative values, the function:

$$r = -exp\big(-(r_a + r_b)\big)$$

where: $r =$ combined reward

$r_a =$ reward from process a

$r_b =$ reward from process b

is proposed as a function that will produce appropriate reward values for this problem – the rewards behave like a simple average reward until at least one of the two individual rewards gets disproportionately negative compared to the other, at which point the combined reward starts to decrease exponentially and magnify the cost (an exaggerated warning). This function should be less sensitive to the problem of diluting the effect of particularly costly actions and thus provide a more realistic reward function to guide the autonomous system towards behaviour optimal to both of its mission goals.

It should be reiterated that this choice of reward function is not a prescription for all problems. It is specifically chosen for the properties of this particular example, and other options exist – infinitely many in fact. The question of which reward function to use will be as much a part of the design of the controller as the choice of state space representation or the selection of the numerical reward values themselves, and will depend on the design and purpose of the system in question.

6.3.2 Generalising to Higher Dimensions

Two generalisations of key MDP concepts proposed in Chapter 5.6.7 have now been featured in this chapter's worked example (states are now state *m*-tuples and rewards are now reward functions), and in order to solve the parallel decision process using standard techniques (with only minor modifications) the next step is to consider how to integrate these generalised concepts into the MDP framework.

Recall that the basis for most MDP solution techniques is the Bellman value equation, which was defined in Chapter 5.6.2 as:

$$V^{\pi}(s) = E_{\pi}[R_t|s_t = s] = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'}\left[R^a_{s,s'} + \gamma V^{\pi}(s')\right]$$

with all variables as previously defined. The concept of a state needs to be replaced with a state pair (or in general, a state *m*-tuple).

Firstly, note that since there is one of the above Bellman value equations for each state, a common representation of the set of state values is to arrange them as a state value vector:

$$\underline{V}^{\pi}(s) = \begin{pmatrix} V^{\pi}(s_1) \\ \vdots \\ V^{\pi}(s_m) \end{pmatrix}$$

Each element of the vector is the value of state $s_i, i \in \{1,2,\dots m\}$ defined by the appropriate Bellman value equation. Recall that, for the usual MDP solution methods to be applicable, the number of states has to be finite, and so this vector is finite. The ordering of the states within the vector is arbitrary, although logical orderings often present themselves in many state space representations.

To illustrate the adaptation of this framework for state pairs, suppose there are two state variables s and t, then system state is described by the state pair $\underline{s} = (s,t)$. It is not necessary to establish a relationship between the value of the state pair $V^{\pi}(\underline{s}) = V^{\pi}(s,t)$ and the values of the individual states $V^{\pi}(s)$ and $V^{\pi}(t)$ independently. There are two reasons for this.

Firstly, although in the example in this section the two pieces of state information are drawn from the two individual decision problems (cell number is derived from the navigation task and relative separation is derived from the evasion task), in general such a clear division may not always apply. For example, suppose the two pieces of state information were location and time and the goal was to intercept someone. The value of either the location or the current time is not useful in isolation; in this context they are only meaningful together.

Secondly, the value function is an expected return, and so the laws of probability apply. It is likely that individual state values $V^{\pi}(s)$ and $V^{\pi}(t)$, even in a context where they do make sense individually, will generally not be statistically independent of each other.

Thus each state pair is to be treated as a separate element of a higher dimensional state space, avoiding the temptation to assume the value of a state pair is the sum or product of the values of its individual arguments (if this even has any recognisable meaning in the physical world). As such, the natural representation for state pairs is to use a matrix of state values, with the two components of the state information (the two arguments of the state) ordered along the rows and columns of the matrix:

$$\underline{V}^{\pi}(\underline{s}) = \underline{V}^{\pi}(s_i, t_j) = \begin{bmatrix} V^{\pi}(s_1, t_1) & \cdots & V^{\pi}(s_1, t_m) \\ \vdots & \ddots & \vdots \\ V^{\pi}(s_n, t_1) & \cdots & V^{\pi}(s_n, t_m) \end{bmatrix}$$

Here, the state pairs are $\underline{s} = (s_i, t_j)$ for $i \in \{1,2,\ldots..n\}, j \in \{1,2,\ldots m\}$, the above comments about ordering and finiteness still apply, and it should additionally be noted that there is no reason this needs to be a square matrix (i.e. $n \neq m$ in general). Then in the same way that with conventional state vectors the Bellman value equation defines each element of the vector, so here an adapted version of the Bellman value equation will define each element of the above matrix. When working with more general state *m*-tuples, this will be an *m*-dimensional value array, but applied to this example results in the state value matrix:

$$\underline{V}^{\pi}(\underline{s}) = \begin{bmatrix} V^{\pi}(1, S0) & \cdots & V^{\pi}(1, S10) \\ \vdots & \ddots & \vdots \\ V^{\pi}(36, S0) & \cdots & V^{\pi}(36, S10) \end{bmatrix}$$

The state information relating to the position within the room is ordered down the rows and the state information about the relative position with respect to the monster is ordered across the columns in the logical way (although they could have been ordered arbitrarily). Note that the matrix is finite because there are only 36 cells within the room and there can never be more than 10 steps between the two agents (when not permitting diagonal steps), hence 11 possible relative position states including S0. It might be argued that 396 states are unnecessarily many for such a simple problem, but this approach is merely to illustrate how to adapt the MDP solution algorithms to handle parallel decision processes. Remember also that the entire 24th row and 1st column is zero, since being in cell 24 or being zero steps away from the monster are terminal states and have zero value by definition.

In the same way that the set of value equations for each state in the state vector forms a family of simultaneous linear equations, one could attempt to write this matrix formulation as a family of messy matrix equations (possibly utilising some tensor algebra), but this is not recommended and does not bring any clarity to the problem for present purposes. The important point is that the solution algorithms can work iteratively on elements of this matrix rather than elements of the normal state value vector.

Although there are no theoretical constraints to working in higher dimensional state spaces, there will be computational implications for some algorithmic solution techniques. *m*-dimensional state space *m*-tuples would have to be expressed as elements of an *m*-dimensional array in the solution algorithms. However, reducing state space dimensionality as far as possible is always desirable so as not to cause exponential increase in solution run time, which, as well as being impractical, might also cause it to be unsuitable as a learning model if computation in real time is too slow.

The next step to address is how to incorporate the combined reward function into the MDP. In a parallel decision process such as this worked example, there are two rewards earned with respect to two decision processes:

$$r_a = R^a_{s,s'} \text{ and } r_b = R^a_{t,t'}$$

Writing the combined reward function as:

$$\tilde{R} = \tilde{R}(r_a, r_b) = \tilde{R}\left(R^a_{s,s'}, R^a_{t,t'}\right)$$

where the functional form of $\tilde{R}$ is determined by the properties of the system in question. For this example, the functional form proposed in the previous section was:

$$\tilde{R} = \tilde{R}(r_a, r_b) = -\,exp\left(-(r_a + r_b)\right).$$

It is this combined reward function that should be used to calculate the value of a state pair, and thus $\tilde{R}$ rather than $R^a_{s,s'}$ that should be used in the Bellman value equation.

Thus one can now write the amended Bellman value equation for state pairs (or *m*-tuples) as:

$$V^\pi(s,t) = V^\pi(\underline{s}) = \sum_a \pi(\underline{s}, a) \sum_{\underline{s}'} P^a_{\underline{s},\underline{s}'}\left[\tilde{R} + \gamma V^\pi(\underline{s}')\right]$$

where $\underline{s}' = (s', t')$ is one of the possible next state pairs the system could transition into, and thus the sum over $\underline{s}'$ in the above equation represents a sum over all the elements of the value matrix (or the more general value array).

The values of the state transition probabilities $P^{a}_{\underline{s},\underline{s}'}$ and the action selection policy $\pi(\underline{s}, a)$ will depend on the particulars of the system in question. It will be necessary in general to define these for each pair (or tuple) rather than for each individual state variable, because of the problems potentially arising with the statistical independence (or otherwise) of state variables, as noted above (unless one is working with a machine learning algorithm which does not require a priori knowledge of the transition probabilities).

Finally, it is important to note at this stage that an equivalent matrix of action values, adapting the Q-value variant of the Bellman equation (as defined in Chapter 5.6.4), could be formulated in a similar way, but where each element in the *m*-tuple of state values $\underline{V}^{\pi}(\underline{s})$ has to be replaced with its own *mn*-tuple for each state-action value $\underline{Q}^{\pi}(\underline{s}, \underline{a})$ (assuming *n* is the number of possible actions), adding another dimension to the value matrices. This means the equivalent Q-value presentation would be extremely messy to write out on paper as an illustration (hence using the traditional state value formulation in this worked example). However, when implementing such a model in code, all this means is adding another dimension to a digital array (although this may have an impact on solution run time for large problems).

6.4 Implications for the Control of Multi-Agent Systems

Building up an example of a parallel decision process throughout this chapter motivates the detailed development and simulation of this kind of model later in this thesis. However, from the general discussion in this chapter, some observations can be made about what appropriate and inappropriate behaviours might mean in this context, from the point of view of a systems engineer potentially adopting such a process.

The worked example was of a navigational task that could be structured as two distinct decision processes with separate but potentially competing goals. Whilst simplified in order to proceed step-by-step through a worked example, the general principles are clearly applicable to a larger number of autonomous vehicles trying to avoid one another (each agent being a part of the others' environment which needs to be avoided, or in other words each agent could act as one of multiple 'monsters' from the perspective of its peers). The potential generalisation of this worked example to variations of the vehicular navigation examples of complex multi-agent system applications from Chapter 2.5.1 is clear. Considering the multiple moving systems to be manoeuvring robot appendages which also have to seek to avoid collisions with one another

also makes this applicable to the production line robotics exemplar from Chapter 2.5.3 (and which is the basis for the experimentation and simulation activity presented in Chapter 8).

The worked example is characteristic of the kind of higher level strategic or planning layers of decision making for directing physically moving objects, but the structure of the problem and the solution approach can be duplicated at other layers of the decision process, the example given previously being of two separate and distinct system components competing for the same limited system resources. An example of this might be two agents controlling two separate mechanisms in a robot, both of which decide that they should activate now when there is only enough power for one. This could also apply to systems whose constituent entities are not physical objects with easily recognisable patterns of behaviour; for example, this could apply to digital systems, where some or all of the actions and consequential state transitions are virtual.

One might imagine that whatever system proving methodology is applied to determine how safe one's system is might need to be different (or at least to be tailored) for the different levels of decision making. The discussion that follows focuses on the strategic or planning layer, since that is the realm of the worked example and the most easily recognisable layer of most application exemplars throughout this thesis. However, thinking about state spaces abstractly, it is proposed that system proving can be addressed in similar ways across decision layers following general principles.

Desirable or appropriate behaviour may be loosely defined as behaviour which remains within any parametrically defined limits or operating envelope that the system is designed, permitted or desired to remain within, which should be individually determined for the system being assessed. Within the worked example, safety is naturally interpreted as remaining some suitable distance away from the monster.

However, one may also want to consider a second condition of not being too far away from the exit. Continuing with the computer game analogy, given a ticking clock or a restricted number of moves, then being too far away from the exit could prevent the player from completing the game – a failure even if one successfully avoids the monster.

This is analogous in the vehicular navigation domain to maintaining safe separation from other vehicles while also not diverting too far off course and not having enough fuel or battery power to get to its target location (its pick up or drop off point). Alternatively with production line automata, this is analogous to maintaining safe separation from the other automata one is

manoeuvring in the vicinity of, while not moving too far out of position that the assigned tasks cannot be satisfactorily completed.

These second considerations are more of an operational efficiency issue than a safety issue in the traditional sense – if an autonomous vehicle runs out of battery and cannot complete its task then in many scenarios this is annoying and inconvenient but not actually unsafe in the sense of harming anyone (although one can imagine scenarios where this may not be the case due to knock on effects following a failure to complete a task). However, it is still a valid consideration from the point of view of whether the system is behaving in a suitable way to ensure it is still able to achieve its purpose. This leads again to the issue of bounding, constraining or guiding the decision process so that the system in question remains within a specified region of the state space, and so mathematically it is part of the same problem of ensuring desirable or appropriate behaviour.

Continuing with the computer game analogy, two key operational requirements can be determined for the system:

1. To maintain a sufficient separation from the monster.
2. To remain sufficiently close to the target.

The next question is then what 'sufficiently' means, and again this is a judgement that has to be made based on the system in question. Recall that when using MDP-derived control policies, implicit in the reward scheme are the criteria for which the system has calculated an optimal policy to cope with. If the operational criteria are different then this policy is not guaranteed to be optimal. Care must be taken when selecting the reward scheme so that it accurately reflects the behaviour desired from the autonomous system. In short, determination of an acceptable operating region for a system to remain with must be consistent and coherent with what the system is being incentivised to do.

Referring back to Figure 32, the rewards are defined to be negative when there are two or fewer steps between the system and the monster, thus implying the edge of the safe operating region is the set of cells that are three or more steps away from the monster. [Note that normally this thought process is the other way around – the safe operating criteria would be established as part of the system requirements capture, and the reward scheme would follow later as part of the solution design that should be specified to meet these requirements. The logic is reversed here only because of the order in which the problem has been presented.]

Admittedly arbitrarily, suppose one also imposes the criteria that for operational reasons it would be unsafe to be more than five cells away from the exit (for example, being that far away might be deemed too risky and likely to compromise one's ability to complete the game). Then the safe operating region is the union of cells that are simultaneously three or more steps from the monster and five or less steps from the exit. This is illustrated in Figure 36 below, where the states that satisfy these criteria in the current situation are shaded yellow.

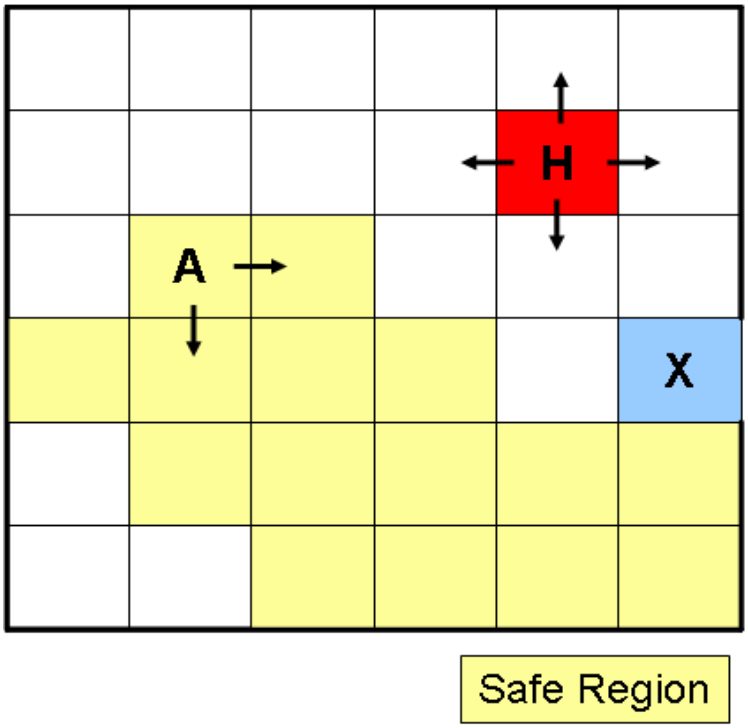

Figure 36 – Initial Safe Operating Region

Although in principle the autonomous system in cell 14 has four possible actions to choose from, the safety criteria restrict this to just two safe options, because the other two would take the system outside of the safe operating region. In practice, the action space of notionally four options has undergone *action filtering*, and the system at this decision point has a reduced action space. In this situation, the optimal policy would have been to choose one of these two actions anyway, so this is an example of optimality and safety agreeing.

However, even this simple example can quite rapidly become more complex. Suppose the system follows its optimal policy and heads towards the exit whilst remaining within the safe operating region. If the monster makes two moves downwards, the next two situations are illustrated in Figure 37 and Figure 38, with the safe operating region (those cells both three or more steps from the monster and five or less steps from the exit) again shaded yellow.

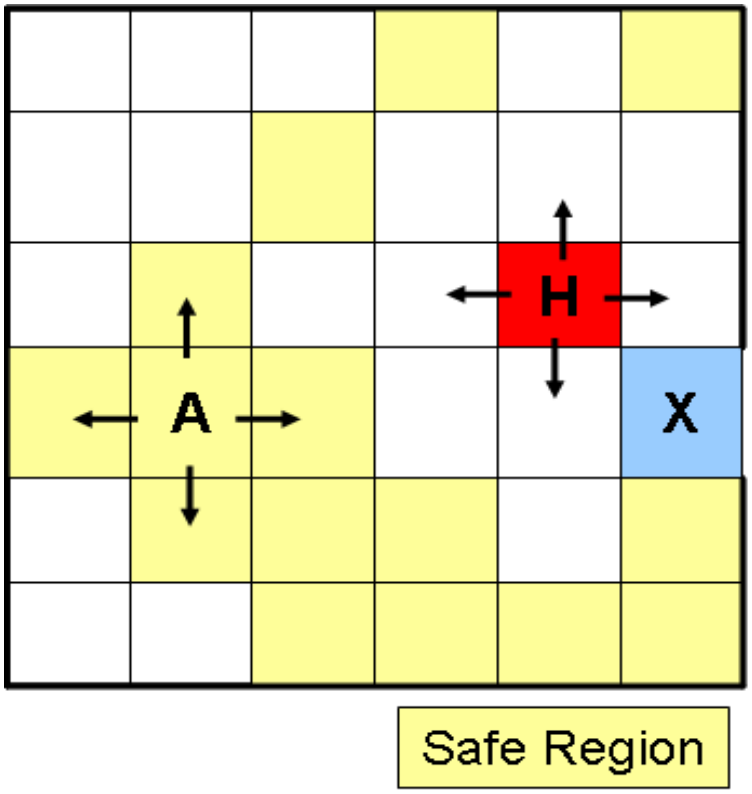

Figure 37 – Safe Operating Region After One Time Step

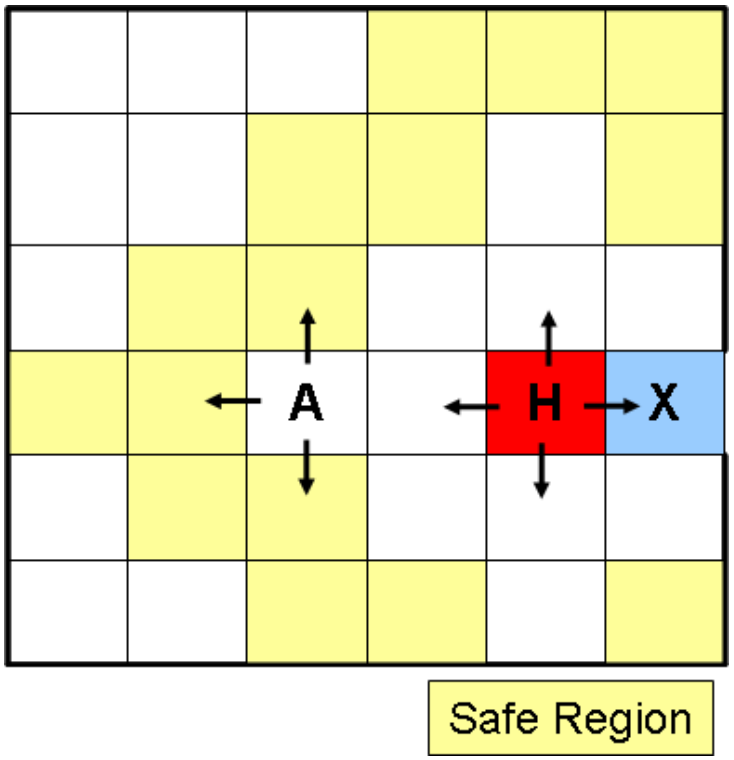

Figure 38 – Safe Operating Region After Two Time Steps

After one time step, notice that the system has selected an optimal move that has kept it within the safe operating region of the state space. But also notice that the shape or structure of this safe operating region has changed. Some parts of it have become disconnected from the rest of the safe region, and thus one might choose not to include these as part of the safe operating region at all since the system cannot reach them safely anyway as things stand (there is no requirement for a safe region to be contiguous, although it might have practical implications if not as there may be a requirement, implied or explicit, to traverse only through the safe operating region). A move in any direction is classed as a safe move at this point, given the position of the monster, but clearly the optimal moves will be those which keep the system moving closer to the exit. Suppose a move to right is chosen, as this is a move towards the exit and so at that point in time it is one of the optimal actions. What can be seen from Figure 38 is that what might actually happen is that despite following the optimal policy, the system has nonetheless unexpectedly drifted out of the safe operating region. This is not because the optimal policy was wrong nor because the action did not have the result intended, but because the local environment has changed, and thus the safe operating region has changed. The safe path to the exit the system might have had has moved and is now along the opposite edge of the room. In light of the move the monster made, a move downwards which at the time was suboptimal would in retrospect have turned out to be better for the system than the supposedly safe and optimal move right, demonstrating the trade-off between optimality and appropriateness inherent in these problems and the difficulties with operating in dynamic environments.

This example is admittedly rather artificial. One could argue that the system should have known the move down would have been more likely to be safe given the current position of the monster. But then a counterargument could be made that in practice the system may not be in a fully observable environment – say if its sensors can only look in the immediate direction of travel, then at the previous time step it could not have known where the monster was and thus could not reasonably have predicted its subsequent movements, or that a move which appeared to keep the system within the safe operating region actually would not do so. The problem is fundamental – even if performance criteria can be defined quite easily, if the operating environment is dynamic then which subsets of the state space correspond to appropriate, desirable or safe operating regions is also dynamic. A transient safe operating region with a changing shape is a significant complication to the analysis. Methods for dealing

with this would be a significant further branch of research beyond the scope of this thesis (although some introductory discussion on the subject is contained within Annex C).

Figure 36 to Figure 38 may illustrate an artificial example, but they illustrate it well – it is not a simple problem predicting how the safe operating region will evolve in a dynamic environment, and even in a relatively static environment it may not be easy to identify which actions represent the best trade-off between optimal performance and appropriate or safe operation. What is required is for a learning system in such a scenario to have the ability to filter its own actions to guide it towards action selections that should keep it within an appropriate operating region (however that is to be defined for any given problem), whilst also having the ability for onboard adaptation in the event of unplanned environmental change inhibiting this.

6.5 Application to Interactive and Synchronised Autonomy

In this thesis, the description interactive autonomy is taken to mean any interaction between two or more autonomous systems where the behaviour of one impacts another (with a formal definition proposed in Chapter 2.2). These interactions could be collaborative, cooperative or competitive – the key point is that there is some kind of dependency where one's actions affect another. An interaction could take the form of assisting each other towards a common mission goal, not interfering with each other's progress towards separate but simultaneous mission goals, or deliberately acting to disrupt the operation of another system. Synchronised autonomy is a subset of these situations, where the autonomous systems in question are all acting in a unitary fashion unaware of each other and not programmed (or perhaps not even capable of being programmed) to coordinate actions with their peers, but are nonetheless required to perform their functions in a synchronised pattern in order to achieve the overall system objective (or objectives).

The aim of the remainder of this chapter is to provide an introduction to the application of the parallel decision making framework to the interactive autonomy problem, building on the simple example of the dynamic obstacle avoidance problem developed in detail previously. Chapter 6.5.1 will discuss the analogous relationship between collective operations and hierarchical systems that motivates the common adoption of the parallel MDP approach.

Chapter 6.5.2 will develop an example of a pair of interactive automata in a simplified scenario, to illustrate that the collective behaviour of the composed system is a function of the individual behaviours of each (although not in general a simple additive combination). The nature of the interaction will in some respects extend and in other respects constrain the effectiveness and scope of each individual action. Practical implementations will, as usual, be sensitive to the particular characteristics of the problem in question; the example presented is an extension of the worked example developed throughout this chapter for illustrative purposes. Chapter 6.5.3 contains a brief aside on the potential impact should a system designer choose to use the partially observable formulation of the decision-control problem.

6.5.1 Collective or Hierarchy: Analogous Perspectives

Returning to the dynamic obstacle avoidance example developed from Chapter 6.2 onwards, suppose that the other entity is another autonomous system with some independent decision making capability, with mission goals of its own to fulfil. With both systems engaged in separate operations but influencing each other, this is an example of interactive autonomy.

The parallel decision process is a composition of two (or more) separate decision processes taking place simultaneously, the choices of action taken by each process influencing the other. Any situation where two or more autonomous systems are compelled to interact because their mission goals, resources or locations overlap is an example of dependent parallel decision processes. In the same way that separate, but dependent, mission goals in a single system can be incorporated into a single parallel decision process, so also can separate but dependent goals in a collective of systems be analysed as a composition of parallel decision processes. Each element of the collective of systems can be seen as a subsystem of an overarching system, and the mathematical framework developed in Chapter 5.6 can be applied to collective operations, with rewards now describing the overall gain or cost to the collective from the actions of the individuals.

It was noted above that parallel decision processes are not strictly a game theory problem, because the autonomous system is acting in response to its state and environment rather than to other players. Clearly this assertion is no longer applicable as two systems interacting is precisely what is now being proposed for this framework to apply to – each system is a part of the environment of the other, with a relative state to each other. However, notwithstanding this, this is still not a game theory problem, due to the other reasons described earlier in this thesis, primarily in Chapter 4.3.3.

6.5.2 An Example: The Guard-Intruder Scenario

The worked example scenario can be extended by introducing an active ‘guard’ or ‘sentry’ whose goal is to prevent an ‘intruder’ from approaching a particular location. The resultant gestalt, or whole-system, behaviour can be illustrated as a function of the intruder policy.

In general, systems in such situations will make decisions which will often be based on what it is anticipated that the other systems will do (much like people would do in human interactions). Challenges may arise over both the quantity of state information that will exist in a collective of autonomous systems, and the difficulty of obtaining such information. It is not difficult to imagine scenarios where communication between systems is limited or impossible, for technical and/or operational reasons. Competitive interactions, which require autonomous systems to interact with uncooperative and potentially hostile counterparts, are particularly unlikely to feature much (if any) communication between the competing systems. In this latter case in particular, a system must make operational decisions without any ‘voluntary’ input from the other entity. Since the effectiveness of a selected action will often depend on the simultaneous and independent actions of the other systems, the decision making process is fundamentally linked to the process of predicting the intent of observed entities.

Applied to the worked example, the guard and intruder are a generalisation of the earlier ‘player’ and ‘monster’, in that both are now considered to be agents taking purposeful decisions, with ‘intent’. However, from the perspective of either one of these systems, the other is effectively playing the same role as the earlier monster, by obstructing the system as it attempts to reach a particular location or vice versa. In the developments up to this point the policy of the guard has been kept fixed so that differences in collective behaviour are solely a function of the change in the policy of the intruder (a random monster has a 25% chance of taking any of the four optional directions, subject to the obvious restrictions at the edges). The intruder will have no knowledge of the control policy of the guard, and so the guard is in a practical sense part of the intruder’s environment just as the monster was before. Lacking any knowledge about the guard, it can assume no better than that the guard could move in any direction at any time. It may become apparent or deducible following sufficient observation if the guard has a more sophisticated control policy than a uniform distribution, but this will not in general be the case from the outset of the interaction, and may not be possible at all if the first system is neither equipped for nor capable of recognising or interpreting any observed patterns of behaviour from the other system.

The focus of this discussion is on the application of the parallel decision process created for competing subsystems to apply to collective (and ultimately to synchronised) autonomous operations. In summary, a simple, and in scenario terms ineffective, guard policy could be implemented in which the guard merely moves towards the last perceived position of the intruder. This means that it is forever pursuing, but never catching, the intruder. This is similar to the monster in the earlier example simply chasing the primary system or, looking ahead, to a system being trained to deal with the case of trying to avoid a collision with another who is making life as difficult as it can by choosing the most awkward counteractions possible.

How this scenario might play out can be seen in Figure 39 below, where the policy of the intruder (blue) is to avoid the guard (red) and approach the 'protected location' at the origin.

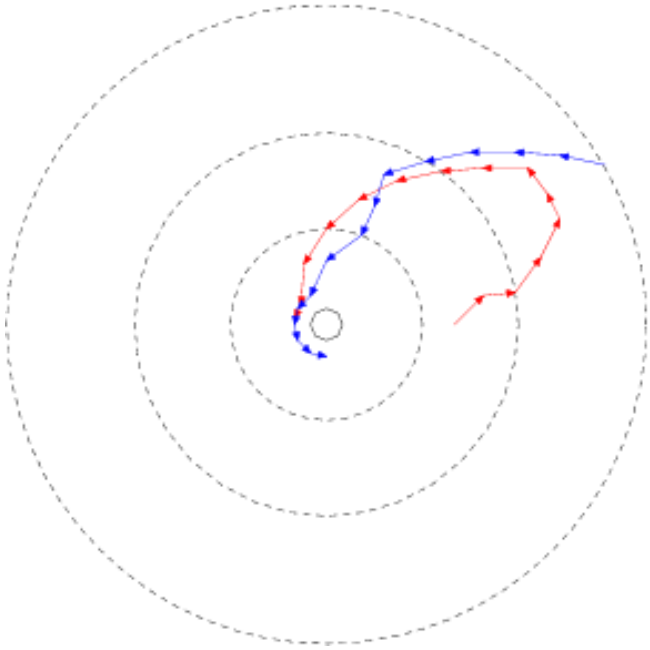

Figure 39 – Hostile Intruder Policy

There is a distinctive collective behaviour visible in this interaction, made obvious by the contrast with the behaviour illustrated in Figure 40. In this figure the intruder behaviour is simply to avoid the guard. The goal of approaching the origin has been removed from the calculation of its policy.

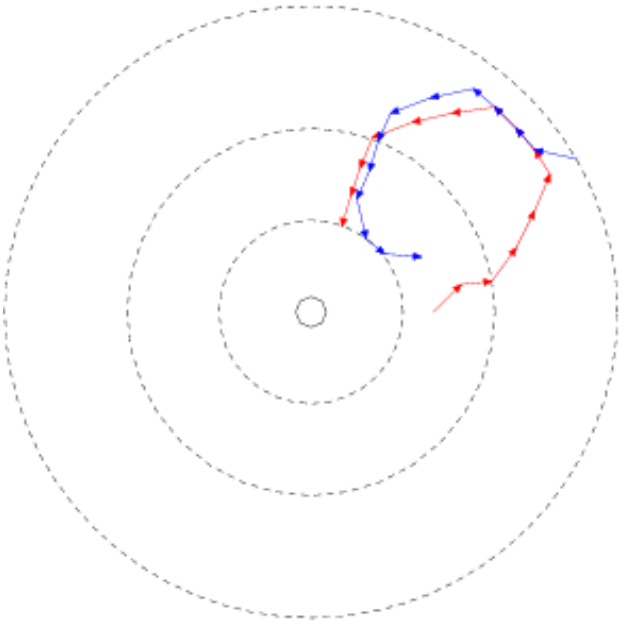

Figure 40 – Benign Intruder Policy

This is, of course, a trivial example, produced merely to illustrate that a significant observable difference should be apparent in the gestalt behaviour. However, the purpose of these simple visualisations is to highlight one of the key difficulties faced by an autonomous system whose operations require interaction with another uncontrolled entity, which is that the system will typically have to consider the states of the other systems it is interacting with as it selects its action, else very different patterns of joint behaviour can result. This is analogous to the dynamic obstacle avoidance problem described in Chapter 6.2, where the estimation of the likely state transitions (specifically, the estimation of the current control policy) of the other entity affected the expected returns from particular action selections and thus impacted upon the policy. One might imagine an endless cycle of nested beliefs about the other entities in an interaction, significantly complicating the decision making process. Candidate approaches for tackling this problem, besides the obvious option to simply assume a particular policy on the part of the other entity, include revisiting the potential application of game theoretic approaches to interactive decision making (although this will be highly sensitive to issues such as whether one is dealing with zero sum games, and the concern raised previously as to whether Nash equilibrium solutions are appropriate to this problem domain, for example in this case if systems have different rules of engagement) or to stimulated dialogue approaches (see [16] or [26] for alternative approaches to tackling this, only the latter of which involved the author, but for which both are subject matter beyond the scope of this PhD).

6.5.3 Revisiting Partial Observability

Note that one significant difference between this example and that developed previously is that there are prescribed system dynamics (i.e. it is stated in the problem definition what the probabilities of making particular state transitions are). Despite there being uncertainty over exactly where the system would end up relative to the other system, since the movement of the other system was obviously unknown to the system at the decision point, there were only a few possible options. Even though the exact end state could not be known, some aspects of it (absolute position) would be known and other aspects (relative position) was known to be a finite set. The MDP framework (and dynamic programming solution methods) could handle this, provided one makes reasonable assumptions as to what the probabilities of each possible transition might be. The reason this is not the preferred approach, proposing instead a reinforcement learning solution technique, is to avoid having to prescribe these transition dynamics, even probabilistically, because this still requires an implicit assumption about the policy of the other entity, which could be highly uncertain or ambiguous. Where a dynamic programming method will require assumptions about the other entity's policy in order to construct even probabilistic transition models, reinforcement learning methods need make no such assumptions and, providing there is sufficient state space exploration, will implicitly learn the policy of the other entity and factor it into the assessment (although that is in itself an assumption that the control policy of the other entity is fixed).

This problem with providing the decision process sufficient information about the state of its counterparts suggests that bringing partial observability back into the problem may provide an alternative approach to that illustrated above. Referring back to Chapter 5.6.8, the key difference between the MDP and the POMDP is that states are replaced with belief states – probability distribution over the state space – and observations are included as an additional element in the model. Action selections are then based on the last action and the current observation, which in turn transforms the belief state according to how the observation has changed. Unfortunately, this is computationally difficult (often intractable), and in any case, prescribing how one believes the state has changed is not far removed from prescribing exactly how the state changes, which one might not want to do in a dynamic operating environment with the inherent uncertainty that entails. Put simply, it might be considerably more work and a harder computational solution for little or no improvement in model fidelity or accuracy.

While the purpose of this short discussion is merely to consider the merits of the potential reintroduction of the POMDP formulation for this problem, it is important to consider the sort

of example for which one might wish to do this. Referring again to the dynamic obstacle avoidance example, supposing a probability distribution of possible next positions of the mystery entity could be defined (even if it were simply a two-dimensional Normal distribution centred on the currently observed position – similar to the 'dumb monster' from before, equally likely to go in any direction) then a POMDP solution technique would allow, in theory, for all the possible future positions to be included in the decision making process (weighted by the likelihood of their occurrence). Of course, to actually implement this and solve it with the techniques currently available, the distribution might have to be quantised into a set of probability measures at sample points around the currently observed position, which is then not far removed from the earlier MDP approach with specified transition probabilities (or learned transition probabilities if working with a machine learning technique).

Despite the challenge around determining the control policies of other interacting entities, and the way this appears to prompt a re-visit of the POMDP approach, in practice the same concerns and difficulties around the use of the POMDP previously discussed in Chapter 5.6.8 arise again. Questions remain about whether the POMDP is a viable way forward for interactive autonomy applications. However, it would be remiss not to note that this may admit a useful formulation to this problem given the inherent uncertainty, and hence the potential for a subjective belief of the likely next state of interacting peers and of the overall system's state dynamics in a complex multi-agent system. Exploring this issue is beyond the scope of this thesis but would be an interesting future direction for further studies (see further recommendations in Chapter 9.2.4).

Chapter 7

7 Extending the MDP to Hierarchical Decision Processes

7.1 Context

This chapter will further extend the mathematical framework introduced in Chapter 5 to also encompass multiple layers in the internal architecture of complex systems. Combined with the developments in Chapter 6 into the mathematics of parallel decision processes, this completes the mathematical formalism needed to effectively model the decision making processes within a complex multi-agent system that can be decomposed into a hierarchy of decision nodes, as described in Chapter 5.5.

This chapter does not include a detailed worked example of layered or fully hierarchical decision processes in the way Chapter 6 did for parallel decision processes. This is because the exemplar multi-agent system application developed and implemented as part of the PhD is a parallel process and so this thesis necessarily has to concentrate on those aspects. It is also to clearly differentiate the work undertaken as part of this PhD from the author's precursor work, which took entirely the opposite approach of developing a simple example of a layered system – and a much simpler exemplar with no interacting peer systems or emergent behaviours to contend with.

However, it is still important that this thesis tells a complete story, and the mathematical formulation proposed for the wider range of complex system organisations and structures proposed in Chapter 5.5 requires further extension of the mathematical models to accommodate multiple layers of decision nodes within complex systems. This chapter develops the mathematical content required to extend the models developed in this way, illustrating how to incorporate interactions up and down between layers of a system. Although the multi-agent system exemplar developed after this is naturally modelled as a set of parallel decision

processes, the material developed in this chapter provides the tools that could be used for building models for other applications where layering is explicit (including the alternative applications described in Chapter 5.5 and the detailed example illustrated via Figure 17 and Figure 18, or alternatively those example application areas documented in Annex B).

7.2 The Mathematics of Hierarchical Decision Processes

It is first necessary to incorporate multiple action selections (via action *n*-tuples, similar to the state *m*-tuples introduced previously), and the influence of decisions made at different levels of the hierarchy (and the communication of these influences through the reward functions).

As discussed in Chapter 5.6.5, typical solution techniques for MDPs involve a value or policy iteration process conducted using the recursive Bellman value equation:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'} \left[R^a_{s,s'} + \gamma V^{\pi}(s') \right]$$

where all variables are as defined in previous sections. In Chapter 6.3 this was refined for the case of parallel decision processes. Specifically, states were generalised to state *m*-tuples (and thus the set of state values typically written as a state value vector became an *m*-dimensional array of state values), and rewards from multiple decision processes were combined to provide a total system reward by means of a combined reward function $\tilde{R} = \tilde{R}(r_1, r_2, \dots)$ that need not be merely the arithmetic average reward if an alternative functional form is more appropriate. The refined value equation that results (for the two state variable problem) is thus:

$$V^{\pi}(s,t) = V^{\pi}(\underline{s}) = \sum_a \pi(\underline{s},a) \sum_{\underline{s}'} P^a_{\underline{s},\underline{s}'} \left[\tilde{R} + \gamma V^{\pi}(\underline{s}') \right]$$

which will extend to higher dimension state spaces in the logical way.

For the full hierarchical decision problem, it is necessary to additionally incorporate processes involving the selection of multiple different types of action simultaneously in support of a single mission goal. This immediately means that not only do states have to be generalised to state *m*-tuples, but so do actions have to be generalised to action *n*-tuples.

Fortunately, the changes that need to be made to the model are not too severe. In general, the action *n*-tuple selected at a given decision point can be written $\underline{a} = (a_1, a_2, \dots a_n)$ for individual actions $a_i, i \in \{1,2, \dots n\}$. Each action *n*-tuple $\underline{a}$ should be treated as an element of a higher dimensional action set rather than as some functional product of individual actions, for the simple reason that individual actions a_i could be totally unrelated (for example, the

actions being a direction to move and an electronic signal to send, which are completely different types of action).

Different actions can be selected by the various layers in the decision hierarchy (different strategic actions and control actions in a simple two-layer example). But then different state information is required to make decisions at different layers of the hierarchy. The medium through which an MDP determines its performance towards its goals is, as usual for MDP models, via a reward scheme, and so in the hierarchical model spreading reward between layers of the hierarchy appears to be a natural approach to modelling the way in which the different layers of the hierarchy influence the progress of the overall system. Therefore, for a generalised hierarchical MDP, there should be a set of functional value arrays, one for each layer of the hierarchy, each of which consists of its own state and action arrays but, crucially, each using a combined reward function containing all the rewards from all levels of the hierarchy, with the rewards acting as the communication channel between layers of the hierarchy, influencing the action selection process.

Specifically for a two-layer scenario, if poor control decisions affect the system's progress towards its strategic goals (or vice versa) then each layer of the hierarchy will only learn this through consideration of the combined reward, otherwise layers of the hierarchy in isolation will operate according to what is best just for them and not resolve any arising conflicts. For example, the control layer might think it is making good choices if it is receiving good rewards, but if these actions are unexpectedly hindering strategic progress then the system will not learn this and adapt to it without also receiving the negative rewards being accumulated at the strategic layer. It is the combined reward from the whole system that informs each individual layer about the overall whole-system performance. All the same comments as in the previous section about the importance of choosing an appropriate functional form for the combined reward function are still relevant, and a good choice of functional form for the reward function should allow for segmentation of the states and actions, making the system simpler to model and solve and, ultimately, to analyse and understand.

To bring this together into a single hierarchical MDP model, it is necessary to introduce one final mathematical concept – the Kronecker product (or a more general tensor product for higher dimensions). This product is defined as the following $np \times mq$ matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & \cdots & a_{11}b_{1q} & \cdots & a_{1m}b_{11} & \cdots & a_{1m}b_{1q} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & \cdots & a_{11}b_{pq} & \cdots & a_{1m}b_{p1} & \cdots & a_{1m}b_{pq} \\ & & & \ddots & & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ & & & & \ddots & & \\ a_{n1}b_{11} & \cdots & a_{n1}b_{1q} & \cdots & a_{nm}b_{11} & \cdots & a_{nm}b_{1q} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{n1}b_{p1} & \cdots & a_{n1}b_{pq} & \cdots & a_{nm}b_{p1} & \cdots & a_{nm}b_{pq} \end{bmatrix}$$

where A and B are (not necessarily square) $n \times m$ and $p \times q$ matrices:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \text{ and } \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix}$$

These are often referred to as *block matrices* because of the way they are constructed with each block of the matrix actually being one of the constituent matrices multiplied by the element of the other corresponding to that region of the matrix. The whole matrix contains all combinations of products of elements of the constituent matrices arranged by block. Note that there are tensor products that can be used to generalise this to higher dimensions, but this two-dimensional matrix version is sufficient to illustrate the principle.

This block matrix (or the more general block array) formulation is useful because one can form simple matrices (or arrays) of states and actions and use a Kronecker product to combine these into one complete array containing all possible combinations of states and actions, in a form suitable for a value or policy iteration algorithm. A block matrix at each level of the hierarchy, with combined rewards function used in the value estimations at different layers, will then provide the mathematical apparatus to describe the whole hierarchical MDP.

The adapted version of the value equation used to calculate the value of each state is just a minor alteration of the earlier version, adapted for action *n*-tuples $\underline{a}$:

$$V^{\pi}(s,t) = V^{\pi}(\underline{s}) = \sum_{\underline{a}} \pi(\underline{s},\underline{a}) \sum_{\underline{s}'} P_{\underline{s},\underline{s}'}^{\underline{a}} \left[\tilde{R} + \gamma V^{\pi}(\underline{s}')\right]$$

Note of course that there is an equivalent formulation using state-action pairs (technically now a set of state-action *mn*-tuples) or Q-values, which is constructed in an equivalent way, but starting from the action value or Q-value formulation of the Bellman value equation. As

previously noted, it is actually the Q-value formulation rather than the traditional state value version that is a more natural formulation for an agent-based system owing to the nature of agents as software entities that chose best-value actions, but the principles are equivalent.

Next consider a state vector pair $\underline{s} = (s_1, s_2, s_3, \ldots s_N)$ at the strategic or planning decision layer and $\underline{\sigma} = (\sigma_1, \sigma_2, \sigma_3, \ldots \sigma_M)$ at the control decision layer, with a decision at the strategic layer to choose from a set of actions $\underline{a} = (a_1, a_2, \ldots a_P,)$ and a decision at the control layer to choose from a set of actions $\underline{\alpha} = (\alpha_1, \alpha_2, \ldots \alpha_Q)$. The combined reward function will be $\tilde{R} = \tilde{R}(r_s, r_c)$ of a functional form to be determined based on individual system context, where r_s, r_c are the rewards from the strategic and control layers respectively but $\tilde{R}$ is the combined reward that should be used in value calculations.

Then the block matrix $S \otimes A$ orders every possible combination of state and action tuples, with each block of the matrix corresponding to one of the state *m*-tuples and containing all of the possible action combinations that could be selected.

Then:

$$\underline{\underline{V^{\pi}}}(\underline{s}) = \underline{\underline{V^{\pi}}}(s_i, s_j) = \sum_{\underline{a}} \pi(\underline{s}, \underline{a}) \sum_{\underline{s'}} P^{\underline{a}}_{\underline{s},\underline{s'}} \left[\tilde{R} + \gamma V^{\pi}(\underline{s'})\right]$$

defines the value of each element of matrix S (or alternatively, the sum over each block of matrix $S \otimes A$, where the sum over $\underline{a}$ represents the sum over all elements in the corresponding blocks), and these values can be placed in a second matrix array:

$$\underline{\underline{V^{\pi}}}(\underline{s}) = \underline{\underline{V^{\pi}}}(s_i, s_j) = \begin{bmatrix} V^{\pi}(s_1, s_1) & \ldots & V^{\pi}(s_1, s_N) \\ \vdots & \ddots & \vdots \\ V^{\pi}(s_N, s_1) & \ldots & V^{\pi}(s_N, s_N) \end{bmatrix}$$

Iterative policy estimation can be carried out on the elements of this matrix to identify the actual values (and hence identify an optimal policy).

Alternatively, using Q-values, a block matrix can be used:

$$\underline{\underline{Q^{\pi}}}(\underline{s},\underline{a}) = \ldots.$$

$$\begin{bmatrix} Q^{\pi}((s_1,s_1),(a_1,a_1)) & \ldots & Q^{\pi}((s_1,s_1),(a_1,a_M)) & \cdots & Q^{\pi}((s_1,s_N),(a_1,a_1)) & \ldots & Q^{\pi}((s_1,s_N),(a_1,a_M)) \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ Q^{\pi}((s_1,s_1),(a_M,a_1)) & \ldots & Q^{\pi}((s_1,s_1),(a_M,a_M)) & \cdots & Q^{\pi}((s_1,s_N),(a_M,a_1)) & \ldots & Q^{\pi}((s_1,s_N),(a_M,a_M)) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ Q^{\pi}((s_N,s_1),(a_1,a_1)) & \ldots & Q^{\pi}((s_N,s_1),(a_1,a_M)) & \cdots & Q^{\pi}((s_N,s_N),(a_1,a_1)) & \ldots & Q^{\pi}((s_N,s_N),(a_1,a_M)) \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ Q^{\pi}((s_N,s_1),(a_M,a_1)) & \ldots & Q^{\pi}((s_N,s_1),(a_M,a_M)) & \cdots & Q^{\pi}((s_N,s_N),(a_M,a_1)) & \ldots & Q^{\pi}((s_N,s_N),(a_M,a_M)) \end{bmatrix}$$

and iterative policy iteration can be carried out on the elements of individual blocks within this matrix to identify an optimal policy solution.

The equivalent structure as above can be used for the control level MDP, using states $\underline{\sigma}$ and actions $\underline{\alpha}$. The key is to select an appropriate functional form for the combined reward function $\tilde{R}$ used in value updates in the selected solution, calculated with respect to the goals and action selections on both layers and then used in value updates across both layers, to ensure the optimal policy generated supports all system goals at all levels of decision making.

One can already see that the number of state-action combinations is already becoming a problem for presenting the array. However, high dimensionality is often unavoidable and the typical value or policy iteration algorithms that will solve MDP problem, even in these higher dimensions, will still be an improvement on direct calculation.

All the developments in this section have assumed two layers. This is just for simplicity of presentation in written form. For three or more layers, an analogous approach can be used, but the resulting arrays will now be higher dimensional arrays and best handled by computer. In this way though, the system designer has the flexibility to build a layered set of hierarchical MDPs for multi-layered systems in the way that they require.

Chapter 8

8 Experimental Problem, Simulation and Analysis

8.1 Motivating Example: Synchronised Robotics on the Industrial Production Line

The technical challenge that motivated this research is that of the synchronous operation of co-located production line robots. This is not the only application domain in which challenges with synchronising robotic operations might be anticipated, but it does provide perhaps the most widespread use of simple robotic automata for which robust learning techniques could have significant future benefit. In particular, this work has been motivated by the work of Doltsinis, Ferreira and Lohse into the potential application of Q-learning to improve the ramp up period of automated production lines [28]. That work considered the ramp up period for workstations on a production line, and the performance that Q-learners could deliver as a substitute for the human decision making process during ramp up; that is, the process of adopting a pattern of behaviour with sufficient confidence that cyclical or episodic repetition of that behaviour will deliver the desired performance against the relevant criteria, specifically in that case functionality, quality and performance criteria.

The promising results of that research motivated investigation of whether the technique could be generalised to concurrent co-located workstations. Doltsinis et al. considered a single workstation and the ramp up time associated with learning the optimal behaviour for a particular automaton – not the ramp up time or subsequent performance associated with a number of them working simultaneously and interacting (or interfering) with each other. Extending this thinking to this more advanced problem presented an ideal case study for this PhD.

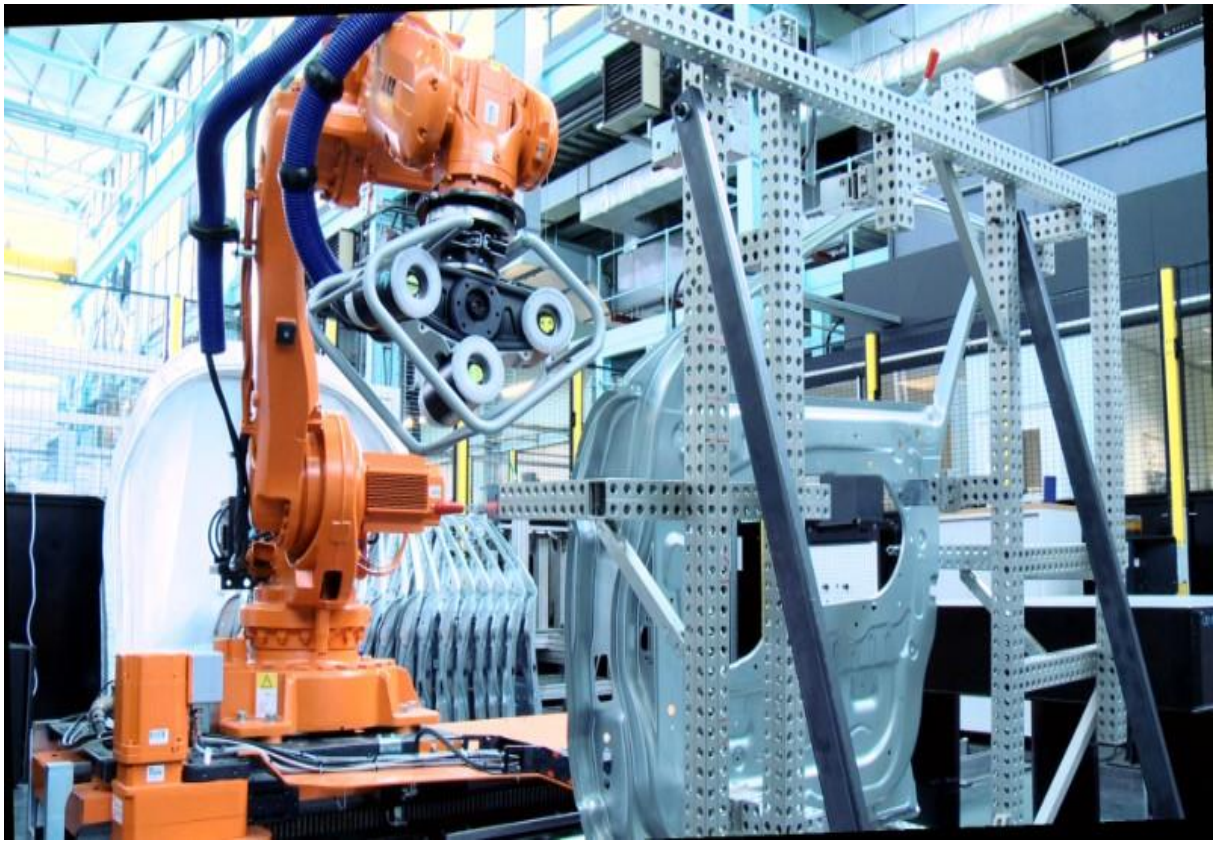

Figure 41 – Production Line Automaton with Robotic Effectors

It can be assumed that a manufacturing director is unlikely to choose to co-locate automata when there is sufficient space or time available to conduct tasks separately and there is no pressing driver for concurrency. Nevertheless, there are foreseeable circumstances in which there might be benefit or necessity in doing do. These include, but are not limited to:

1. Space-constrained factories, where new projects require new or additional equipment but there is a desire not to locate away from the existing premises. Reversing this logic, were speedy and effective synchronous operation of multiple automata sufficiently robust, there is the possibility that some organisations might actively seek to locate to smaller premises for commercial reasons.

2. Fast turnaround manufacturing runs, where relatively small numbers of items are required. The time taken to physically reconfigure the facility, moving machinery around and then reconfiguring the necessary units for the new assignment can be a significant overhead. The ability to leave them in place and reliably work around them would enable much speedier turnaround.

3. Mass production runs, where on some tasks there may be scope for greater concurrency on the production line to increase throughput. One obvious approach to take to support this is to enable multiple automata to operate on a particular production element simultaneously.

From the point of view of the industrial ramp up problem, Doltsinis et al. have already demonstrated the potential efficacy of Q-learners to the problem of machine learning industrial production line techniques. However the work they conducted represented a single workstation and the ramp up time associated with learning and adopting the optimal behaviour for that particular automaton. Not addressed by that work was the question of whether Q-learners would remain an effective approach to speedy solution of the ramp up problem should the automata controlled by said agents have to interact with one another – for example should they simultaneously be working on the same object. A further challenge arises from having the learning agents interacting with each other, as part of each other's environment, where the evolution of that environment is not inherently predictable to that agent (i.e. assuming limited or no communication between agents).

Figure 42 – Production Line Automaton in situ

Clearly in an ideal world the individual automata would be separated, with little possibility to physically collide with one another. However not all industrial plants will be able to support having as many pieces of equipment as they would like available at any time, and some may see operational advantages in being able to increase the number of automata that they are able to keep on the factory floor, to reduce the time and effort associated with rearranging the factory layout for new production runs. Even if there is ample physical space, there is clear potential for operational and commercial advantage through reducing the number of distinct steps in a production line – and one obvious step to take to support this is to enable multiple automata to operate on a particular production element simultaneously. For example, if there are four distinct cuts that need to be made with a laser, the production line will be able to produce more units more quickly if two or even four lasers could be able to operate on the same piece of metal simultaneously (when this is possible).

In theory of course each individual automaton could be programmed separately with a particular sequence of control actions, and each coordinated to ensure that it will not get in the way of what the others are doing. There may still be operational advantage in doing this, but the ramp up time will be much longer. Not only does each automaton have to be configured and programmed separately, but then once each one is configured and tested, there is the risk that when another interacting automaton is added to the production line then not only will that new automaton require ramp up time for configuration and testing, but each existing automaton may require reconfiguration and further testing to ensure the control policy it was going to follow is still appropriate in the presence of the new unit. It cannot be assumed that multiple automata configured individually can be left to operate side by side and acceptable performance will still be achieved – there is a clear risk of emergent behaviours, as well as the more obvious risk of impedance.

In much the way that Doltsinis et al. proposed that a reinforcement learning methodology such as Q-learning might provide a mechanism for automata to more speedily ramp up, so it follows that network of Q learners might provide a mechanism for multiple automata to more speedily ramp up and find a synchronous configuration that will be acceptable overall. It is to be expected that the control policy and thus the observed behaviours from the multiple automata will not simply be the superposition of their individual control policies – such a scenario would be relatively trivial. It is expected that more complex patterns of behaviour from the interaction of the Q-learning agents will be observed.

8.2 Experimental Approach to the Simulation of Synchronised Robotics

To begin investigating this supposition, a simplified example of multiple interacting automata has been developed and demonstrated. The key research question is whether or not acceptable patterns of behaviour can speedily and effectively emerge from co-located systems using networked reinforcement learning agents. Successful demonstration of this proposition will represent an initial proof of concept, which might then unlock the possibility of more extensive deployment of networked learning agents in more complex problems, with larger state spaces, greater numbers of automata and different state space models or geometries.

The systems modelling paradigm and the mathematical framework is based on that developed in the earlier sections of this thesis.

An experimental simulation-based test bed was developed specifically for his PhD. It has been designed to be extensible for future enhancements (specifically the addition of more agents and/or more complex patterns of required behaviour, as these are loadable elements into the test bed), but is initially exercised with a much simpler and more comprehendible problem. The mechanics of this experimental simulation-based test bed, including pseudocode for the Q-learning agents and other major functions, is described more fully in Annex A to this thesis.

8.2.1 Introductory Problem

Suppose that an industrial automaton is required to guide a laser along a predefined arc on a piece of metal. The factory owner would like to have multiple automata conducting several such tasks simultaneously if possible (for reasons of factory space and/or throughput). The arcs that each individual laser has to trace need not be the same shape.

The laser head is fitted to a rotating gimbal and the natural geometry to use is polar coordinate geometry (r, θ). For initial simplicity consider the two dimensional case, where r is the distance between the laser head and the target point, and θ is the rotational angle (the third dimension, which is the downward pointing angle from the suspended laser head down to the surface is neglected for simplicity).

The gimbal rotates – without loss of generality this is assumed to be in the anticlockwise direction. It has the ability to adjust its cutting angle and orientation but only in the direction

it is rotating. Projected into two dimensions, or looking down directly from above, this means that r and θ vary as the gimbal rotates.

The gimbal can adjust the speed of its rotation – for initial simplicity assume the options are fast, slow, and still. Whether the laser is to be turned on or off is a decision available to the controlling agents. This is not included in the first version of the model, but one might assume that when the automaton is stationary the laser is turned off, and when to turn it on or off would be controlled by a separate agent from the one controlling the automaton's movement. This is an instantiation of the example application area discussed in detail in Chapter 5.5.1 and illustrated in Figure 17, only with the control of the laser in this example taking the place of the 'applicator control' decision process in that general statement of the problem.

Once the workstation is configured, tasks will be repetitive episodic tasks; that is, identical patterns of behaviour completed within a period and repeated according to a defined schedule. However, each individual automaton has a different task and these may not take the same amount of time to complete, requiring slack time for some of the automata. Furthermore, the amount of time each individual task is expected to take could be aperiodic with respect to each other.

With more than one automaton each deploying a separate laser head, the requirement is for them not to collide with each other; that is, when they would otherwise cross each other's path, they should be able to adjust their position and orientation to maintain the path of the laser along the required arc.

The mathematical construction of this problem is therefore as follows:

- The state space is a discretised torus parametrised by discrete (r, θ), where r ranges from r_{min} to r_{max} based on the orientation of the laser head, and where θ ranges from 0 to 2π radians.
- The action space is a set of seven distinct actions: Rotate slow; Rotate in slow; Rotate out slow; Rotate fast; Rotate in fast; Rotate out fast; Remain stationary.
- Task is a requirement to trace out a parametrised curve in the $x - y$ plane, with a completion time of period t (as it will be repeated episodically).
- Each agent will have a different task, including a different period, and optionally may use a different state space model (although this has not been used in the initial experiments). The action space remains the same throughout.

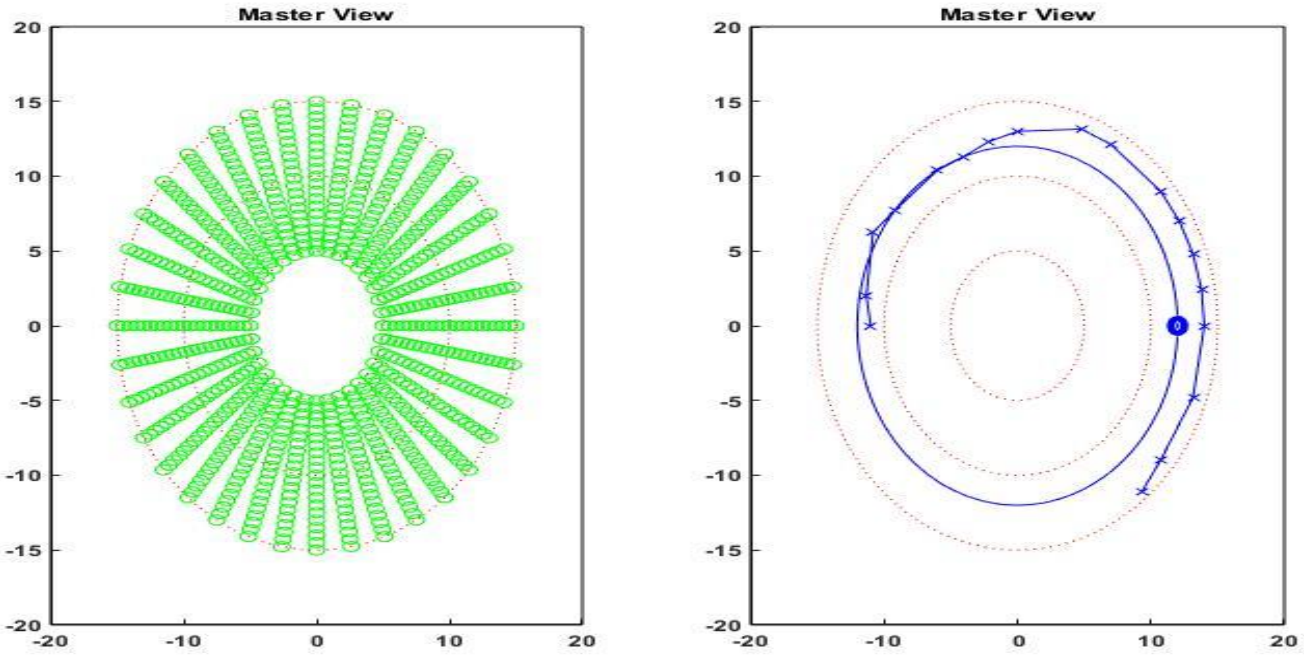

Figure 43 – Discretised Toroidal State Space for Initial Simulation, Single Agent Learning Arc Progression

This model needs to be discretised in order to be suitable for solution by a reinforcement learning agent. Therefore a lattice of (r, θ) points around the torus is the spatial state space (see Figure 43). The density of the state space is a variable to be decided, trading off the increasingly accurate reflection of the true physical state space versus the exponentially increasing computational burden associated with solving the reinforcement learning problem. A number of state space densities have been experimented with during the course of this work.

For time, the natural method for discretising time intervals is by defining them as a series of unitary time steps. That is, if a task has period T then the natural discretisation for initial studies is to have T increments of 1 time unit. In general application, the exact scaling and consequentially the representation of time as a state depends on the properties of the particular system and task.

The action space is already discretised, therefore given the state space model for this problem is (r, θ), the extended Q-space for Q-learning is (r, θ, a_t), where a_t is the action selected at time t.

The other elements that need to be defined in a reinforcement learning problem are the state transition model and the reward scheme – that is, the mapping from state-action pair (or a point

in Q-space) to a new state, and the mapping from a state-action pair to a real number R, which define what state transition will occur following the execution of an action, and a reward enumerating how good it was relative to system objectives. Transition models are in the general formulation stochastic, however in a tightly constrained problem such as this, where there is little probability of the robot physically malfunctioning during normal operation and producing entirely the wrong behaviour, it is reasonable to adopt a deterministic transition model and therefore define explicitly what the state transition is within the state space following a given choice of action.

The reward scheme is the mechanism by which the agent differentiates good actions (in the sense of moving it towards its goal or objective) from bad ones (in the sense that some actions are ineffective or even counter-productive). The reward scheme must therefore encapsulate what the goals and requirements of that system actually are. A correctly implemented Q-learning system will optimise its behaviour to that promoted by the reward scheme – it is therefore vital to ensure that the reward scheme is incentivising the right behaviours in the first place. In this case, the objective is to maintain a trace of a defined curve, and to maintain this trace on an episodic schedule. After some initial experimentation, the reward scheme chosen was as follows: all points in Q-space are initialised with value 0, and all actions that leave the system in a state from which is unable to 'see' the arc it is supposed to be tracing earn reward 0. Any actions that leave the system in a state from which it can see the arc earn reward 1. Reward 2 is given for actions which leave the system in the 'ideal' position (i.e. directly tracing the arc, or in a 2D projection being directly overhead). Reward is then scaled by where in the period the system currently is. If the action chosen has left the system trailing where it needs to be in order to trace the arc on schedule, reward is reduced by a factor proportional to the degree of delay. The proportionality is to ensure that if a system is tracing the arc behind schedule, it is incentivised to catch up and disincentivised to fall even further behind, which is important if an automated system is to recover from interruptions by itself without having to stop and rest the whole line (or part of the line). In this manner, reward can only be maximised by tracing the arc as closely as possible in the time period prescribed. Maximum reward can only be obtained by correctly tracing out the arc and completing it on schedule.

Policy is the mapping of state to action that a system is currently following. An initial learning or conditioning phase will have an exploration factor which determines the (non-zero) probability of sampling any given action from each state. After a sufficient period of state space exploration and reward propagation, the policy can be defined as the optimal action to

take from a given state (i.e. the action corresponding to the maximum Q-value). This could potentially be updated or refreshed, perhaps with learning on going in the background – indeed this is the mechanism that enables Q-learners to be robust to dynamic environments.

In isolation, this is a conventional Q-learning problem. Although there are many additional variables to take into account (e.g. quality scores) in the Doltsinis paper [28], this simple representation of the space-time geometry of the problem should still be sufficient to demonstrate the efficacy of Q-learners. There are also clear qualitative similarities with other traditional machine learning and Q-learning problems, such as Watkins' navigation agent that motivated the original introduction of Q-learning approaches [67], not to mention clear similarities with the parallel decision making discussion and exemplars from Chapter 6.

The feature that will make this more than just another example of a Q-learning navigation problem and instead a more advanced synchronised robotics problem is the requirement for the Q-learning agents to adapt their behaviours in response to obstructions that they cause each other. In other words, despite anticipating interactions between agents, no explicit guidance is pre-programmed into the agents in advance for what to do, allowing the learning agents to solve this by themselves.

In this initial model, interactions consist of a sense-and-avoid routine where, before each action selection by the learning agent, the set of allowed actions is pruned to remove actions that could bring the automaton into a collision with any other entities detected within the locally neighbouring states. There is no communication between agents, rather there is a reactive redefinition of the internal logic. Thus, when there is a potential collision, which is guaranteed to happen at regular aperiodic intervals (with the model designed to ensure it), the agents will find that they do not have the option of the action they would have chosen in the absence of the other entity, and must instead conduct online recalculation of the best alternative action to take. Action filtering in the event of an unexpected obstacle impeding what would otherwise have been an optimal action under the local control policy was seen in the detailed worked example developed through Chapter 6 of this thesis, specifically during the discussion of variable operating regions accompanying Figure 36 to Figure 38 in Chapter 6.4. This is now a real example of the phenomenon – otherwise optimal actions suddenly becoming undesirable (or outright being removed, if only temporarily, from the set of selectable actions) by virtue of one of the other systems with which one is interacting becoming an obstacle.

To demonstrate the potential efficacy of Q-learners to the synchronised automata problem, it is necessary to demonstrate that:

1. These techniques can be applied in this way and still converge to effective control policies despite perturbations caused by complex interaction; and
2. That the policies generated still deliver appropriate behaviour within the defined geometric constraints.

Figure 44 below provides an illustration of the learning agents captured mid-learning phase. Four learning agents are seen learning their assignments, all following the same arc but with aperiodic schedules. Path histories represented by line plots, which are coloured differently purely to help differentiate them when overlaid on the master view. Larger markers on the reference arc represent where each agent should be at that point in its own cycle. The four individual local perspectives of the individual agents, with no awareness or cognisance of each other, can be seen, in addition to the composited behaviour of all four agents together as a global collective view (noting that during the learning phase captured here, this is not at this point in time a meaningful representation of an overall system behaviour – but it will be once the agents are operating 'for real' based on their learned policies).

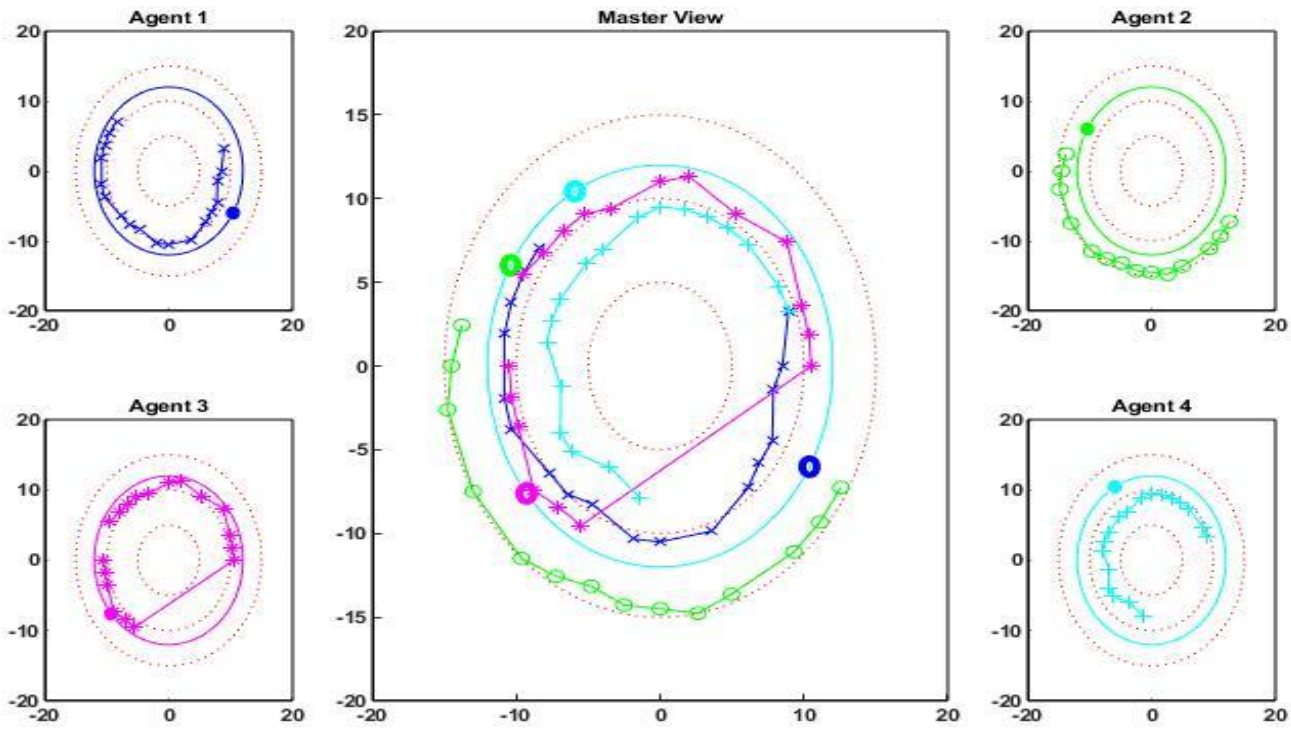

Figure 44 – Four Learning Agents, Local and Global Views of Arc Progression

Note that enforced state space exploration is optionally enabled in order to speed convergence, accounting for the large jump in the path of Agent 3. This is not an allowed behaviour in a control policy, it is merely a tool to accelerate the learning phase.

8.2.2 Larger Scale Problem

Following this initial implementation to demonstrate the efficacy of Q-learners, a more complex problem more representative of robot automata undertaking different activities but with inevitable overlaps or moments of conflict was required. A larger model was designed featuring an increase to ten agents tracing out arcs of different shapes, some going clockwise and some anticlockwise, with arcs that were designed to partly overlap, to enforce potential conflicts between the agents' goals and thus their learned optimal control policies.

Ten agents are simulated as operating within this lattice. Each agent has a particular pattern of motion that it is required to learn. They are not programmed with the specific path they are required to follow nor any awareness of the other agents that are operating robots around them. During a learning phase, the agents follow a series of state-action transitions (random walks on the lattice) and rewards are calculated based on the outcome of each agent's state transition.

Note that the simulation environment and design logic is the same as with the initial simulation and is not described again. The differences here are that:

- The number of agents increases to 10.
- Toroidal state spaces are replaced for convenience with a regular discrete square lattice, parametrised in the usual way (50 by 50).
- Tasks continue to be to trace out an arc to a defined episodic schedule, but using polygon shapes which more naturally align to a grid lattice, and with some agents required to go clockwise and some anticlockwise in order to increase the complexity of the required learning and of the learned behaviours.
- Based on learning from the initial simulation, some novel extensions were made to the standard mathematical formulation to capture some particular characteristics of this problem. These are detailed in the next section.

The overall test environment with ten Q-learners taking random walks (again denoted by using different path colours) to learn the value of different state-action pairs in the state space is illustrated in Figure 45.

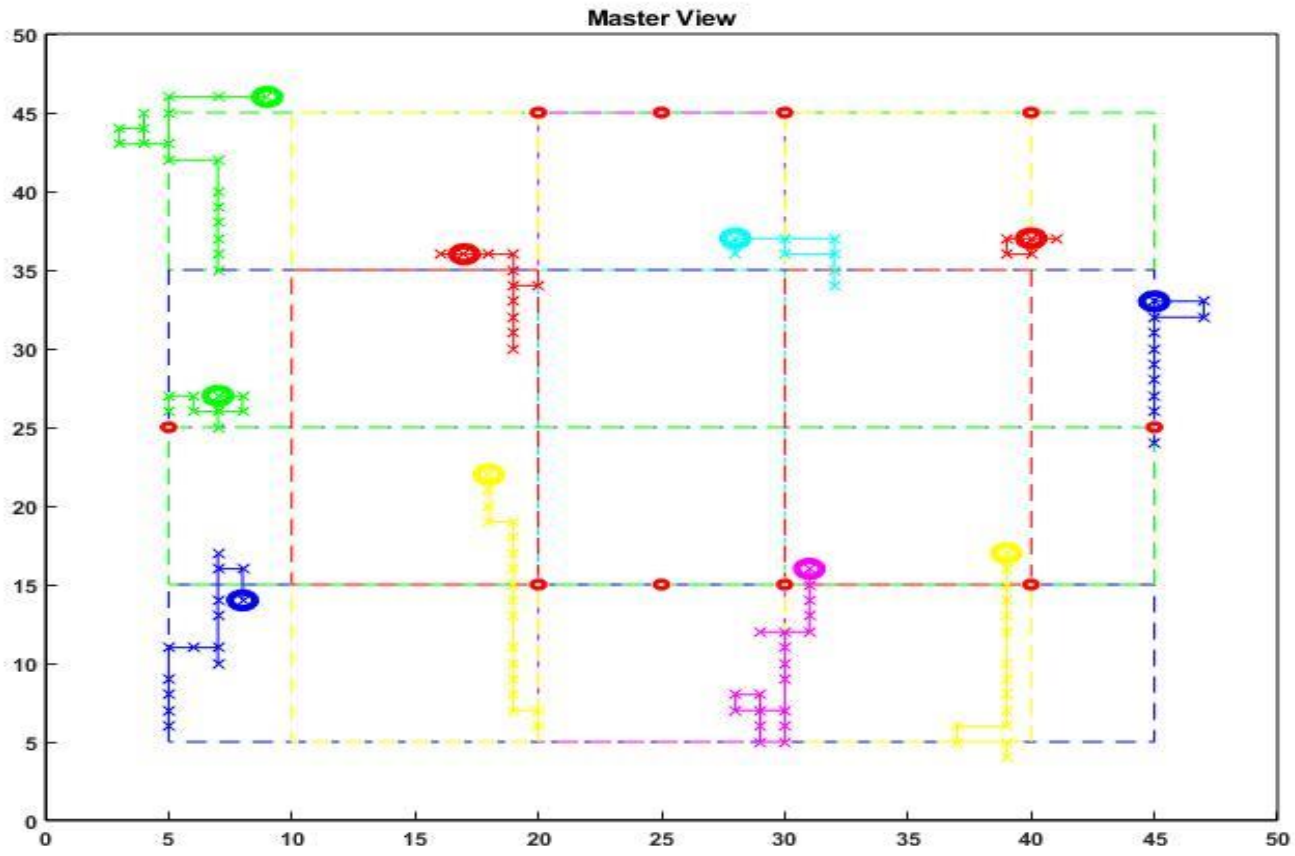

Figure 45 – Visualisation of 2D Agent Training Environment

8.3 Novel Aspects

In order to capture the required behaviours of the ten learning agents, a number of novel features were added to the standard mathematical formulation of parallel decision making within a multi-agent system as described across Chapters 5 and 6.

8.3.1 Temporal Extension to the State Space

Each agent will be expected to learn to complete its arc to a specific time scale. This means that a state is only the correct state to be in at a specific time, and the reward received for a

move into this state must reflect this if the agent is to learn from it. Hence the introduction of a temporally extended state space:

$$\{S\} \rightarrow \{S^*\} = (s_1, s_2, s_3, \dots, s_N) \times T$$

This reflects the need for a point in state space to have different values and hence action selection strategies at different points or epochs within the episodic window to complete the task. Within a Q-learning context, it is no longer the value of action a when in state s that determines the optimal control policy, but the value of action a when in state s at time point t, or equivalently the value of state-epoch pair (s, t), noting that this will in general lead to different 'best action' selections from the same state at different time points in the episode.

In terms of implementation, this adds an extra dimension to the Q-value array and inevitably slows learning time. However it is necessary to differentiate between similar spatial positions at different times because states have different intrinsic values at different points in the episodic period, based on how likely it is the agent can complete its task from that position in whatever time remains.

There are things which can be done to reduce the impact of the significantly increased size of the state space. For example, if it is known when the system should have reached a particular point on the arc, rather than defining all of the possible time steps as individual points in the extended state space, it might be more efficient to partition the temporally extended state space into only three state-time points – the given point in space at 1) a time before when the system 'should' have reached it, 2) the time when it 'should' reach it (perhaps including a small interval either side for flexibility), and 3) any time after when it 'should' have reached it. In this way, although the state space has been temporally extended, it is not by nearly as much, limiting the impact on computational run-time.

Whether this proposed partitioning or a more finely grained alternative is appropriate depends on the characteristics of the system in question and what the system designer is trying to achieve. The problem with the above simplification is that compressing the entire range of times 'before' and 'after' into a single point in the temporally extended state space is that it assumes a single action will always be the most appropriate at any specific time within that period. If there is an unexpected or unplanned interruption, for example another automaton regularly impeding one's path, then a system may learn that at one particular point during this long time interval something different should be done and associate that alternative strategy with that particular point in space-time. It would not be able to differentiate the greater value

of a different action at that particular point had that point been subsumed into a lengthier time interval for the majority of which the unexpected obstacle does not arise and the alternative strategy would not be (and would not need to be) recognised. For this reason, it was not appropriate to make such simplification in the model developed and the computational time penalty from the significantly expanded state space had to be accepted. In general application, care should be taken if considering such compressions of the temporally extended state space to determine whether or not this would be similarly counterproductive, by losing the ability to learn the value of unanticipated policy differences at particular space-time points in the temporally extended state space.

8.3.2 Action Filtering

As the premise of this work is that the agents should be simple and be unaware of the other agents, it will be necessary to incorporate some sort of environmental signal to guide agents in the case of possible collisions. While this could be achieved through giving a heavily negative value to reward R when a state transition either leads to a collision, or leads to a state that has increased the probability of collision, this is an unsuitable approach in this case. Firstly in practical terms an operator will probably rather prohibit such actions so as to prevent damage to physical equipment. Secondly, on a theoretical level, if a negative reward is received following a particular transition that particular transition is devalued, even when in the absence of the obstacle it would actually be the correct move. One has to be careful with rewards that they accurately capture the desired behave, and any potential confusion should be avoided.

An alternative approach is to use an action filter, as has been alluded to earlier in this thesis. It is not used in the learning phase, as the intention is for each agent to learn what they should ordinarily do – their optimal policy. It does not matter if they crash into each other in a simulated virtual training phase, indeed it might be beneficial to let them do so, to learn what they should not do. In the subsequent execution phase however, an action filter serves to create a subset of the usual action space:

$$A^T = \{a_1, a_2, a_3, \dots, a_N\} \backslash \{a_i, a_j, \dots\}$$

This subset contains only those actions which would not cause a transition to a potential collision state, i.e. $\{a_i, a_j, \dots\}$. It is assumed that the robot has a simple motion detector or equivalent device, which is alerted to the presence of something the system could collide with, thereby warning the agent which actions are not to be taken at this time and filtering them out

from current policy options. It can be assumed this is only a temporary exclusion for as long as the obstacle remains, but in general the actual action space from which actions can be selected will be the filtered action space A^T with only the currently permitted subset of permissible actions. A^T is a subset of action space A and is a set with variable membership whose composition changes in time, continually updated and refreshed depending on what is detected in the local external environment. This does not constitute the introduction of coordination between the agents since any external objects could also cause obstructions, and the agent would have no communication or data exchange in either case.

If either no actions are filtered or those that remain after filtering include the action that the calculated optimal policy would have selected anyway, then behaviour will remain the same. The novelty here arises when the optimal action is amongst any that are filtered out. This is where the Q-learning approach is powerful, because it is a simple matter to obtain from the agents' Q-value tables what the new optimal action should be based only on this subset of the action space. Recall from Chapter 5.6.6 that given an optimal policy:

$$\pi^*(S, A) = max_\pi\big(Q_\pi(S, A)\big)$$

The optimal action (or a set of equally optimal actions) will be:

$$a^*(s) = argmax_{a \in A}\big(Q_\pi(s, A)\big)$$

What action filtering does is limit this to a subset of values already contained within the optimal policy and so, importantly, requiring no further calculation (other than maintaining any on-going learning iteration in an online learning implementation, which would be occurring anyway). The filtered optimal control policy is therefore:

$$\pi^{*,T}(S, A^T \subseteq A) = max_\pi\big(Q_\pi(S, A^T)\big)$$

The optimal action post filtering then follows logically as:

$$a^{*,T}(s) = argmax_{a \in A^T}\big(\pi^*(s, A^T)\big)$$

Note that $a^{*,T}(s) \neq a^*(s) = argmax_{a \in A}\big(\pi^*(s, A)\big)$ for $a^*(s) \in A^T$.

This is a form of policy switching based on local observation which enables the agent to adapt and follow a different plan to avoid the hazard without needing to re-learn or re-sample.

Essentially, action filtering and policy adaptation in this manner utilises the volume of data contained within the Q-value tables within the agent but ordinarily unused. This is the attribute of Q-learning that makes it a suitable candidate for synchronised autonomy, because the agents

should be able to implicitly learn to avoid each other through learning a large set of 'contingency' policies without any additional knowledge or training time being required.

It should be noted that the optimal action post filtering may not be the same as the optimal action under the unfiltered optimal policy, nor will it necessarily be the same as the preceding optimal action in the case of one filtered action space changing to another filtered action space at the next time step. One can imagine situations where, to give a simple example inspired by the worked example from Chapter 6, an uncooperative or hostile entity in the environment blocks one's path. Imagine the optimal policy is to head, for example, east, but following action filtering this is no longer an available option, and so the system automatically adopts the 'best remaining' policy, which is to head instead, for the sake of argument, north. But then soon after, potentially as soon as the very next time step, another obstacle appears to the north, prompting another action filter, resulting in a new optimal action to head, say, south. What result action filtering will have depends on the environment around the agent and is something outside of the agent's control. There is therefore no guarantee that the changes in optimal policy due to action filtering will be in any sense 'close' to the true optimal policy action under the unfiltered action set. An avenue for future research might be to investigate the theoretical implications of action filtering like this on some of the analytical results in the wider machine learning literature. For present purposes, what is important is that it is producing useful and observable behaviour in practice.

8.4 Simulation Results and Analysis

8.4.1 Introductory Problem

The rates of convergence to the *a priori* calculated optimal control policy for the four agents both in isolation and with the ability to detect each other's presence are shown in Figure 46. An example of the behaviour of the synchronised automata is captured in Figure 47.

Regarding convergence, Figure 46 shows the convergence of learned policy to the optimal control policy that was calculated *a priori*. Solid lines are the convergence rates of agents in isolation (i.e. ignoring potential interactions) and dashed lines are those allowing for perturbation. It is clear that allowing for interactions and resulting path perturbation usually causes policy convergence to take longer (as any deviation results in some state-action pairs not being sampled as they might have been, requiring further sampling), but it is clear that the

effect is small as a proportion of the time the simpler process required anyway without this additional consideration. In the one case where this did not happen (the line coloured light blue), the policy convergence rates were almost identical, and the difference is believed to be a consequence of randomised state space exploration (i.e. a simulation artefact rather than a property of the underlying system). A more extensive simulation averaging these results over many random trials is recommended in order to verify this conclusion.

It is clear that introducing the ability to recognise obstacles has not significantly altered the convergence rate for the agents. This is an encouraging result, as it indicates that the novel elements applied to model the interactions are not affecting the overall learning performance of the system. Simulation run times were also similar, and can surely be improved upon following deeper consideration of the sampling strategies and state space models.

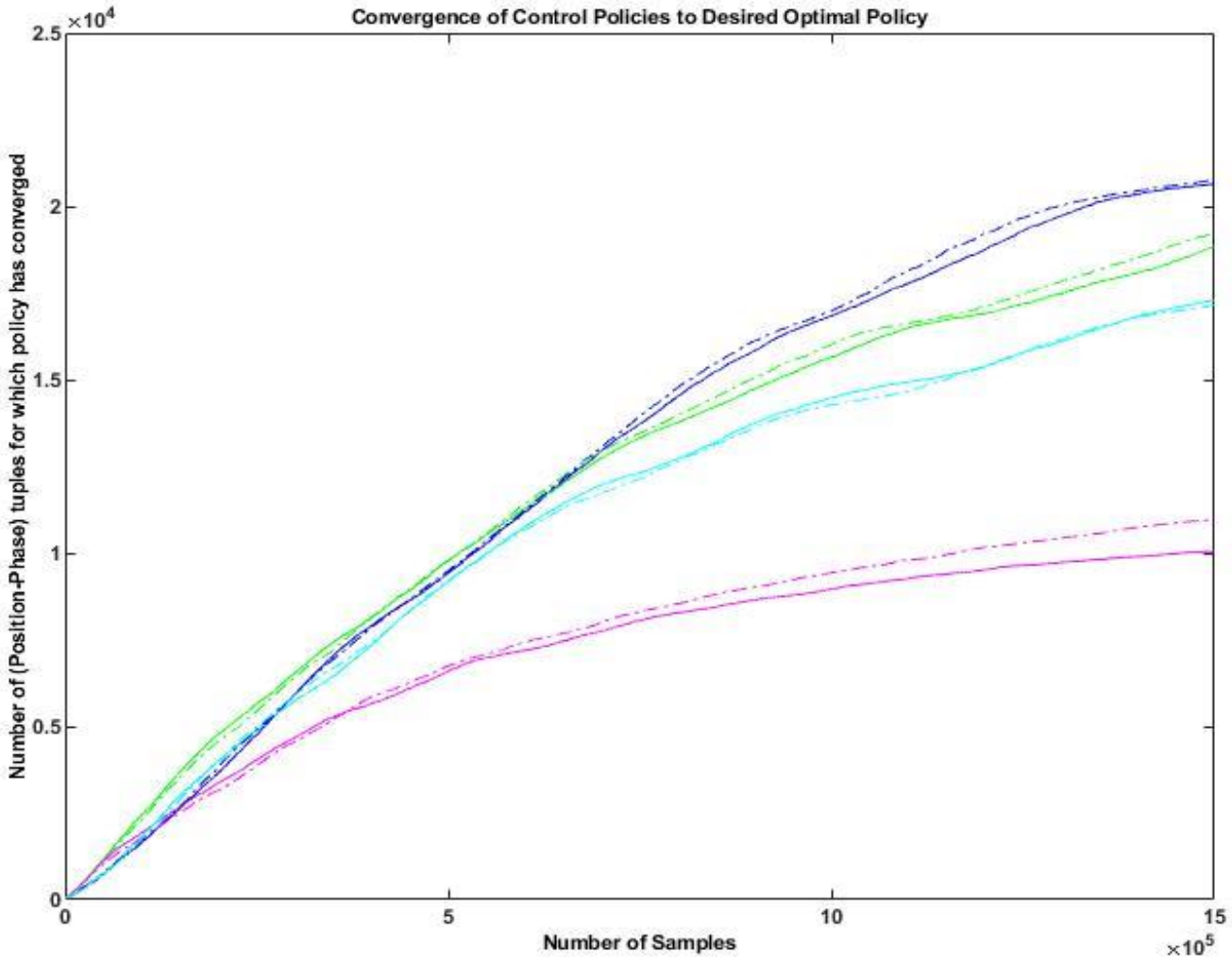

Figure 46 – Convergence of Learned Policy to the Optimal Control Policy

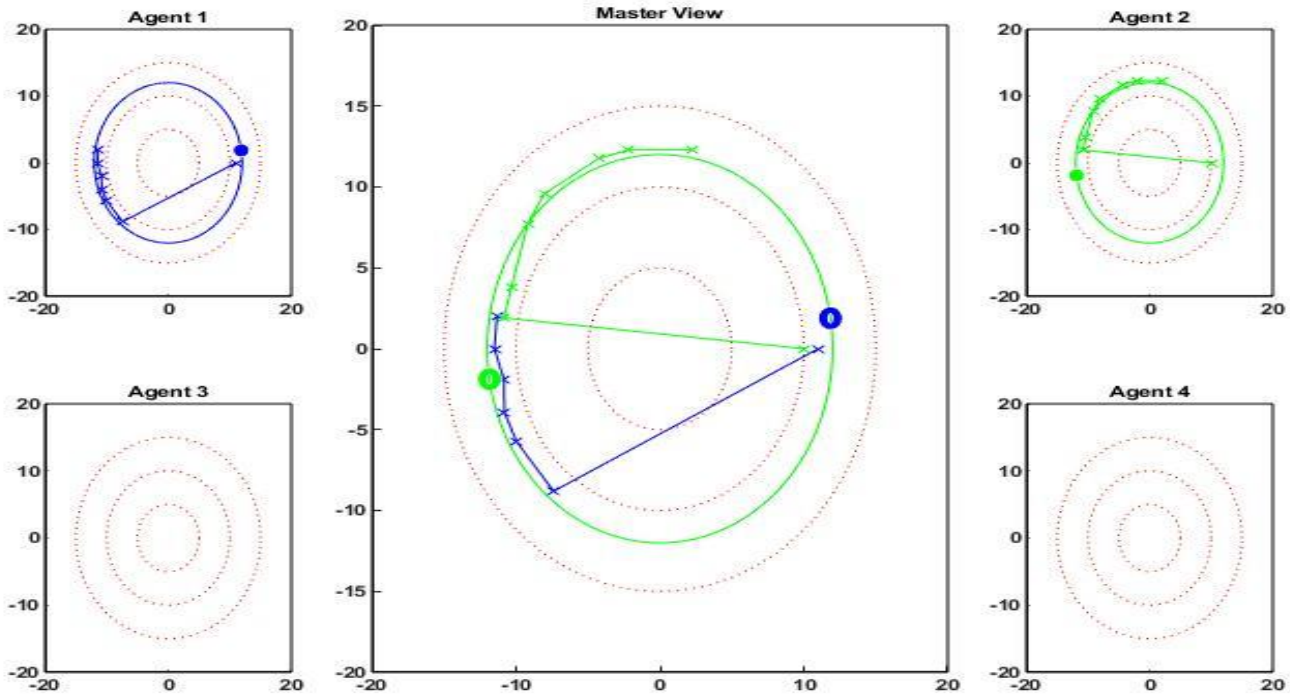

Figure 47 – Two Learning Agents Arc Tracing (with Action Filtering)

Figure 47 shows two agents learning their assignments with action filtering enabled (Agents 3 and 4 were also part of the simulation but have been omitted from the figure for clarity). The agents are still in their learning period, hence the sudden jump in the agents' paths as they re-initialise for another trial.

Regarding Figure 47, note the path of Agent 1 in the lower left portion of the arc, where it can be clearly seen to have stepped aside in order to bypass Agent 2. Such a perturbation is within the observation range of the automaton and within the geometric constraint or bound of an operating annulus centred on the arc. It demonstrates that the policies that are being learned are enabling the agents to adapt to the presence of other automata with no knowledge other than of their own task and that there is an entity to avoid. This is consistent with the convergence results which show that, despite these aperiodic perturbations, the agents are still able to learn stable control policies that deliver behaviour slightly perturbed from the optimal policy but still within the relevant constraints. This is encouraging evidence that the desired application of interacting learning agents to the synchronisation of automata can be achieved with relatively simple models, without substantial system configuration and extended ramp up time.

8.4.2 Larger Scale Problem

The test environment with ten agents illustrated in Figure 45 was used to generate a number of Q-value arrays and optimal policies. The ten agents learn concurrently, but without awareness of each other. It could not be done like this in practice because there would be many physical collisions and a lot of potentially damaged assets, but in simulation this is beneficial during the training phase. The arcs were chosen to be spatially dispersed across the lattice, but with concentrated areas and overlapping portions of arcs to ensure robustness under action filtering would be tested. These arcs were also scaled to ensure an optimal policy to complete each arc will require the selection of both single and double steps, to ensure action space coverage.

Q-value arrays and optimal policies were produced after 100,000 learning steps, and then at increments of 100,000 steps up to 2,000,000 steps. It transpired that the agents had learn to follow their arc to the required schedule in less than 1,000,000 steps. Without incorporating collisions, this is an expected result since the problem formulation (learning a rectangular angular walk) is not especially difficult. The reason it takes as long as it does is the learning of the schedule – the extended temporal dimension to the state space significantly increases the learning time both due to the overall larger number of states, but also due to the longer time to even reach portions of the state space as consecutive states visited will tend to be clustered.

During the execution phase, collision detection is enabled. This means that the optimal policies that the agents have learned might now be defining optimal actions to take from a given state which will actually be forbidden at particular points on the arc (and the test environment was deliberately set up so that a number of path conflicts were inevitable, to force this to occur).

A screenshot illustrating the observed behaviour is provided in Figure 48. This was produced with the agents following the policy produced after 1,000,000 learning steps, and it is clear that the agents have by this time learned to follow their intended arcs. There are also some instances where a simple digression from the arc is clearly distinguishable – these occur when the optimal action to continue along the arc is unavailable and an alternative action has to be selected instead.

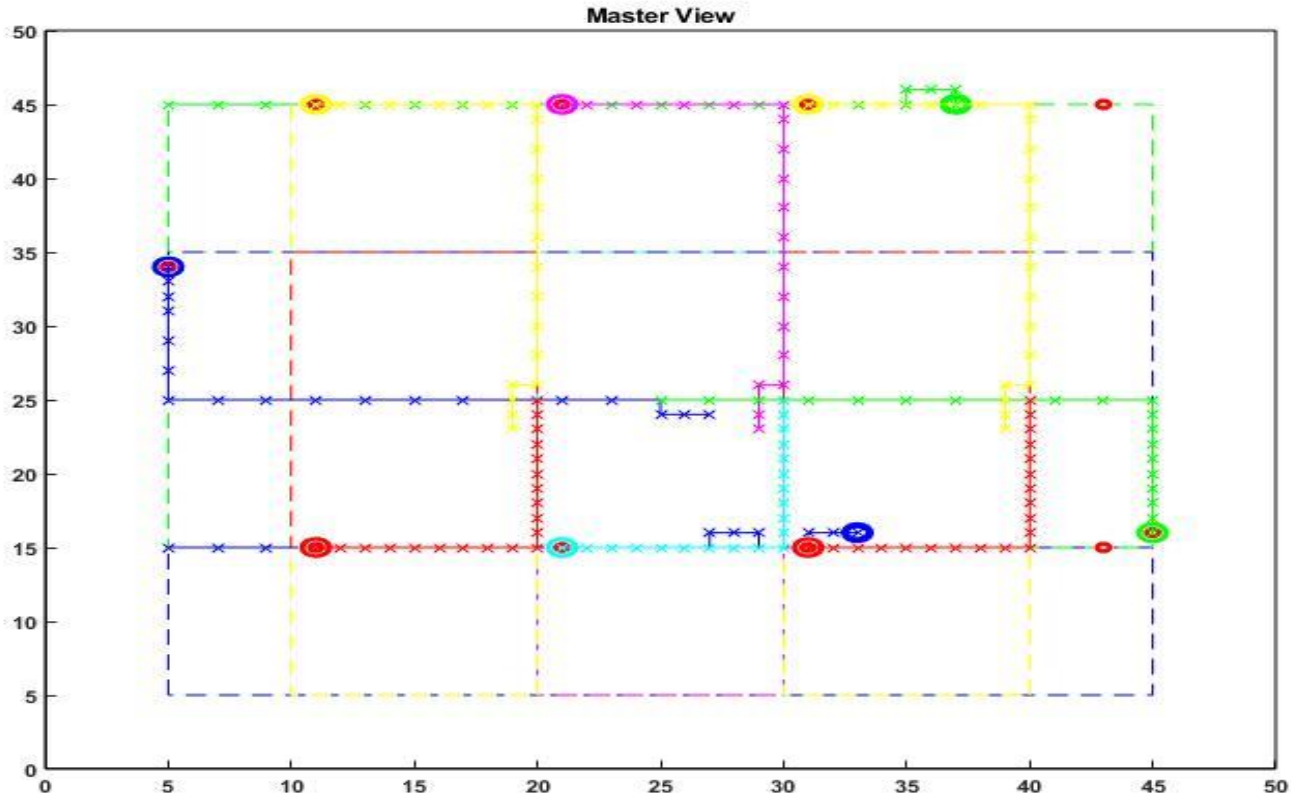

Figure 48 – 2D Visualisation of Trained Agents Performing Learned Path Tracking with Controlled Deviation

An unexpected emergent property was that both colliding agents might respond with the same avoiding move and thereby still be blocking each other – similar to the situation where two people meet in a corridor and both move the same way to avoid each other. To avoid these situations, as the agents are internally numbered from one to ten, the lower numbered agent was programmed to choose and then make its avoiding move first, so the higher numbered one did not need to. This simple fix was to allow these simulations to run to completion, but may be indicative of the kind of practical fixes that might be employed with real automata.

There are two primary criteria that determine how well the chosen method is performing: (i) how well are the agents performing their task, and (ii) how robust is their performance to perturbation. The former was investigated in the initial experimental simulation work but the example therein is acknowledged to be simplistic, in that all the agents in that work were trying to do the same task at different speeds. In this more complex example there are more agents

and more conflicts, and it is not trivial to determine in advanced when and where any diversions should be and what performance implications these might have.

Figure 49 and Figure 50 summarise the key findings regarding these two criteria. Figure 49 illustrates the average mean position error across an episodic period, averaged over 100 trials. For clarity only the results for four agents are illustrated, as is the performance for three calculated optimal policies – the policies generated after 100,000 learning steps and after 1,000,000 learning steps (which has converged to optimal as it is clearly seen to deliver the required behaviour), and an intermediate policy produced after 500,000 learning steps.

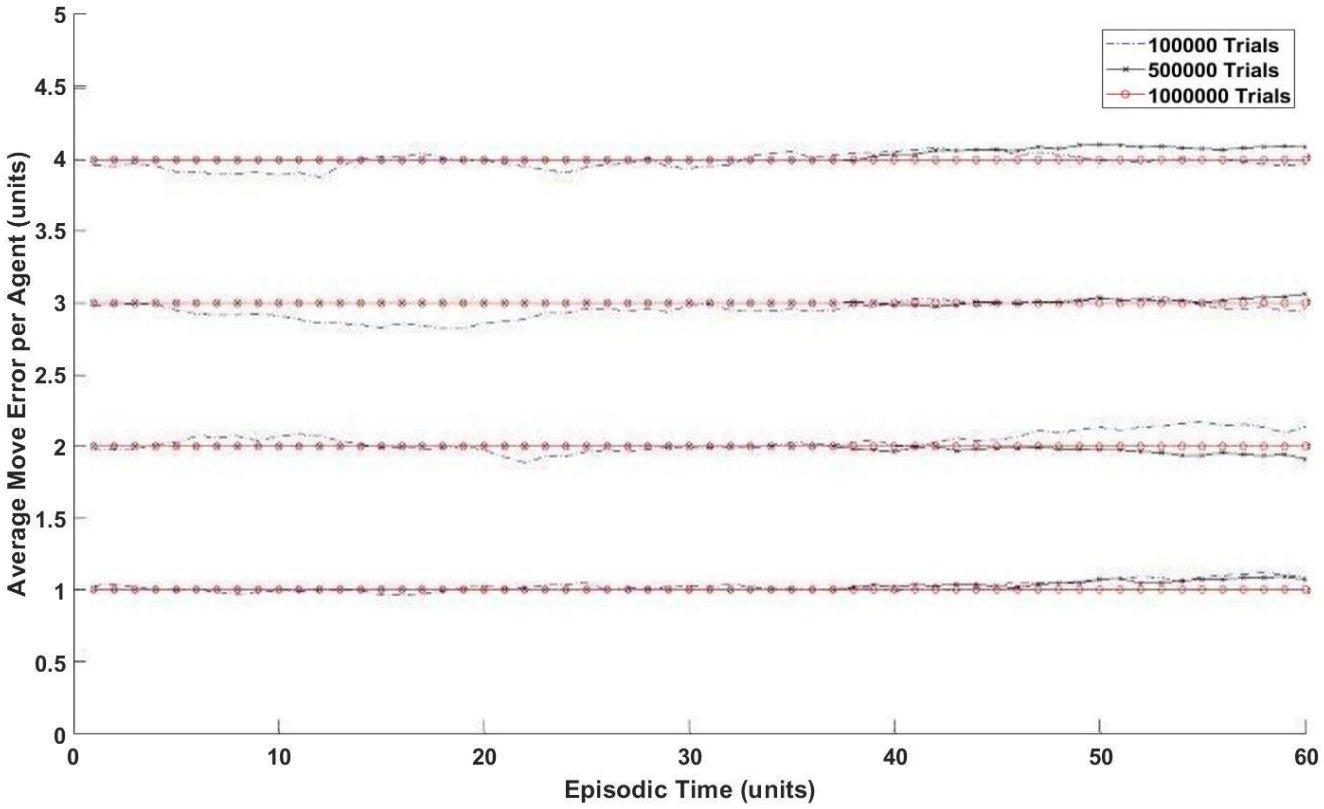

Figure 49 – Average Mean Error per Agent Across One Episodic Period per Learning Period over 100 Trials

Note that the results in Figure 49 are limited to the first four agents and three examples of policies for clarity.

These cases are sufficient to illustrate the anticipated result (convergence to the desired behaviour), and to capture the timescale required (on a standard desktop PC running MATLAB 2014a, these policies were generated in approximately 20 minutes, 90 minutes, and 3 hours

each). For each agent, it is clear that the networked Q-learning approach with its novel adaptations is capable of solving this problem and producing coordinated behaviour.

Of more importance is Figure 50. This represents the equivalent data when in execution mode with collision avoidance enabled. The 1,000,000 step policy is clearly seen to be capable of making small digressions from its optimal policy. These steps are when an obstacle was detected and a simple sideways move was undertaken until the obstacle had passed. When the action space returned to its full size, the agent was able to immediately return to its original task. Agents 1 to 3 all display examples of this, although Agent 3 was not able to subsequently complete its task on its original timescale. This is because it faced several obstacles and deviations that used up enough time that it was not able to recover (i.e. it was not able to fully 'catch up' with where it would otherwise have been) in what time remained in that episodic period. This could be resolved in future work either by extending the episodic period, or by allowing the agents three or more step moves rather than the current limit of two.

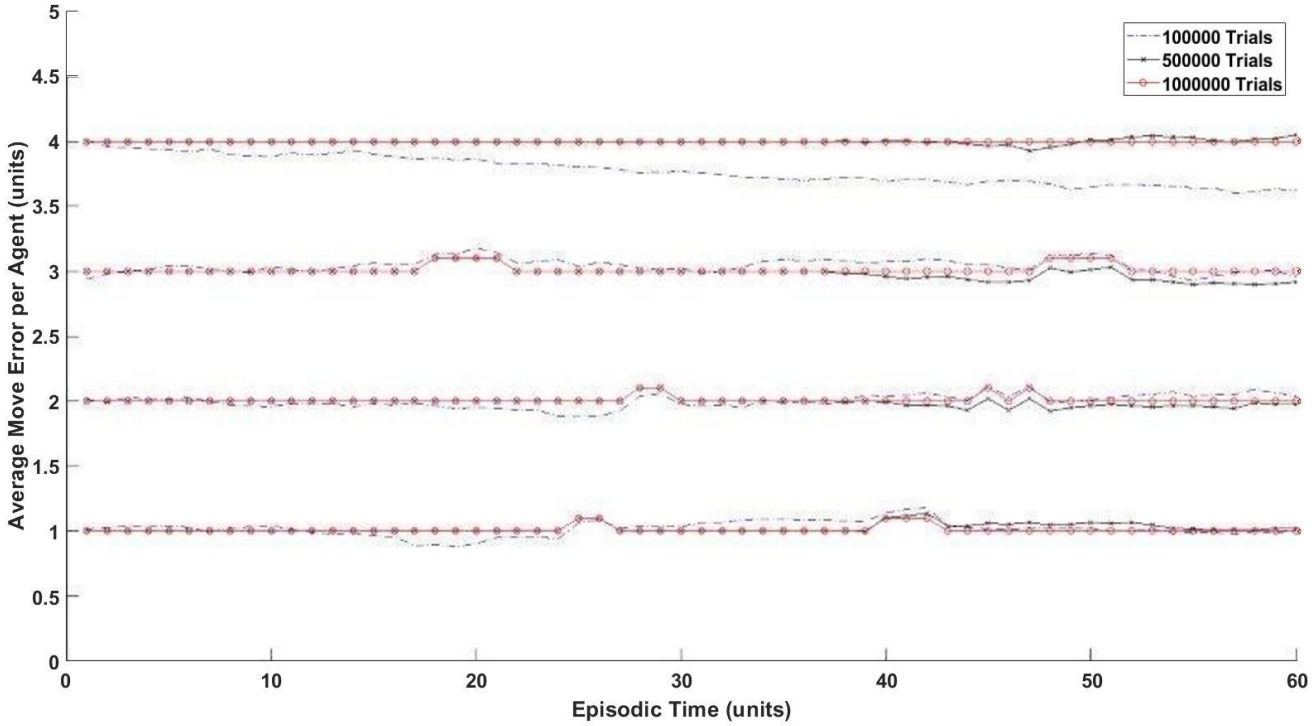

Figure 50 – Average Mean Error per Agent Across One Episodic Period during Execution Phase over 100 Trials

Note that the results in Figure 50 are limited to the first four agents and three examples of policies for clarity.

Note that in both figures, the deviations in the vertical direction are the average mean error per step across the episode. Clearly this is greater with the less mature policies that had shorter learning periods. It is important to recognise that this is the average error incurred at that point in the episode, not a cumulative error. The problem with a cumulative error calculation in the given geometry is that different actions allow the agent to move backwards and forwards. One can make almost every step incorrectly and not trace out much of the arc at all, and yet end up in about the right place. Hence the cumulative track error is not a useful metric in this context, and the average error per step is presented instead. Hence although the errors appear small, it should be understood that when the error line is consistently offset from the optimal policy line, the physically observable error could be growing exponentially.

Chapter 9

9 Conclusions and Further Work

9.1 Conclusions

This thesis has presented a fusion of systems design process and underpinning mathematics to construct a novel method of decomposing complex systems into networks of Markov Decision Processes, adaptable for many varied system organisational structures via hierarchical and parallel decision processes. The underpinning mathematical background, including some original contributions from the author prior to the commencement of this PhD, has been extended as part of this PhD with further original contributions to enable this particular application, and the result is a novel methodology with analytical potential for modelling and, in due course, controlling the behaviour of complex multi-agent systems.

It is accepted that for many traditional systems, this would be considered unnecessary overkill, and that existing techniques and methodologies are quite adequate. The usual drawback of reinforcement learning techniques, and of using MDPs in general, is the required overhead in designing the models, selecting state and action representations and reward schemes that accurately reflect the important characteristics of systems and thus incentivise the right behaviours. The learning or conditioning phase with such techniques is sometimes inefficient by comparison to alternative methods.

However, the premise behind this thesis is that, when dealing with complex multi-agent systems, most of these other techniques cease to be applicable or effective. From the engineering perspective, system design, assurance and lifecycle management techniques and practices struggle to deal with emergent properties within systems, something that readily arises in complex multi-agent systems. Assurance of autonomous systems is already a problem that has stymied the uptake of technology that was once attracting significant funding and research effort and had many potential exploitations routes, due in large part to the on-going inability to be able to prove they can be relied upon to perform safely. Combining this problem

with the assurance of autonomous agents with a lack of tools and techniques to apply formal analytical methods to such systems creates a space where new approaches are required. Whilst this thesis has not attempted to prove the approach is a perfect solution for all manner of problems, it has built up a proposed approach to complex multi-agent system design and control based on aligning systems science principles with systems engineering practical requirements. It is a novel candidate for a problem domain that is lacking in viable alternatives.

From the mathematical perspective, MDPs have been shown to be applicable in areas not widely explored in the literature but for which the author has long believed they had greater potential. The work contained within this thesis has demonstrated how to generalise the mathematical formulation into scalable networks of MDPs that may mirror systems' organisational structures, and demonstrated that at least one solution technique (the reinforcement learning technique known as Q-learning) can be readily adapted to generate solutions to these structured networks of MDPs. This provides a potential gateway to modelling and understanding (and more importantly, controlling) the complex and potential emergent behaviours of multi-agent systems. Again, it is a novel candidate for a problem domain that is not completely lacking in alternatives, but is lacking an alternative that can be used in quite this way and which is aligned with the engineering community's practical requirements.

A simple case study has been developed and demonstrated as part of this PhD, inspired by the synchronised autonomy displayed by production line assembly robots. Restricted initially to 2D simulation for clarity and convenience, models of initially 4 and subsequently 10 agents have been developed showing a multi-agent system approach to such modelling and controlling such problems, where forced interactions between agents tracing arcs represents the potential for inadvertent collision or obstruction by the production line robots. The results reported in this thesis in Chapter 8 (and in a number of external publications listed in the Declarations section at the beginning of this thesis) has demonstrated that networked Q-learning is a novel but viable approach for attempting to produce robust solutions to the synchronised autonomy problem. Incorporating novel features such as temporal extensions to the state space and action filtering allow learning agents to undertake a number of concurrent, overlapping tasks to a fixed schedule. This approach is not a perfect solution – for example, in the published results from the experimental work, it was clear that Agent 3 became unable to complete its arc as required within the episode following enforced deviations due to external obstructions, but there are foreseeable solutions to these problems given more development time.

The usual drawback of reinforcement learning methods such as Q-learning is the relatively lengthy learning time in comparison with other established machine learning techniques. This is worsened through temporal state space extension. However, the attribute that enables rapid adaptation when using a run-time action filter is that Q-learners are maintaining large Q-value arrays full of continually updated orderings of contingency policies, which enables immediate policy adaptation. Essentially, Q-learners can be relatively slow because they are learning and updating a full set of multiple scenarios all at once – a negative if one only needs the optimal solution quickly, but a positive in situations like this where maintaining the ability to adapt rapidly to follow the best available policy (which is itself being updated in real-time) is vital.

In summary, it has been demonstrated how the Q-learning solution technique teaches an agent not just an optimal course of action, but also multiple prioritised courses of action which, when certain actions become unavailable during operation, immediately provide 'next best' contingency plans without any retraining being required. The drawback of applying such techniques normally is that they are often a more onerous means to determine an optimal control policy than other techniques might have been. But in this context these approaches have shown their worth, in that the extra burden on model establishment and greater agent training time results in a family of prioritised and continually refined control policies, and therefore greater resilience in the face of unplanned deviation or obstruction. Demonstrating how traditional weaknesses with a technique can be turned into a significant strength is a valuable result that puts the potential application of these techniques in a new perspective and, it is hoped, makes them more attractive for use in complex systems scenarios once again, not least in application areas such as synchronised robotics and autonomy, but also the challenging domain of complex multi-agent systems more generally.

9.2 Further Work

Throughout the course of this research and the process of compiling this thesis, a great many ideas from across different research domains and application areas have been covered – from machine learning to systems design processes, from the conceptual domain of systems science to practical considerations about simulation toolsets in an industrial time-sensitive environment. The material presented has been trimmed to fit around a core argument about the proposed modelling and mathematical framework for complex multi-agent systems, why this is important for niche problem areas with particular characteristics that are difficult to

tackle with techniques in the current engineering or mathematical toolkit, and conducting some relatively simple 'proof of concept' simulation work to a) demonstrate that networks of Q-learners can be applied in new and novel applications such as synchronised autonomy, and b) show that the techniques and models have demonstrated basic proof-of-concept that complex emergent behaviours within multi-agent system can be implicitly 'designed in' and that agents can be seen to adapt to unexpected events in a way that still remains within acceptable operating bounds, and from which the agent can attempt to recover lost time and return to its original assignment and schedule.

It has not been possible to include everything within this thesis:

- Related areas of interest such as software architectures for the agent implementations, and some of the anticipated impacts on engineering tools and methodologies, has had to be left out of this thesis in order to appropriately focus the thesis on the core argument.
- Two alternative applications areas were partially developed before identifying the industrial exemplar from within Warwick Manufacturing Group's research interests; the work that was undertaken is documented for the record in Annex B, but these models have yet to be taken further into detailed simulation or analysis.
- A significant area for further research is to identify complementary mathematical techniques to address the system proving and assurance problem; it was beyond the scope of one PhD to tackle both the control and assurance problems, but the work the author has done on the latter (including some background joint work, and some updates and further thoughts produced in the early part of this PhD) are documented for the record in Annex C.

All of these topics present avenues for potentially fruitful research and development activities in future work. In addition to these topics, this final section of this thesis also documents some other recommendations for further research, grouped by the subject matter within this thesis to which they relate.

9.2.1 Proposed Enhancements to the Exemplar Simulations

Interesting avenues for further research relating to the experimental simulations of the industrial case study were identified in the published work:

1. External obstructions that occur randomly and at different points with the state space; an agent cannot learn through habit to avoid these.
2. More thorough parameter sensitivity analysis. Different learning rates and exploration strategies might further speed up learning performance, as might some parallel processing techniques.
3. Increasing the volume of agents by an order of magnitude and investigating any emergent effect in particularly clustered areas of the state space.

There are other avenues for potential extensions or further development of the experimental simulations and test beds. Given that the detailed description of how these multi-agent system models were designed and simulated is presented in Annex A, the recommendations for further developments in this area are covered there, in particular in Annex A Section A.4.

9.2.2 Engineering Process Considerations

The proposed modelling approach has been shown to describe the behaviour of complex systems through the novel representation of the system's state and action spaces, incorporating the expected consequences of interactions, and making the system more robust to the occurrence of complex or emergent behaviours. The extent of this approach's scalability to increasingly complex systems needs further research; computational constraints could potentially limit applicability. However, wherever this limit lies, it is still expected to be an improvement on techniques not designed for complex systems at all, that are not suitable for assessing system failures that do not arise from traceable component faults.

Hierarchical MDP models based on the three-axis decomposition principles have been defined, however the exemplar developed in detail only featured the 'parallel' aspects of decision making. A (very simple) instance of a layered decision process was demonstrated in the author's precursor work, but producing a far more complex model of a system with multiple layers, and parallel decision making on each layer, has not yet been attempted – essentially, for reasons of time and scope, it has not been possible to fully test the scalability and efficacy of the proposed framework in its entirety. This would be the logical next step for this work, to really test the scalability and efficacy of these proposals.

Whether the proposed method is scalable to increasingly large state spaces is a valid concern, not least because of computational overheads. As is usual with such problems, the fidelity of the model is compromised if the state space is too course, but the computational burden could

make the entire process infeasible if the state space is too fine. Important design choices will have to be made early on, which could have unforeseen consequences on the practicality of the proposed approach. It has not been within scope of this PhD, but a recommended future activity is to consider the issues of scale and of the computational burden of the proposed approach, perhaps in the context of one of the exemplars.

Some further specific proposal for future research related to this topic include:

- Formal development of a design process for complex systems incorporating the mathematical insights outlined in this thesis and, possibly, applying techniques from linear algebra to optimise the state space architecture.
- Investigation of where non-linearity may arise in decision hierarchies, possibly resulting from asymmetric connections across levels of the hierarchy, or from cyclic dependencies of components on particular levels.
- Demonstration that the decision control framework for systems of subsystems is equally applicable to collectives of systems (i.e. vertical scalability within the hierarchy), ultimately leading to an integrated hierarchical structure for collectives of multi-layered complex systems.
- Investigation of further and/or alternative algorithmic solution techniques for the decision controller, which may increase the scope and robustness of the hierarchical approach. In particular, MDP planning algorithms might facilitate online refinement of the world model, potentially improving system sensitivity to changing environmental conditions. Another option the author recommends considering further is the work by Barry et al. looking at techniques for approximating solutions to very large MDPs by decomposition into solvable hierarchies [6], to determine whether these approaches can simplify the process of learning solutions to these kinds of problems without accepting too great a degree of suboptimality of solution due to the approximation.

9.2.3 Theoretical Aspects of Decision Process Design

Selecting the architecture of the state space is a crucial stage in the design process. It must answer the basic control system requirements of *observability* (that the state space provides a sufficient representation of the true system state and one has or can obtain sufficient information to understand the system's behaviour without ambiguity) and *controllability* (that controlling the state achieves the desired actions). These two properties tend to increase the

size of the state space by increasing either or both of its dimension (i.e. the number of independent state variables) or, to an extent, the density of significant values in each state variable. The dependence of performance on these structural factors was apparent during the development of the experimental simulations. These can be sensitive to variations in the specific choice of state variables. This is a consequence of the engineering trade-off between model complexity (in terms of the number of states required) and the fidelity of the 'represented' state to the 'actual' state; in other words, no approximated, discretised model can ever truly have full observability.

Assuming that a basis (i.e. a set of independent state variables) can be found which both adequately represents the behaviour and also offers useful control actions, then the system designer or engineer may begin to consider some form of modular decomposition of the state space in an attempt to abstract complex behaviour into a set of simpler components. In this, although this has not been explicitly explored in this PhD, one might follow the criteria proposed by Bertrand Meyer [47]:

- **Decomposability:** The structure must help the engineer split the overall design into simpler components, each of which is independent enough to be worked on separately.
- **Composability:** It is desirable that the engineer can re-use such components from previous designs, thereby reducing the cost of a new design and increasing the assurance for the individual components.
- **Understandability:** The engineer should be able to understand the operation and limitations of each component without excessive reference to any others.
- **Continuity:** A change to one component should have the least possible effect (at design time) on any other component.
- **Protection:** A fault condition in one component (during operation) should have the least possible effect on the operation of any other component.

At the same time, since this work has been concerned with systems which are amenable to mathematical analysis, this requires systematic techniques for decomposition which have a well-defined mathematical representation. A number of potential strategies and techniques for this have appeared, detailed analysis of which would be a useful future research direction. One example is the work of Puterman [56] to express the dynamics of the MDP in the notation and concepts of functional analysis, which raises the possibility that tools such as Fourier analysis might be applied to synthesise complex behaviours (policies) from a number of simpler ones –

in effect, the competing goals mechanism. The work of Barry et al. [6] mentioned above might also be relevant to further work in this area.

One particular challenge which may arise with such models is the problem of 'cyclic dependencies', or relationships between state variables which either prevent the separation of components, or create interfaces between them which contravene the criteria suggested by Meyer for well-structured systems. This area warrants further research because it has a direct bearing on the ability to realise effective models of complex systems.

9.2.4 Partially Observable Markov Decision Processes

The use of the Partially Observable Markov Decision Process (POMDP) was discussed in Chapters 5.6.8 and 6.5.3. This is distinguished from the MDP by the property that variation in the environment, which manifests as uncertainty in the system state, is explicitly incorporated into the design process. It should therefore be possible to model the expected statistical behaviour of the environment in far greater detail. This suggests follow-on work into the use of the POMDP as a technique for optimising the information flows required for collaborative systems. However, the lack of a general, tractable solution to the general form of the POMDP remains a major stumbling block.

9.2.5 The Topological Approach to Agent Communities

The topological approach to agent community formation, and to complex and continually evolving internal dependency structures and associations, needs to be consolidated around a mathematical model for the conceptual approach introduced in Chapter 5.8 of this thesis.

In particular, there is a need to:

- Define a structure for the topology for the system state.
- Define 'actions' which modify the topology.
- Interpret behaviour as the distribution of elements within that topology.

To formalise this model, the following assertions will need to be further evaluated and proven:

- Topological subcovers provide a basis for a hierarchical composition of nested MDPs.
- Then MDP decision actions can be represented as a topological transformation.

It may turn out that 'emergent behaviour' can be identified by its topological properties. As an example, it might transpire that the state trajectory of an element when analysed in isolation is *connected* but becomes *disconnected* when it interacts with others.

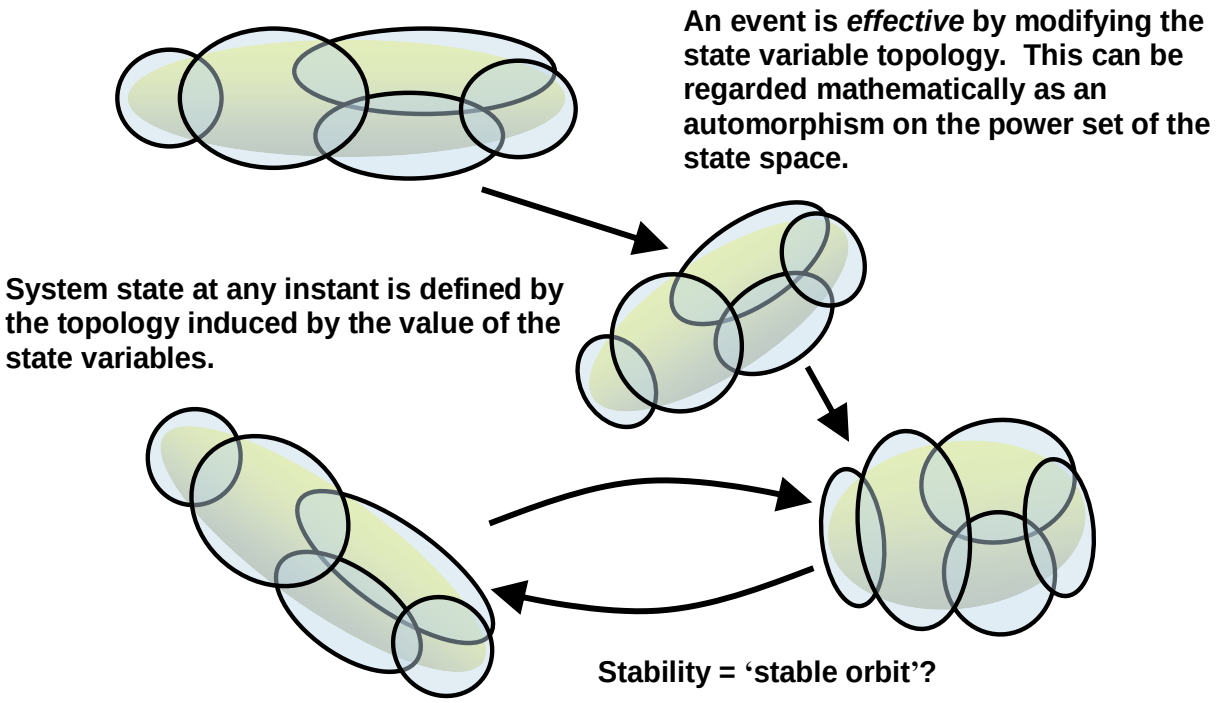

Figure 51 – Topological Interpretation

One way of looking at the action of events on the topology of the system state space is to view each one as an automorphism on the power set of the state space (because each topology is a member of the power set).

- This portrays the time evolution of the system as a set of 'group actions'.
- This allows an alternative representation of trajectories and orbits within the state space which might prove more tractable than trying to explicitly track individual elements.

Annex A

A. Description of Experimental Simulation Test Beds

A.1 Context

The techniques developed in the main body of this thesis need to be simulated in order to demonstrate that appropriate adaptive behaviour within a multi-agent system can be learned, and that the behaviour which is learned can be seen to include unplanned or unprompted adaptations in response to interactions whilst remaining bounded and with a speedy return to the originally intended pattern of behaviour.

MATLAB was chosen as a suitable tool for this purpose. It should be noted from the outset that the simulations developed are not claimed to be optimised or suitable for deployment, nor is the choice of MATLAB as the simulation tool a recommendation that it is the tool that should be used in future should techniques such as these be taken further or applied in real world applications. MATLAB was chosen because of its ease of use and the native support for graphical visualisations at run time, which were vital for extensive debugging and experimentation by the author, as well as for capturing snapshots of the simulations to illustrate the behaviours being induced (presented as figures in Chapter 8).

Two experimental simulation test beds were developed as part of this PhD, for the introductory and larger scale problems are described in Chapters 8.2.1 and 8.2.2 respectively. Whilst the introductory problem, and correspondingly the first simulation test bed, were useful for demonstrating the efficacy of the proposed techniques for enabling agents to learn bounds to their potential adaptive behaviour, it was clear that the problem (using polar coordinates and a simple orbital arc) was insufficiently complex to evaluate multiple forced interactions of greater numbers of agents in a visually clear and easily identifiable way. Expanding to the larger scale problem required a switch to cartesian lattices with alternative mechanics for

loading assignments and initiating agents within the simulation test bed. Although not the primary intention, which as with the first test bed was to provide an experimental platform for developing and demonstrating the learning techniques and bounded adaptive behaviours, the second test bed was more indicative in terms of logical structure as to how a real tool might be used, with the ability to define assignments in separate files and have these 'loaded' into the learning environment. The second version of the simulation test bed is therefore closer to being a prototype for a practical tool than the first, although producing such a prototype was not a primary objective of this PhD, and it certainly would need further work before it could be claimed to be a proper prototype.

Developing the second version of the simulation test bed also provided an opportunity to apply some lessons learned from the development and use of the first test bed to ensure that the second version was more in line with the natural logic of information flow within a multi-agent system without intra-agent communication (i.e. using global variables for environment parameters and persistent variables within the virtual agents for local state information and calculation). As a result, whilst there are many similarities between the two simulation test beds, there are some important differences in implementation.

In this Annex A to this thesis, an overview of the mechanics of the two simulation test beds is provided, including pseudocode for the major functions, as a guide for how to recreate or extend these approaches in the future.

A.2 Experimental Simulation Test Bed for the Introductory Problem

The first version of the simulation test bed was developed in order to simulate the learning process and the learned behaviours of a set of four virtual agents learning acceptable patterns of motion in the 'introductory problem' scenario as described in Chapter 8.2.1, as well as to calculate and record a number of performance metrics about the learning process for subsequent analysis. A logical architecture for the simulation test bed built for the introductory problem is provided in Figure 52, followed by summary descriptions of the main elements. It consists of a number of functional modules, differentiated as:

- Functions necessary to create the simulated environment in which the virtual agents live. This includes the executable script AgentTestEnvironment which initialises all

other variables and maintains the 'ground truth' of the state space lattice, the current positions of the agents, and all parametric information necessary to drive the simulation. Two additional functions called by AgentTestEnvironment calculate the transitions the various agents make once they have selected their next action, and make determinations about when there is a collision risk between agents. These are calculated every time increment, with current states updated and provided as inputs to the virtual agents, along with collision warnings and lists of restricted actions when appropriate (for action filtering by the agent).

- Functions necessary to simulate the agents themselves. The main function here is Agent, which contains most of the internal logic of an agent, receiving as inputs only environmental signals from AgentTestEnvironment (current state and time, and any warnings and restricted action lists when appropriate). Only one agent script is used and an agent number label maintained by AgentTestEnvironment (and provided as an additional input to Agent) is used to differentiate which instantiation is which. An additional function called by Agent is used to calculate rewards based on the observed outcome following from the state transition after the previous calling of that agent, once AgentTestEnvironment has updated all agents' states, which requires the recording of the last state and last action information for each agent internally using persistent variables. A PolicyComparison function is also included for the agent to incrementally update global variables monitoring how quickly the agent is learning what has been pre-calculated to be the optimal policy – data gathering at run time, rather than being a part of the agents' decision making processes. This function was used to collect the data to investigate convergence results such as those presented in Figure 46. [It should be noted that this approach, whilst effective through careful use of global variables, is not in keeping with the natural information flow in an agent-based system, and was one of the elements changed in the second version, with such calculations being more logically conducted within the test environment rather than within the agent.]
- Functions which provide tools for running the simulations. LoadAssignments is a user-editable script in which the coordinates and parameters of the required assignments can be entered and edited manually. Visualisation produces the graphical representations for debugging purposes and to demonstrate the behaviours that the agents are learning to follow (and was used to create Figure 44 and Figure 47), but can optionally be disabled in the master AgentTestEnvironment to significantly speed up the simulation.

StateIndexCalculator is simple tool used for expediency, to index all states tuples with an integer. AgentTestEnvironment calls this function to translate states that it has knowledge of into a unique integer identifier, which the agent needs in order to maintain internal state-action value arrays and operate a Q-learning algorithm. Finally, the function OptimalPolicyCalculator calculates from the outset what the optimal policy should be (based on the loaded assignment, which is simple enough in the first instance that the true optimal policy can be worked out analytically by the user beforehand). This is an analysis tool rather than part of the learning process, because learning would be trivial if the answer were known by the agent at the outset. Instead, this is calculated up front but is not shared with the agent, so that there is then a baseline against which to compare the incrementally improving control policies learned by the agents (using the PolicyComparison tool discussed previously).

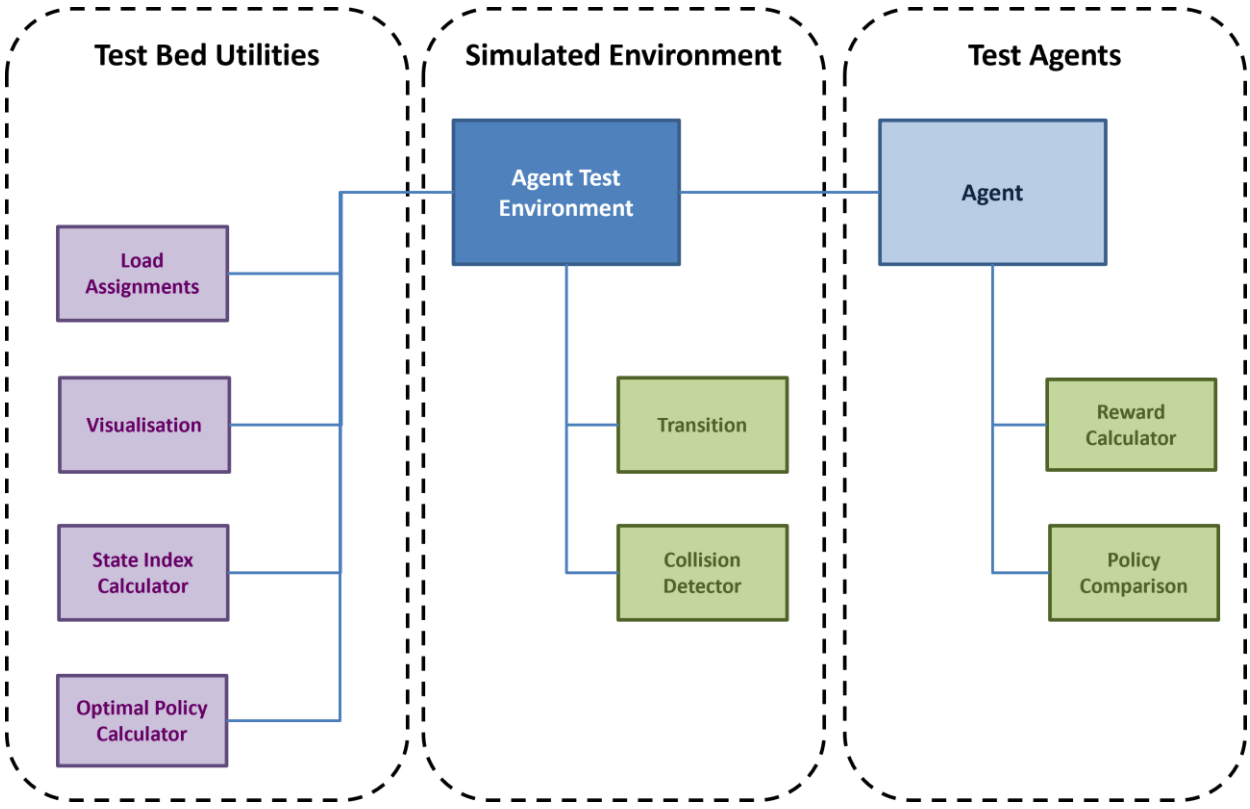

Figure 52 – Logical Architecture of the Experimental Simulation Test Bed for the Introductory Problem

There are two ways in which the users could interact with the test environment. One is by controlling the inputs which are read into the simulation at the outset, and the other is via the graphical display. The goal was that eventually these would be merged, so that the user would have the ability to place waypoints and times directly on a display, choose the coordinate system and episodic period, and configure any other useful parameters, and then watch the agents progress in real time. Beyond that, there should also be longer-term potential for task reconfiguration and agent adaptation (for example, amending assignments mid-period, or deliberately inserting obstacles to test control policy robustness). However, in the initial simple model the interface was not (and did not need to be) that advanced, and only by directly editing parameters within the scripts could the user influence the simulation. Developing this tool further was another of those paths this research could have taken, but ultimately was tangential to the main research objectives of this PhD (see Chapter 1.2.5). It could be pursued further in future work.

In the following subsections, each of these functions is described individually, and some pseudocode is provided for those that have non-trivial or algorithmic content relevant to the techniques described in this thesis.

A.2.1 AgentTestEnvironment

This is the master file or the main executable for the whole simulation test bed. This master file contains global state information (specifically the current location of each agent and the global system time) and therefore represents 'ground truth' for the whole model. In this sense it is simultaneously serving as both a simple simulated environment for the agents and as the simulation test bed controller.

Pseudocode for the AgentTestEnvironment function is as follows:

1. Initialise all the necessary global variables and arrays.
2. Call LoadAssignments to load the descriptions of the tasks the agents are required to complete and other necessary configuration information as specified by the user.
3. Call OptimalPolicyCalculator to calculate an optimal policy based on rules determined by the user, which has been pre-determined to define the desired pattern of behaviour.
4. Initialise the desired number of agents by calling Agent that many times in a loop using a different index for AgentNumber. For simplicity, the agent index number is the same

as the assignment number (so assignment one goes to agent one, with no loss of generality so long as all virtual agents function similarly).

5. Set initial states for each agent. [Currently, the user writes this directly into the script. A graphical interface for this was considered but time prevented it.]
6. Call the Visualisation script to initialise the graphical display if required (determined by a binary variable edited by the user in the main script amongst the global variables initialised at the start).
7. Commence a control loop for incremental learning. For t from 1 to user-specified T_{max}, then for each agent from 1 to n in turn:
    a. Call CollisionDetector. This requires the agent index and its current state, and checks whether that agent is in close proximity to any other agent at that point in time. It will return a binary warning flag, and an array of restricted actions (actions that would bring the agent closer to collision if undertaken next), or an empty array if there is no warning.
    b. Call each agent in turn (in practice, call Agent n times with a different index for each initialised agent) to select a next action. Agent requires as inputs its index (purely for internal cross-referencing to the appropriate Q-value array) as well as the current state of that agent, current time, and the warning flag and restricted actions list. The agent returns the selected next action. [For analytical purposes it also returns information about reward received, a pointer to the latest increment of its Q-value array in order to measure the rate of policy convergence during testing (noting that this does not play a part in the actual simulation), as well as debugging information. This is noted to be unrepresentative of a real implementation, and much of this calculation was conducted outside of Agent in the second version of this test bed.]
    c. Call Transition. This requires the agent index, its current state, and the action the agent has just reported, and determines what the next state will be.
    d. To facilitate the visualisations when active, and in any case for debugging purposes, record a string of the past actions, transitions and rewards as a new row in a path history array. [This array is not provided to or accessible by Agent and does not form part of the core simulation.]

e. When required, call Visualisation to update the graphical display in real time to reflect that agent's chosen action. The most recent entries of the path history array (ten prior entries in the figures presented in this thesis) are provided to Visualisation when active to facilitate this.

A.2.2 Transition

This function requires as input an agent index, and the current state and the just-selected action for that agent, which will have been selected and output by the Agent function immediately prior to the calling of this function. It identifies the coordinate system (polar in the original implementation, but later adapted to be able to select a polar or cartesian geometry) and using simple rules it calculates what state transition will occur given that input state and action. The output is simply the index of the next state that agent will be in. In the implemented models this is a deterministic function, always returning one specific next state depending on the input state and action. This function ultimately is a set of simple state transition rules (moves from one point in the coordinate space to another), plus logging functions for analytical purposes.

As well as potentially including some stochasticity to state transitions should one wish to model behaviour in an uncertain or dynamic environment, this function could also be extended to work in more complex geometries and/or in more dimensions, as the script includes a look-up from an extendable list of recognised coordinate systems (although enabling work would be needed to define the states and state transitions that would occur in new coordinate systems).

A.2.3 CollisionDetector

This function represents the agents' (limited) cognisance of their immediate surroundings. The problem definition supposes that the agents are able to detect the presence of any other objects within a specified (but small) radius from themselves – representing a simple collision avoidance detector or proximity sensor. It is not necessary for these purposes to actually simulate a sensor; a function that reports on any other agents within a specified radius of the current agent is emulating such a system.

Action filtering is the novel feature added to this implementation of Q-learners, where, in the event a nearby object is detected, any actions that might bring them closer together is

temporarily excluded from the list of available actions. This is changing the mathematical model of the Q-learning problem (removing part of the policy matrix, requiring renormalisation over the remaining permissible actions) and may result in policy changes (where what would have been the optimal policy action is now removed from the action space).

To enable this, this function when called requires the index of the relevant agent and access to the global state variables. Simple rules written into the script are then applied to calculate whether any of the other agents' current states are within the specified radius. If there are any, the variable WarningFlag is set to 1 and further calculations based on simple rules written into the script are then applied to calculate the direction of the nearby agents, and thereby the actions which, if selected next, would move the current agent closer to the nearby agents. These actions are then enumerated and collated into a vector of RestrictedActions, and this and WarningFlag are the outputs. If there are no other agents within the specified radius, WarningFlag is set to 0 and this plus a null vector are the outputs.

A.2.4 LoadAssignments

This is a simple read-in function that specifies a number of assignments that the agents are expected to undertake, consisting of waypoints, times, and reference geometry (currently only 2D polar and 2D Cartesian, but designed to allow for expansion). For simplicity, one assignment per agent is intended. This function has no inputs or outputs. It simply accesses a global variable initiated by AgentTestEnvironment and writes the corresponding assignments into this state variable, as part of the initialisation of the simulation. The reason for establishing this as a separate function was to enable this to ultimately be replaced by a more complex function or even a graphical interface to manually input assignments. However, such directions were tangential to the primary research objectives for this PhD and this is left as an objective for any future work, with the simple reading in of assignments manually typed into this script being adequate for present purposes.

A.2.5 Visualisation

Currently this function is a passive graphics generator, with no user control enabled. To enable the graphics to run in real time a global background is generated on initialisation, as are a large number of image handles for the variable elements being represented. How much of the agents'

path history is to be represented is a variable the user currently has to set manually (choosing to show ten time steps back from the path history in the figures presented earlier in this thesis). Thereafter the function is called within the main control loop within AgentTestEnvironment for each agent after each action selection and state transition, leading to the erasure of the last most image handles and addition of the latest ones. The effect is to produce a real time simulation of the agents moving around the simulated environment, and also featuring the 'target' point for where the agents should be at that point in time if they were following the optimal policy without interruption or perturbation. Visualisation has access to all global variables held within AgentTestEnvironment, including PathHistory that is compiled incrementally each time step by recording all state, action, transition and other data reported by Agent each time it is called.

Pseudocode for this function is not provided partly because it is not relevant to the functioning of the core Q-learning algorithm, and partly because it is a lengthy series of image handle updates and native plotting tools, which would result in a long section of pseudocode that was not at all illuminating as to how the techniques proposed in this thesis actually function.

A.2.6 StateIndexCalculator

State is a (currently two dimensional) coordinate and a current time (within the episodic period). These have to be enumerated by discrete integers to enable Q-learning, which requires a discrete number of enumerated states. This script is therefore a coordinate transformation, to turn coordinates into discrete state numbers, representing the agent's internal logic for where it is at and what its situation is. In the original implementation this was limited to polar coordinates, but an updated version of the function has as an additional input variable the state space system being applied, and has within it the ability to apply polar or cartesian coordinate transformations as required.

A nested function called InverseStateIndexCalculator does the reverse, transforming enumerated states back into physical coordinates (using whichever coordinate system has been specified amongst the inputs) for debugging purposes.

A.2.7 OptimalPolicyCalculator

This function is called once by AgentTestEnvironment immediately following the calling of LoadAssignments and the loading of the user's specified assignments to calculate what an optimal solution to the given problem would be. This function would not exist in reality – after all there is no point having agents learning the solution to a complex problem if it were possible to explicitly calculate a solution *a priori*. This function is included within this experimental simulation activity in order to determine whether the learning agents are indeed converging to optimal solutions. For model fidelity, clearly the output of this function (the solution which the agents are trying to find) cannot be made available to the agents and is held in the main memory of the AgentTestEnvironment for later comparison (as well as to allow Visualisation to plot the target position when active).

This function works by accessing the global variable Assignments that contains all the assignments that have just been read in by LoadAssignments. One of the reasons for using simple orbits in the introductory problem (and regular polygons thereafter) is because these can be defined by waypoints and required times to reach each waypoint. It can be easily determined how many time steps will pass in the time it is expected to take to traverse a particular edge or arc length, and so expected positions at each intermediate time can be interpolated using simple rules written into this script. Once this is known, at each time point in the episodic period the location the agent should have reached is known, and the optimal actions to take can be easily calculated and collated into an optimal policy array, which is returned as the output from this function for future comparative analysis.

It is acknowledged that this simple calculation of an optimal policy is a deliberate consequence of the use of simple coordinate geometries and assignments that involve tracing out regular shapes, which is unrealistic in reality. This has been done deliberately so that the true optimal policy can be derived mathematically without needing any learning – but then learning it anyway, to prove that the agents are learning, and to gather evidence and metrics about their learning. This is also beneficial later because, knowing what the uninterrupted optimal policy should have been and being able to present this visually, makes it considerably easier to identify when complex emergent behaviours are subsequently being observed (i.e. recurring, orderly but limited deviations from this ideal pattern of behaviour followed by a speedy return and catch up).

A.2.8 Agent

This function is the Q-learning agent that is the entity being modelled, tested and assessed. It is called from AgentTestEnvironment for instantiation, and thereafter repeatedly called in a control loop, and the inputs received from the test environment and outputs returned to the test environment are the interactions of primary interest.

In practice, one Agent function is repeatedly called for, simulating each of several agents. This is making an implicit assumption that each agent within a multi-agent system would have similar control logic (in this case, each one being a Q-learner). It would be possible to have multiple functions for Agent 1, Agent 2, and so on, but if one is assuming that all agents have the same control logic then this is adding complexity and multiple copies of substantially the same script for no obvious benefit. An alternative approach which has been adopted here is to have a single function covering all agents, with all variables calculated and recorded within the agent being indexed by which 'version' of the agent (or which virtual agent) is being called each time. This is achieved by requiring an agent index number as an input each time the Agent function is called. The first time it is called with index 1, it initialises the agent with a number of persistent variables all indexed by 1, then when it is called again with index 2 it initialises a matching set of persistent variables all indexed by 2, and so on. Then on all subsequent calls, the agent only uses those persistent variables corresponding to the current agent index, and thereby the whole multi-agent system has its decision making all contained within one function. [If there were a need to simulate agents with fundamentally different internal logic (i.e. not just using different parameters, but using different algorithms entirely) then this approach could still be followed by using conditional logic to determine which portions of code to apply during which instantiation, but it might be easier in practice to separate into multiple Agent functions in such cases.]

Most information a Q-learning agent needs is internal to the agent, specifically the current Q-value table (initialised as zero arrays) and a variety of fixed parameters. These include learning rate α and discount rate γ in the terminology used in Chapter 5.6, which are both written into the script as 0.5 in the initial implementation, but which could be varied. This also includes a state space exploration rate which determines (in the learning phase) how often the system should take the current highest value action and how often it should take a lower value one, to ensure state space coverage and to determine through sampling what the true values are. This was written into the script as 0.75 in the simulation runs presented in this thesis, but could also

be varied. [It should be noted that different values could have been adopted for different agents, indexed using the agent index input described above, but this has not been explored in the work performed to date where the same parameters were used for all virtual agents to enable clear comparison of outcomes. Applying different learning strategies amongst the agents might be an interesting direction for further work, but was an unnecessary complication at this initial proof-of-concept stage.]

A Q-learner will have a number of internal variables, chiefly its policy and the Q-value array that places values on different combinations of states and actions. In the current model, these are large arrays held in persistent memory so as to have a separate pair of arrays for each agent. It is how these arrays change from time step to time step that determines how the agent learns and how policies converge.

Knowing the last action the agent took and its previous state, then the reward scheme (which is internal to the agent) will output the reward corresponding to the agent's last move. Given a defined set of actions, a current Q-value array, and all the internal parameters having been set, the currently observed state is the only additional piece of information needed to apply the Q-learning update formula (as defined in Chapter 5.6.6):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

In this implementation, with the novel feature of action filtering, some additional inputs are also needed. Binary variable WarningFlag and an array RestrictedActions (as determined by CollisionDetector) are also required as inputs to this version of Agent.

The only output from a normal Q-learner is the selected action, as what consequence that choice has is not down to the agent (it is calculated and updated within AgentTestEnvironment and its subfunction Transition, and is affected by the choices which the other virtual agents make). For debugging purposes, as well as to monitor the rate of policy convergence, the Agent function also outputs a number of what would normally remain its internal variables – what its current policy is, but also how it is determining reward each time step, what its current Q-values are, and some metrics describing overall state space coverage and state revisitation rates during learning. These outputs play no part in the Q-learning process nor in updating the global state space by AgentTestEnvironment, but they are outputs to that function which are then collated into PathHistory, used in visualisations and in post-learning data analysis.

Finally, one feature to note of the initial implementation of this test bed is that, at specified times and on completion, PolicyComparison can be called by Agent to compare the agent's

most recent internal action selection policy (i.e. its internal Q-value array) against the pre-calculated optimal policy solution (stored within AgentTestEnvironment as a global variable) to determine the rate of convergence to this solution (by comparing all elements of the array, and counting the number that match the optimal solution). Although the global variable containing the true optimal policy is not used during the Q-learning, it was recognised after implementation that this was a curious way of setting up the PolicyComparison, to have it called from within the Agent function, when in reality an agent would not have access to global 'truth' such as the pre-calculated optimal policy. This was removed from the second version of the test bed, but for completeness it is correct to document here that this is the way this was originally implemented in the first version.

Pseudocode for the Agent function is as follows:

1. Initialise the necessary persistent variables and arrays.
    a. On first call, initialise all variables and create a subarray indexed with the first agent index number.
    b. On subsequent calls, create further subarrays indexed with each subsequent agent index number, until all variables exist with parallel subarrays for each virtual agent.
    c. [Note: Q-values hereafter form a structure with a set of n subarrays, one for each agent from 1 to n; at this stage, all entries are zero. The dimensions for each Q-value subarray are the number of states (as enumerated by StateIndexCalculator) multiplied by the number of possible actions (which is fixed), and for convenience these can be read by Agent from the global variable SimParameters within AgentTestEnvironment, as a one-off accessing of global information just to facilitate initialisation.]
2. Set previous state and previous action for each agent to be empty initially.
3. Initialise Policy (highest value action per state). This has been implemented using an internal function called PolicyCalculator. In the initial implementation, for each state in the state space in turn, PolicyCalculator first calls StateIndexCalculator to enumerate each state, then uses simple rules to exclude certain actions from states identified as boundary states (i.e. those on the edge of the grid cannot have a subsequent action that would take the agent beyond the edge of the grid). Then a simple argmax function

identifies the highest value in that column of the Q-value array, and the index of that value is the number of the highest value action, which is therefore saved in Policy as the action to take when in that state. [Note: Action filtering is applied at a later stage in this initial version.]

Then for all subsequent agent calls, using only the relevant subarrays indexed by the agent index number included amongst the input arguments each time:

4. If input variable WarningFlag is 1, filter the action space by taking the input variable RestrictedActions, and creating a new array PermittedActions whose entries are all the actions whose indices are not contained within RestrictedActions. [Note: This is the implementation of the action filtering step.]
5. Extract previous state and previous actions for the relevant agent from persistent variables.
6. Call RewardCalculator to calculate the appropriate reward given the current state (amongst the input arguments) and the past states and actions (from the persistent data accessed in the preceding step).
7. Update the relevant Q-value subarray according to the usual Q-learning formula. In pseudocode terms, this means for the recorded previous state and previous action, the entry in the Q-value array corresponding to that state and action 'OldQ' is replaced by 'NewQ' = 'OldQ' + 'LearningRate'*('Reward' + ('DiscountRate'*'MaxQ') – 'OldQ'), where the rates are stored within persistent variables and 'MaxQ' is simply extracted from the current Q-value array as being the maximum value in the column of the array corresponding to the agent's current state. Note that the set of available actions is limited to those actions whose indices appear in PermittedActions if WarningFlag is 1.
8. Optionally, recalculate policy by calling the internal function PolicyCalculator again at this point. [It is not necessary to do this every time during the learning phase, but it may be useful for debugging or for learning rate analysis. A marker variable can be used to determine how frequently this is undertaken.]
9. Each time policy is recalculated, call PolicyComparison to calculate how closely the agent's current policy matches the pre-calculated optimal policy held in a global variable by AgentTestEnvironment but not otherwise used by this function. How closely the current and optimal policies match is then recorded outside of the simulation test bed (and is used to create results like Figure 46).

10. Calculate next action.
    a. During the learning phase, firstly create a random number.
    b. If this random number is less than the fixed parameter ExplorationRate, randomly select one of the actions (limited to randomly selecting one of the PermittedActions when WarningFlag is 1).
    c. If this random number is greater than the fixed parameter ExplorationRate, follow current policy (either calculated in the preceding step, or from the most recently saved version in the persistent variables if not), and given the agent's current state select the action whose index is the same as the index of the highest value element in the column of the Q-value array corresponding to current state.
    d. If the learning phase has completed and the test bed is set to continue simulating the learned policy for illustrative or analytical purposes, then follow the last calculated policy, as per above.
11. Save current state and the action calculated in the previous step as the persistent variables previous state and previous action (because they will be the previous state and action by the next time that Agent is called with this same agent index number).
12. Output the identified action. Also output reward, Q-values, and other 'internal' agent parameters for debugging purposes or for analysis, even though these do not need to be outputs for the purposes of Q-learning.

Also included within Agent but left out of the above pseudocode are some state space coverage metrics, to measure how comprehensively the agent has covered the state space and repeatedly sampled each state. This takes the form of a 'heat map' of state visitations. A reset marker is included amongst the state variables if it is beneficial to relocate the agent back to somewhere on the intended arc, so that the agent does not waste learning time resampling outlying low-value regions of the state space. This is an internal fix to speed up learning time, and can be deactivated if one is not concerned about lengthening learning times and prefers the simpler, purist approach of allowing sufficiently many random walks for a long enough period of time to ensure state space coverage and adequate sampling.

Coverage metrics and learning reset markers are part of the mechanics of the simulation and do not play any part in the actual Q-learning process, other than a reset marker equalling 1 resulting in Agent not conducting a Q-learning increment on that occasion (because the

observed transition would make no physical sense and does not correspond to anything the agent did or could have done that would make sense within its model, and as such any reward or Q-value update – if they could even be defined – would be misleading).

A.2.9 RewardCalculator

An agent only knows what state it was in and what action it took last time, and what state it is in now. It needs to receive a reward in order to learn from it experience whether the previous action selection was a good choice from that state or not. Due to the criticality and potential complexity of this function, it was implemented as a separate function from the rest of the Agent function. In the initial simulation test bed, the approach that was selected was to issue a simple reward of +1 if the agent was on track and on time or ahead of where it should be, a reward of 0 if the agent was on track but behind where it should be time-wise, and a reward of -1 if it was not on track at all. RewardCalculator therefore has as its input variables the agent index number, the agent's current state, and the current time. It then loads the assignment corresponding to that agent from the global variable Assignments within the AgentTestEnvironment function. Then the nearest point on the arc that the agent is supposed to be following relative to the current state is calculated. Current time is converted to episodic time (i.e. how far through the given episode is the agent, using modulo arithmetic). Based on the relevant entry in Assignments, it is then a simple calculation whether, given episodic time, the agent is on time or ahead of schedule, or behind schedule, and the correct reward is then output accordingly. Some data, including phase differences, are calculated and stored as persistent variables, with the idea being to enable increasing or decreasing rewards on subsequent steps depending on whether the agent is making up any lost time or falling further behind; however there was not time to implement this, and in the results presented in this thesis rewards followed the fixed reward scheme described here. Whether a more complex system as envisaged would speed up learning time is something that could be investigated in future.

It should also be noted that alternative reward schemes could be applied, either written into this script in place of the current reward scheme, or alternatively replacing this function with an alternative. This was a further motivation for separating this function from Agent, because the logic inside Agent is unlikely to change from implementation to implementation, whereas the reward mechanism delivered by this function is likely to change from implementation to implementation, which is easier to manage if instantiated as a separate function.

A.2.10 PolicyComparison

This function is used to compare the latest or final policies calculated by the agents against the optimal policy calculated *a priori* by OptimalPolicyCalculator. This is purely for the purpose of determining how well the agents are performing and does not represent part of the agents' actual deliberations. As noted previously, it is acknowledged that having this function called from Agent is not representative of how a real agent would work. It was implemented in this manner because policy iteration occurred inside the Agent function and was not, as originally implemented, available to any other function, so inserting the function call for PolicyComparison inside Agent was a practical solution. An alternative and more logical approach was adopted for the second version of simulation test bed (by making use of global and persistent variables structures to make data notionally internal to Agent like policy readable to other functions), to remove any perception of illogicality from future versions of the simulation test bed.

A.3 Experimental Simulation Test Bed for the Larger Scale Problem

As noted previously, the rationale for rebuilding the test bed was that to effectively present a larger number of agents undertaking a larger number of concurrent overlapping assignments, a different geometry and a more diverse range of tasks was required than four agents tracing out the same orbit, effective though that first scenario was for proof-of-concept. This prompted a change to a cartesian coordinate lattice, and the use of rectangular arcs that deliberately overlapped to ensure some agents would come into contact with each other as they learned.

As a result, a second version of the simulation test bed was developed in order to simulate the learning process and the learned behaviours of a larger set of virtual agents learning acceptable patterns of motion in the 'larger scale problem' scenario as described in Chapter 8.2.2, as well as to calculate and record a number of performance metrics about the learning process for subsequent analysis. A logical architecture for the simulation test bed built for the larger scale problem is provided in Figure 53.

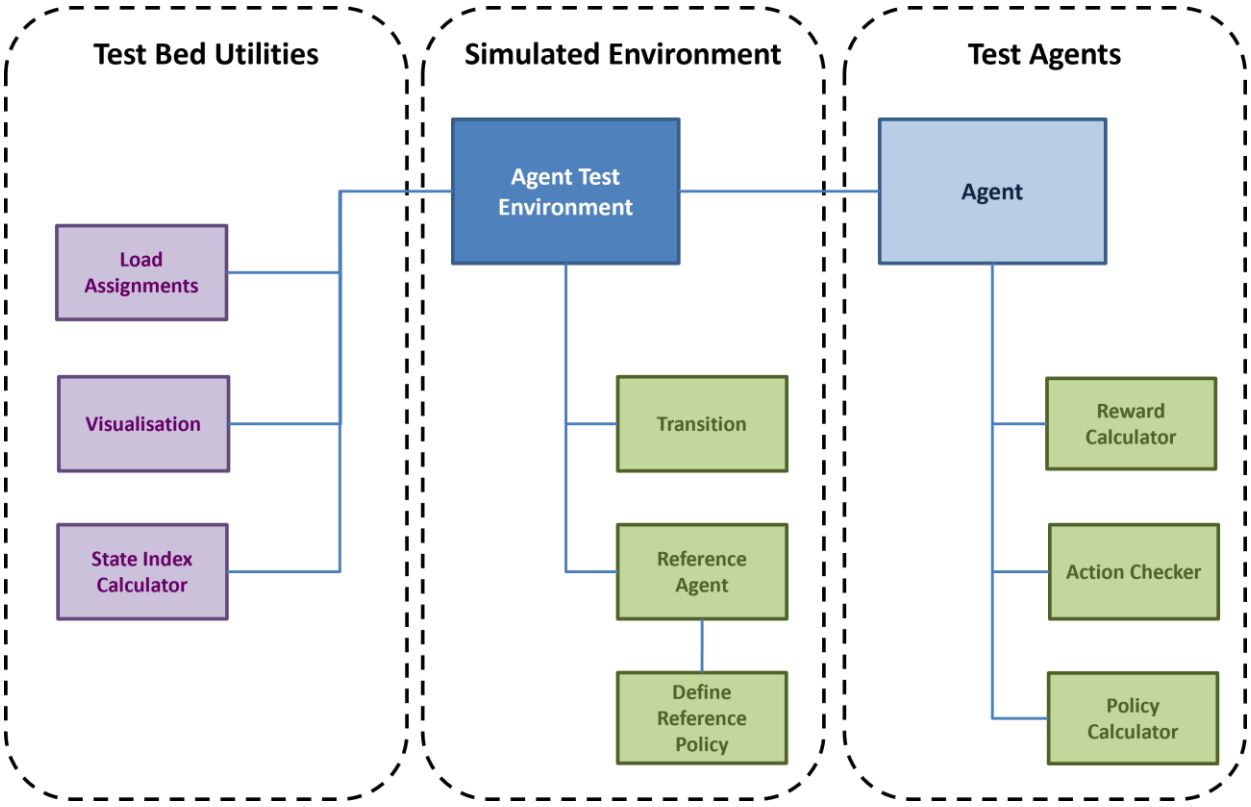

Figure 53 – Logical Architecture of the Experimental Simulation Test Bed for the Larger Scale Problem

By comparison to Figure 52, it is clear that the two test beds have many similarities. The second test bed is an evolution of the first, with some common functions and other functions that are extensions or enhancements of earlier functions. However several of the functions needed to be replaced to facilitate this larger scale model. This presented the opportunity to revisit and improve a number of features within the first simulation test bed based on lessons learned.

It is not proposed to describe the second test bed in as much detail as the first, given that many functions are the same or very similar. The following describes the key differences from the first version, and should be read in conjunction with the detailed descriptions of the functions in the earlier version in the previous section:

- The CollisionDetector function was removed from the simulated environment part of the test bed and replaced by ActionChecker inside the simulated agents part of the test bed, as a function now called directly by Agent. This is more representative of realistic information flows within an agent-based system, with the ActionChecker function serving as a simple emulation of that agent's proximity alert sensor. ActionChecker

requires an additional input variable of the relevant agent's index number as well as the agent's current state, but it has access to global state information and can report back to Agent an AvailableActionList. Binary warning flags and a restricted action list are no longer required; by calling this function from within Agent and getting straight to AvailableActionList, the agent is able to modify its own actions list and filter its own Q-value arrays automatically, removing the need for a number of additional sorting and sifting phases from Agent and other functions. This is a more representative approach and a more convenient approach in terms of reducing the complexity of code in other functions. The functionality inside ActionChecker is the same as was inside CollisionDetector, based on simple rules written into the script to determine whether any next actions would lead to a state that another agent is already in.

- The PolicyComparison function is removed completely, and replaced with a simplified version of the Agent function called ReferenceAgent, which is called from AgentTestEnvironment. This change removed perhaps the most glaring concern from the first version, which was giving the agent notional access to the true optimal policy to calculate for itself how well it was doing. Although for justifiable analytical purposes and carefully implemented so as not to corrupt the Q-learning process, this was nonetheless illogical in terms of information flows. The alternative approach is to maintain a second set of (much simpler) virtual 'reference agents' based on a slimmed down version of the Agent function. These reference agents are initialised at the same time as the Q-learning agents, corresponding one-to-one. At initialisation, they are loaded with the pre-calculated optimal policies, and only follow these throughout. At each time step and for each agent, AgentTestEnvironment will first call ReferenceAgent and obtain a next action to inform the reference agent's next step, and then call Agent and obtain a next action to inform the virtual agent's next step. ReferenceAgent has all the Q-learning functionality removed and always follows the true optimal policy, whilst Agent is a Q-learner as before and operates similarly to that in the first version of the test bed. AgentTestEnvironment maintains both, and is able to produce all the data for comparative analysis (such as the average mean error plots contained in Figure 49 and Figure 50) and to support visualisations (where the path of the reference agent is by construction exactly the same as the 'target point', denoted in Figure 45 by the red dots following the prescribed arcs). Introducing reference agents transforms the simulation into one where the Q-learning agent is essentially learning to

follow or ‘keep up with’ the reference agent, and policy convergence can then be metricated by the mean square difference between reference and virtual agents – how well does the Q-learner mimic the reference agent, and when it is forced to deviate, how effectively does it catch up with and resume mimicking the reference agent.

- As a consequence of the above change, OptimalPolicyCalculator as a test bed tool is replaced by DefineReferencePolicy as a function called by ReferenceAgent. Each time ReferenceAgent is called to initialise a new reference agent associated with a particular assignment, one of ReferenceAgent’s first steps is to call DefineReferencePolicy for each newly initialised reference agent in order to generate and load a control policy into the relevant indexed subarray within ReferenceAgent, after which this policy is the one followed by ReferenceAgent throughout the simulation. The functionality inside DefineReferencePolicy is the same as was inside OptimalPolicyCalculator in the first version of the test bed, still based on user-specified pre-determined rules but operating in this case on the new type of assignments based on cartesian coordinate logic.
- PolicyCalculator is separated from the main script of Agent to become a separate script, called by Agent when required. This was to support future generality. The basic operation and internal logic of the function is the same as in the first test bed.
- LoadAssignments, StateIndexCalculator. RewardCalculator and Transition have all been extended to reflect new geometries. All still operate on defined rules written into the scripts, but now feature the ability to recognise the coordinate system being used in a particular implementation and apply one set of rules or the other. They have already been described in the previous sections.
- Changes have been made to data structures throughout the model, principally concerning the use of global and persistent variables and ensuring the information flows are more representative of real agent-based system information flows. More simulation parameters useful for analysis and debugging are held within global variables in AgentTestEnvironment, reducing the number of inputs and outputs being passed between functions, and improving the comprehensibility of the functions.

The AgentTestEnvironment and Agent functions had many consequential amendments as a result of these changes, primarily to variables names, inputs and outputs, and where key parameters are obtained from given the changes to data structures. These follow from the changes described above and are not listed individually.

In addition, the AgentTestEnvironment function has also been amended to record the key PathHistory data and a record of policies (both Q-learned and reference policies) to facilitate offline data analysis and plotting after the simulation has concluded. None of these amendments change the basic logic of this function. The pseudocode presented in Chapter A.2.1 is essentially the same for the second version, other than:

- Step 3 (calling OptimalPolicyCalculator) and Step 7a in the control loop (repeatedly calling CollisionDetector) are omitted.
- Step 4 (calling Agent n times with a different index number to initialise n virtual agents) now needs to be read as n pairs of calls to Agent and to ReferenceAgent, to initialise n corresponding pairs of virtual and reference agents.
- Within the Step 7 control loop, each time Agent is called to identify its chosen next action, the corresponding ReferenceAgent should also be called (to move the 'target point' along to its next state as well).
- An additional step is added at the end, to conduct the external data recording.

The second version of the Agent function also has some amendments. The pseudocode presented in Chapter A.2.8 for the Agent function is essentially the same for the second version, other than:

- In Step 3 and Step 8, using PolicyCalculator is now an external call to a separate function.
- Step 4 is replaced. WarningFlag and RestrictedActions are no longer provided inputs. Instead, at this stage in the control loop, ActionChecker is called and produces AvailableActions, and the action space and corresponding value arrays in the rest of the control loop are then filtered based on this. All references to actions in the Agent function's pseudocode after this point should instead be read as references to AvailableActions when considering the second implementation.
- Step 9 (PolicyComparison) is omitted. This is now assessed offline based on data recorded by AgentTestEnvironment.

The key Q-learning stages (steps 5 to 11, other than step 9) are the same as in the earlier version, other than for using AvailableActions and filtered action arrays throughout the loop rather than filtering in real time. The policy update rules are the same as described in the earlier pseudocode.

Finally, pseudocode for the ReferenceAgent function is a slimmed down version of the pseudocode for the Agent function, removing all the steps related to Q-learning and inserting an additional call to DefineReferencePolicy on the initiation of each reference agent.

With reference to the Agent function's pseudocode, the ReferenceAgent function's pseudocode consists only of:

- Step 1 (including additional initialisation calls to DefineReferencePolicy for each agent).
- Step 3 (which is reduced to be a look-up from the calculated reference policy).
- Step 10 (only allowing the option of following a defined policy, with no learning mode or exploration rate).
- Step 12 (only outputting the selected action is required, but outputting any other parameters for debugging purposes is optional).

All other steps from the Agent function's pseudocode are not a part of ReferenceAgent's pseudocode.

A.4 Areas for Future Improvement

Through the development and use of two versions of the experimental simulation test bed a number of ideas for future improvement or enhancement have been identified, and for both completeness and posterity these are summarised below:

- To support future generality, allowing AgentTestEnvironment to be initialised with a user-specified coordinate set up from a range of options (polar and cartesian, with the potential to introduce others) rather than writing this directly into functions' scripts.
- Introducing 3D coordinate systems and assignments, more representative of the physical movements required in real examples such as production line automata.
- Using graphical interfaces tools (such as pointers and image handles) to allow user-specification of many assignments and parameters via Visualisation rather than through editing of the scripts. Potentially this could also facilitate real-time adaptation, either through changing assignments, or through introducing additional assignments or obstacles, in order to test robustness.

- Introducing ‘pop up’ obstructions, to test the agents’ adaptability and robustness. This would have been possible within the existing versions by directly editing the functions’ scripts, but was not pursued as it would have been a distraction from the primary research objectives given that the problem had already been designed so that the agents would cause obstructions to one another.
- Exploring not-1-to-1 mappings of assignments to agents was considered. This was not pursued because it is not clear how realistic it would be to implement a real system in this manner; however this may be an interesting avenue for theoretical investigation.
- Exploring dynamic re-allocation of assignments to agents (i.e. job swapping). Alternatively, exploring the addition of new assignments and agents partway through the other agents’ operation (i.e. forced replanning). Either of these could be implemented through a relatively simple extension to the current functions. This was not prioritised as these scenarios are unlikely to arise in a system with a stable configuration and fixed roles for its parts like a set of interacting production line automata, which was the primary exemplar in this work. It may have more relevance to other applications, particularly assignment allocations within a fleet of autonomous vehicles, which was discussed in the main body of this thesis.
- Improving the visualisation functions to highlight a permissible operating envelope for each agent (for example, a shaded or coloured band either side of the prescribed arc). This would communicate effectively that there was permitted flexibility for the agent when it needed to adapt, and the requirement was to demonstrate that the agent could demonstrate suitably bounded adaptation. With the visualisation function as it is, this would make the figures produced cluttered and unreadable and so was omitted, but this would be a feature worth revisiting in future. This is expected to be a particularly useful and informative feature to add to future test beds if simulating more complex problems that may feature different shapes and sizes of permissible operating regions for different agents, or which go further and feature dynamic permissible operating regions changing during run-time, which would be beyond the current test beds’ capability to illustrate.
- Real time plotting of performance metrics such as policy convergence (similar to Figure 46) and mean position error (similar to Figure 49 and Figure 50) would be useful for debugging activity, as well as indicating how such a simulation might be employed in reality were it to be further developed into a practical development tool.

There are two final observations to make regarding future directions for the test beds:

- Developing real engineering tools was not a research objective for this PhD. The test beds developed should not be considered to be prototypes for future tools. Nonetheless, it is hoped that they provide a first look at what future tools for these purposes might look like, in the same way that the theoretical developments in the main body of the thesis are intended to provide underpinnings for what might in future become a recognised approach to system design and proving (in conjunction with further developments to the topics discussed in Annex C). Transforming the test beds developed in this PhD into real tools would be a large development task rather than a research activity, and it is not necessarily the case that one would continue to use MATLAB for this purpose. Whether or if alternative coding languages or software development environments should be applied would be one of the first questions that would need to be answered. There would also need to be further design work considering how user interfaces would work, what kind of results and metrics would need to be presented to whom, and also (outside of the tool itself but of key importance to its design) how such a tool would fit into a recognised systems engineering processes, with accompanying documentation. All of these elements would be necessary future work to bring the vision behind this PhD to fruition.
- The assurance part of the overall systems design and proving challenge (introduced in Chapter 4.4 and discussed further in Annex C) is discussed in this thesis as important context, but was not part of the key research objectives. As such, the simulation test beds have not been designed to support experimentation or visualisation of any such tools or techniques from this area of proposed work. This would be an obvious area for further development of the test beds, in conjunction with further development on the theoretical underpinnings. Ideas such as illustrating the permissible operating bound for each agent as proposed above is a step in this direction, as would be more complex implementations such as dynamic permissible operating regions, seen to be changing during run-time via the test beds. If other techniques are identified in future as part of a more comprehensive solution to the assurance part of this problem domain (from computational geometry, fractal bounding methods, network discovery algorithms arising from the discussion of topological underpinnings in Chapters 5.8.2 and 5.8.3, or otherwise) then it would be valuable to explore whether these could be integrated into a common test bed, with the aim of providing an end-to-end solution.

Annex B

B. Models for Alternative Applications

B.1 Context

Before the industrial exemplar aligned with Warwick Manufacturing Group's research interests in automotive manufacture and digital lifecycle management was identified as an ideal case study for this PhD, two interesting application areas had already been identified as candidates on which to base a case study. Some preliminary work was undertaken considering these applications areas, how models of multi-agent systems might be constructed as representations of these scenarios, and the complex emergent behaviours anticipated from them.

Although these models became superfluous once the decision was made to concentrate on the production line automata case study, this annex presents the work that was conducted towards building up these alternative case studies. There are two reasons for this. Firstly, so the work can be seen and perhaps motivate further development of these alternative applications in future work, either by the author or by another reader of this thesis in the future. Secondly, and perhaps more importantly, because nothing demonstrates the applicability of a system modelling approach to multiple applications better than having been seen to be applied to multiple applications. The development to date of the logistics management and sensor coordination case studies is just that – evidence that, at least in principle, the mathematics developed and the modelling approach trialled within the synchronised automata problem domain in the main body of this thesis can be applied far more broadly, and this annex gives an indication as to how.

B.2 The Logistics Management Problem

Any large logistical deployment provides a perfect example of a large multi-agent system – for example, disaster response, humanitarian relief, or an emergency services or military

deployment. Each division within the deployment has a requirement for supplies, including food, fuel, equipment for the staff, and many other items, and there are obviously practical limitations that affect who can have what quantity of which resource provided when, immediately setting up the potential for internal disputes over how best to allocate and distribute resources. The challenge for the logistician is to make decisions about how best to support the deployment based on incomplete data and potentially inaccurate assumptions about what the current and foreseeable needs of the whole task force really are.

The basic problem is that of the transportation problem, in which there are known supply bases, customer demands and costs for each route from supply base to customer, with the objective to minimise time and cost while meeting all demands within the evolving operation. Classical techniques have sought to look for optimal solutions using algorithms, simulation and modelling, queuing, and stochastic methods to optimise or improve the real-world problem. But in reality optimality is only applicable to a static problem, and the logistics problem is dynamic and contextual, with requirements constantly changing, and as such appears to be an ideal problem to tackle using autonomous agents.

Considering the logistic supply bases, distribution resources and end users as independent entities, it is possible to envisage the system operating an internal market, in which case the need of the individual to maximise their gain may not be to the overall advantage of the whole deployment if individuals are working with an incomplete picture of the relative priorities. A balance between the individual's gain and the overall gain would seem to be a more appropriate approach. The question then becomes how one might apply a complex state machine approach to suitably manage the behaviours of the entities within this model. Presented here is the work-to-date on a candidate model for this task based on the principles developed in this thesis.

B.2.1 The Agent Perspective

It is assumed that:

- There is a finite number of 'types of resource' which need to be distributed.
- A finite number of courier agents handle the distribution, each of which can carry a number of 'units of resource' up to some specified maximum capacity for that courier.
- Neither the total unit capacity nor the breakdown of different types of units that can be carried needs to be similar across different couriers.

- The world in which they exist is represented by a network of two types of nodes, supply depots (sources) and distribution points (sinks).
- Between discrete time steps, courier agents can either stay at the node they are at or move along one edge of the network to a neighbouring node.
- At a source, courier agents gain units of resource, up to their maximum capacity.
- At a sink, courier agents deliver units of resource to the node, up to a maximum of whatever the courier is carrying.

The state of a courier agent at each time step can be written as $\underline{s} = \left(s_1, s_2, \ldots, s_n, \underline{p}\right)$, where $s_i =$ number of units of resource i carried and $\underline{p} =$ current position in the network, for which there are alternative labels, the logical one for the example given being grid coordinates, so $\underline{p} = (p_1, p_2)$ where $p_1, p_2 \in \{1,2,\ldots,5\}$, and so $\underline{s} = (s_1, s_2, \ldots, s_n, p_1, p_2)$.

Each distribution point has a requirement for some number of units of each type of resource, and the goal for the system is to fulfil as many of these requirements as possible as expediently as possible. The actions that a courier agent therefore has to take are a series of moves and deliveries. Assuming they cannot instantaneously jump to their chosen node, possible actions can be defined as being a change in coordinate of one step in whichever direction is chosen (if permissible). For example, a move to the right in the example model above would correspond to a state transition of $p_2 \mapsto p_2' : i \to i + 1$, or $\underline{s}' = \underline{s} + (0,\ldots,0,0,1) = \left(s_1, \ldots, s_n, p_1, (p_2 + 1)\right)$, and the other possible moves can be defined similarly.

Alternatively, a courier agent could choose to make a delivery, which corresponds to no move taking place but instead a transfer of some number of units of resource. For example, a delivery of three units of resource 1 and one unit of resource 2 causes the state transition $\underline{s}' = \underline{s} + (-3, -1,0,\ldots,0,0,0)$.

Note that when visiting a supply node and taking on extra resource, the state transition is just like making a delivery, only the quantities are non-negative and one can assume the decision as to how much is taken on is at least in part outside of the courier agent's control (and in any event is constrained by the quantity of resources available at the supply node). A simple option for initial consideration is to consider the supply process as being a separate decision process external to the courier (and so part of the courier's environment).

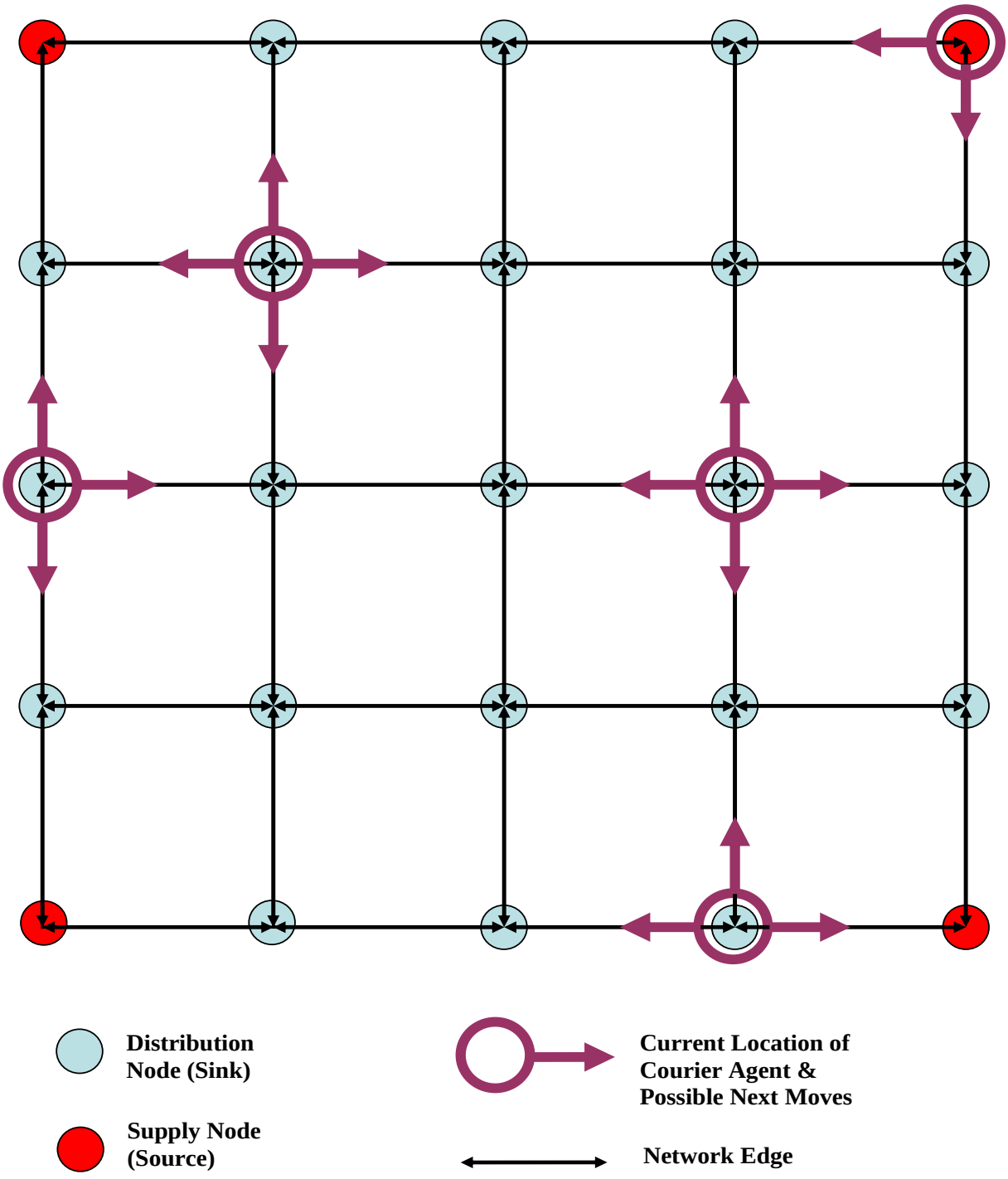

Figure 54 – Logistics Model (Agent Perspective)

Based on the template described above, a simple reward scheme might for a courier agent to receive +1 for every unit resource they deliver. One may wish to include temporal discounting, to reward timely delivery (which would also require orders to not just be wish lists but requirements with a specified due date), in which case reward schemes such as +1 for every

unit delivered on or before the requested due date (which would presumably be defined as one of the time steps), and the ratio (due date divided by delivery date) for every unit delivered thereafter.

There are many other factors that could be incorporated into the model, such as costs for travelling along particular network edges (or even incorporating a dynamic network, where the network edges and the costs associated with travelling along them can change, or even where the number of nodes can change). One could also introduce costs for travelling with too great a load, prompting a decision about what the optimal load to carry will be in order to maintain flexibility to meet requirements without unnecessarily burdening oneself. Other additions to the model can also be envisaged.

B.2.2 The System Perspective

Considering the logistical operation within a large organisation hierarchically, assume that the system as a whole consists of a central management hub, local (possibly mobile) bases, and distribution assets belonging to those bases. Operational requirements flow from the central management down the hierarchy, until placed on an individual at the right point in the hierarchy to fulfil it. In practice, organisational arrangements could be more complex than this, with more levels of decision making, maybe different lines of authority for different assignments, perhaps with different practices and processes.

However, due to the dynamic nature of operations and the prospect of 'live updates' to operational requirements, this task assignment process will generally require at least some degree of flexibility and adaptability. It is for this reason that much of decision making authority has to be devolved away from the central management, leaving them the role of providing the overall governance and guidance, as the 'owner' of all resources. They have to consider not necessarily the particulars of the requirements themselves, but the overall capacity of the system to fulfil them. The state information needed to represent this must therefore describe as succinctly as possible how the current resource distribution through the network matches up to the currently understood requirements on the overall system.

It is assumed that:

- The system can acquire new resource as it needs them (not realistic in practice, but the focus of this model is on the distribution problem, not the procurement problem).

- New resource will appear at a source node and is represented as the state of the node $\underline{\tilde{s}} = (s_1, s_2, \ldots, s_n)$, defined similarly as for the agents (only without the notion of a position).
- The courier agents could either have full, partial, or no knowledge of the resource levels at each source node. This would be the designer's choice. (They will of course need knowledge of at least some of the requirements of the sink nodes, but not necessarily all.)
- A system could be defined where the courier agents have the ability to reallocate resources already delivered to other nodes in response to subsequent higher priority requirements, which will require measures of resource availability at each node and the inclusion of each of these measures in the following definitions in the logical way. It will also require the extension of the action model for the individual agents to include the collection of resource from sink nodes (positive delivery values).

The total capacity of the systems is $\underline{S} = \left(\sum_{\underline{s},\underline{\tilde{s}}} s_1, \sum_{\underline{s},\underline{\tilde{s}}} s_2, \ldots, \sum_{\underline{s},\underline{\tilde{s}}} s_n\right)$, or the *n*-vector giving the sum total of all resource in the system's courier and supply nodes.

The total requirement on the system is $\underline{R} = \left(\sum_{\underline{r}} r_1, \sum_{\underline{r}} r_2, \ldots, \sum_{\underline{r}} r_n\right)$, where $r_i = (r_1, r_2, \ldots, r_n)_i$ is the quantity of each of the n commodities required at node i.

Choosing to not include the procurement problem into this model means that when $\sum_{\underline{s},\underline{\tilde{s}}} s_i < \sum_{\underline{r}} r_i$ for some commodity i, the difference in commodity level has to be instantaneously delivered to the source nodes (whether this should be all at one node or separated over several is up to the designer). This is entirely idealised procurement and supply, but adequate for a model where the focus of interest is on how the courier agents respond.

In its simplest form, the actions on the system are essentially trivial. A requirement for commodity i at node j is communicated to all couriers currently carrying this commodity, amending their goal profile. More complicated models are possible however, for example passing the requirement (amending the goal profile, and potentially changing the community structure) to all couriers carrying commodity i within some given number of steps from node j, or to all courier agents, irrespective of their current possession of any of commodity i, that are within some other given number of steps from a source node. This would be an interesting scenario for future experimentation.

Rewards for the system could be as simple as those described in the initial basic template, namely -2 for each unfulfilled requirements and -1 for each 'exhausted' courier agents at each time step. Again, some experimental 'tweaking' will no doubt be useful, but a simple initial option is to define, at each time step, $\hat{R} = \sum_{i,j} \hat{r}_{ij} + \sum_{i,k} \tilde{r}_{ik}$, where $\hat{r}_{ij} = -2$ when $r_i > 0$ at node j and $\hat{r}_{ij} = 0$ otherwise, and $\tilde{r}_{ik} = -1$ when $s_i = 0$ for courier agent k and 0 otherwise.

The next step with this model will be to produce a basic implementation with which to experiment, tweak as appropriate, and generally explore whether, as is believed theoretically, this set-up will produce observable – but understandable – complex emergent behaviours, and how one might use this understanding to respond to it.

B.3 The Sensor Coordination Problem

Modern sensors are complex systems, and to aid human users they increasingly feature greater levels of automatic processing built into them. As technology advances, one should anticipate in the future having the ability to deploy networks of sensors that will be able to work together autonomously, with not just automated processing but also significantly reduced or even eliminated need for human involvement in the deployment and operation of the network. Security operations such as policing, protection of critical national infrastructure or security of events or public areas are the obvious examples, but alternative applications in ground or oceanic survey, disaster recovery and military applications can also be envisaged.

Networked sensor management has a number of challenges that makes that it a domain ideally suited to autonomous agents:

- Optimal sensor deployment and coverage is context dependent and may change with the evolving environmental conditions.
- The computational cost of central management of this type of system is exponential to the number of sensors.
- The system elements are not co-located and the management system must manage computational and information resources that are physically distributed.
- Sensor networks may be dynamic systems that are affected both by the internal status of the sensors and by the effects of the environment.

- The system may be resource limited due to the individual sensor power and communications bandwidth available at any point in time.
- In managing the network, it may be necessary to coordinate different sensors (and types of sensor) in order to achieve tasks that could not be achieved by individual sensors.

In a given context, one can identify a set of potential decisions that will need to be supported, and thus the information required from the deployed sensors that will be needed to support the making of these decisions. Sensor management could mean managing a set of sensors with respect to their capability, where one may wish for them to be deployed in order to achieve certain coverage, or to enable the sensors to extract particular information in order to best support the decision making process.

In any such deployment, there will be a number of sensors each with a number of attributes such as their capabilities, position, and health. For each sensor, the central 'management' could provide for it a set of prioritised objectives against which the individual sensors acting as agents could seek to provide the maximum return, and the effectiveness of such an approach could be measured using measurements such as coverage and accuracy.

It is not necessary for there to be a networked sensor management problem that the sensors in the network be mobile or deployed, as there will be a degree of local control required for reorientation or refocusing of the sensor even for mounted sensors that remain in the same location (in addition to having digital settings to readjust as necessary). However many of the most promising application areas, for example in survey work or in crisis and disaster management, will depend on mobile sensors fitted to mobile platforms such as UAVs. Therefore for an initial approach to tackling this problem domain, a network of mobile deployed sensors was considered, which have to manoeuvre themselves and set their sensing fields of view to provide the best overall coverage required to achieve their assigned objectives.

As a starting point, assume a mobile sensor is maintaining coverage over a particular area, when a new requirement is received that requires it to take a detailed look at one specific location just on the edge of its current field of view. This will require an adaption of some kind, either by a) reorienting the sensor in its current position, or b) moving the sensor to obtain a better look angle closer to the target location (the dynamics and control mechanisms for movement of the deployment platform being a separate issue neglected here). To assist with visualising the problem, graphical representations are provided by Figure 55 illustrating the reorientation solution and by Figure 56 illustrating the relocation solution.

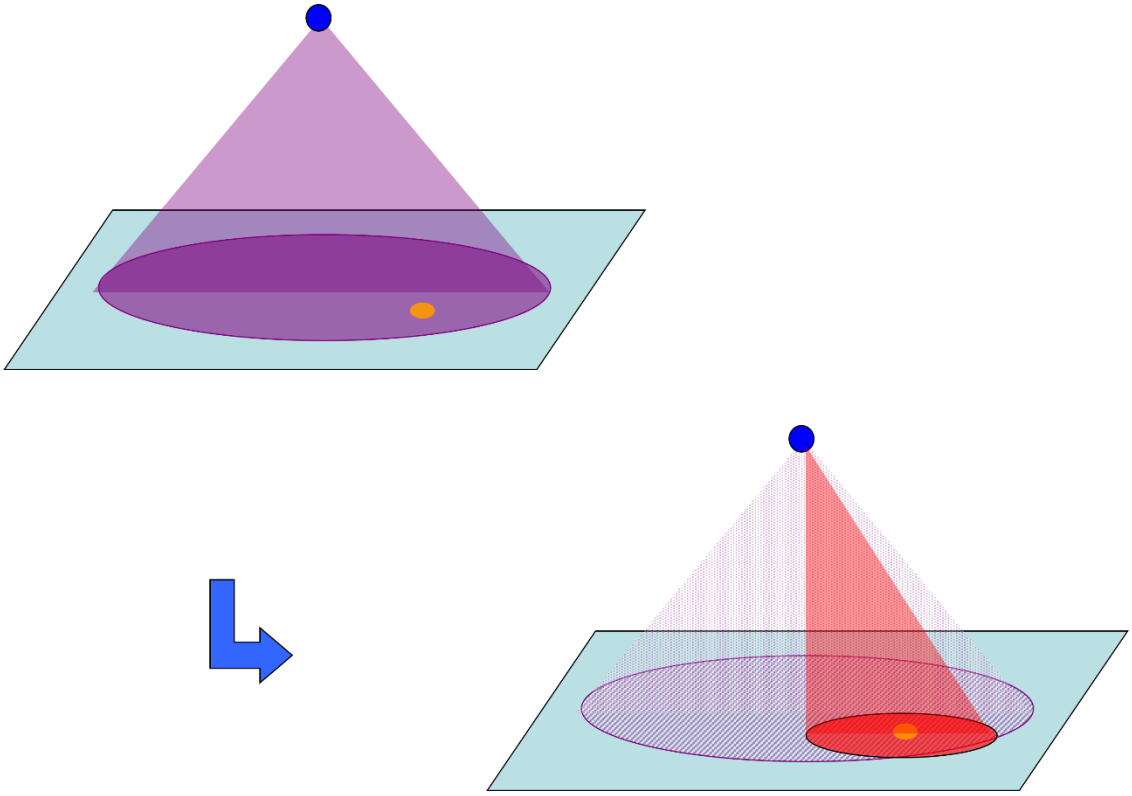

Figure 55 – Reorientation of Mobile Deployed Sensor

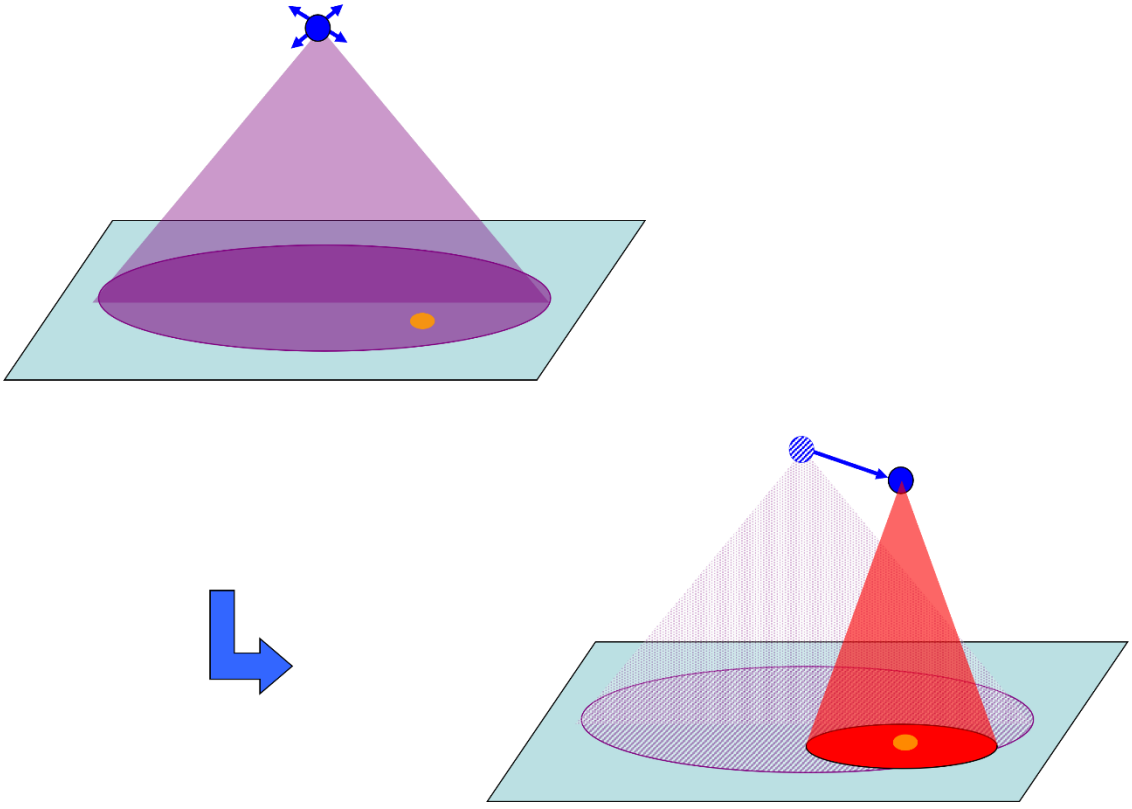

Figure 56 – Relocation of Mobile Deployed Sensor

Which solution to take, or whether to take a hybrid solution (for example, moving part of the distance but also partly reorienting), is a detailed matter for the sensor experts and operational design specialists. The key point for this discussion is that there are choices to be made with competing objectives to balance. Reorient the sensor and it remains in the ideal position to resume overall surveillance at any time; however this will not result in the close up view of the particular target of interest that might be required. Relocate the sensor and this may provide the best close up view of the target, however the deployment platform is now out of place, which may compromise the ability of that sensor to maintain overall coverage of the area as it used to and/or introducing a delay before the deployment platform could return to where it was.

Generalising to a network of deployed sensors, consider the same situation where one deployed sensor is assigned the same task (to take a detailed look at one specific location just on the edge of its current field of view). In the figures below, this is the bottom right of four deployed sensors illustrated. The choices for that deployed sensor system are the same, reorient or relocate. What is different is that, as part of a multi-agent system, other entities in the network might be able to adapt their own positions or fields of view to compensate for the impact of the reassigned sensor's choice (either through collaborative planning or reactive adaptation).

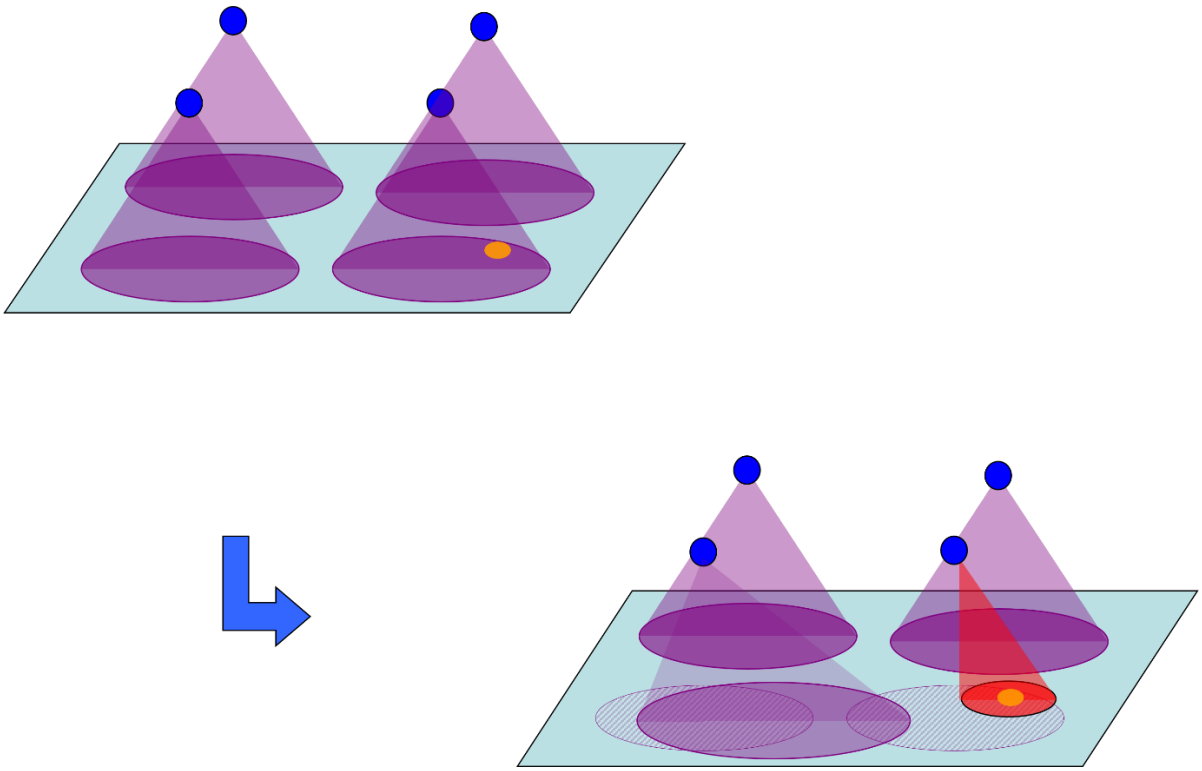

Figure 57 – Reorientation of One Sensor Prompting Reorientation of Peer Sensor

For example, Figure 57 above illustrates the situation if the reassigned sensor reorients itself to focus on the target of interest, and a neighbouring peer then reorients its field of view as well in order to maintain some level of coverage of the area formerly monitored by the first sensor. The overall coverage may not be as comprehensive as it was initially, but that is an inevitable consequence of the choice to reassign one of the assets in the sensor network. However, the overall coverage given compensatory network adaptation should be better than had such compensatory adaptation not occurred (strictly speaking, the decrease in coverage is less that it might otherwise have been).

For completeness, Figure 58 illustrates a similar potential compensatory mechanism where the reassigned sensor chooses to relocate in order to focus on the target of interest, and the neighbouring peer also relocates slightly in order to cover some of the first sensor's original coverage area. If one were to look only at the movement of the deployment platforms, situations such as this would seem like two autonomous agents with some dependency causing a correlation in their movements (perhaps a hidden dependency depending on the type of information communicated between entities and between levels within the system).

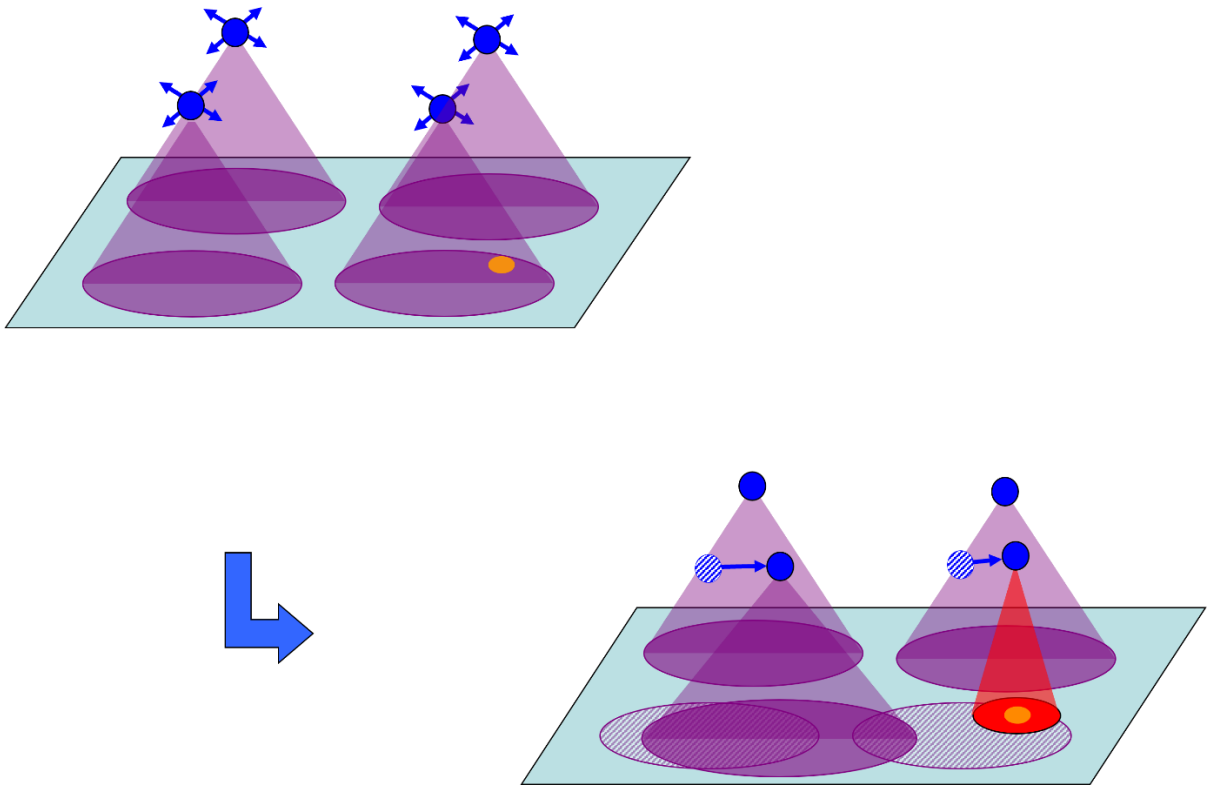

Figure 58 – Relocation of One Sensor Prompting Relocation of Peer Sensor

Note that it is not prescribed that reorientation should prompt reorientation, and relocation should prompt relocation. It may be better in a given context for the reassigned system to relocate but the neighbouring system to reorient itself to compensate, or vice versa. Any compensatory action could involve multiple peers – for example, the two nearest peers could relocate or reorient themselves in an attempt to cover the first sensor's coverage area, whilst the fourth may make a smaller reorientation or relocation to backfill any areas that those previous two sensors have now left with reduced coverage. Much as the first sensor could choose a hybrid solution of partly relocating and partly reorienting, so could any of the other systems choose hybrid solutions as part of any compensatory response. The key point is that relatively simple choices could lead to complex ripple effects across the sensor network.

B.3.1 The Agent Perspective

Suppose that the objective is to maintain maximum coverage of an area of interest, whilst maintaining the operational flexibility to be able to redeploy assets to focus on particular locations or objects within the domain. Responsiveness to local developments motivates the devolution of decision making capability to local agents.

For a simple model, consider the following simple model:

- A number of autonomous sensor agents are trying to maintain coverage of a specified area of interest. They can adjust their coverage area to improve their coverage of particular features or objects of interest.
- It is assumed that there are a number of features or objects of interest, which the sensors will need to adapt to improve their coverage of.
- Complexity may arise from the sensors making their own decisions as to how to adapt to cover the features of interest which may not correspond to the best global solution for maintaining coverage of the whole area of interest.

Perhaps the easiest way to visualise the problem, and certainly the perspective on the problem prompted by the representation developed herein, is when adapting to focus on a particular feature means a geographical movement in order to follow another (perhaps moving) object. Figure 59 provides an illustration of this perspective. However, similar logic would apply to a system that is moving in the sense of diverting its own resources to use a different type of sensor at the expense of another, or reorienting itself from one direction to another.

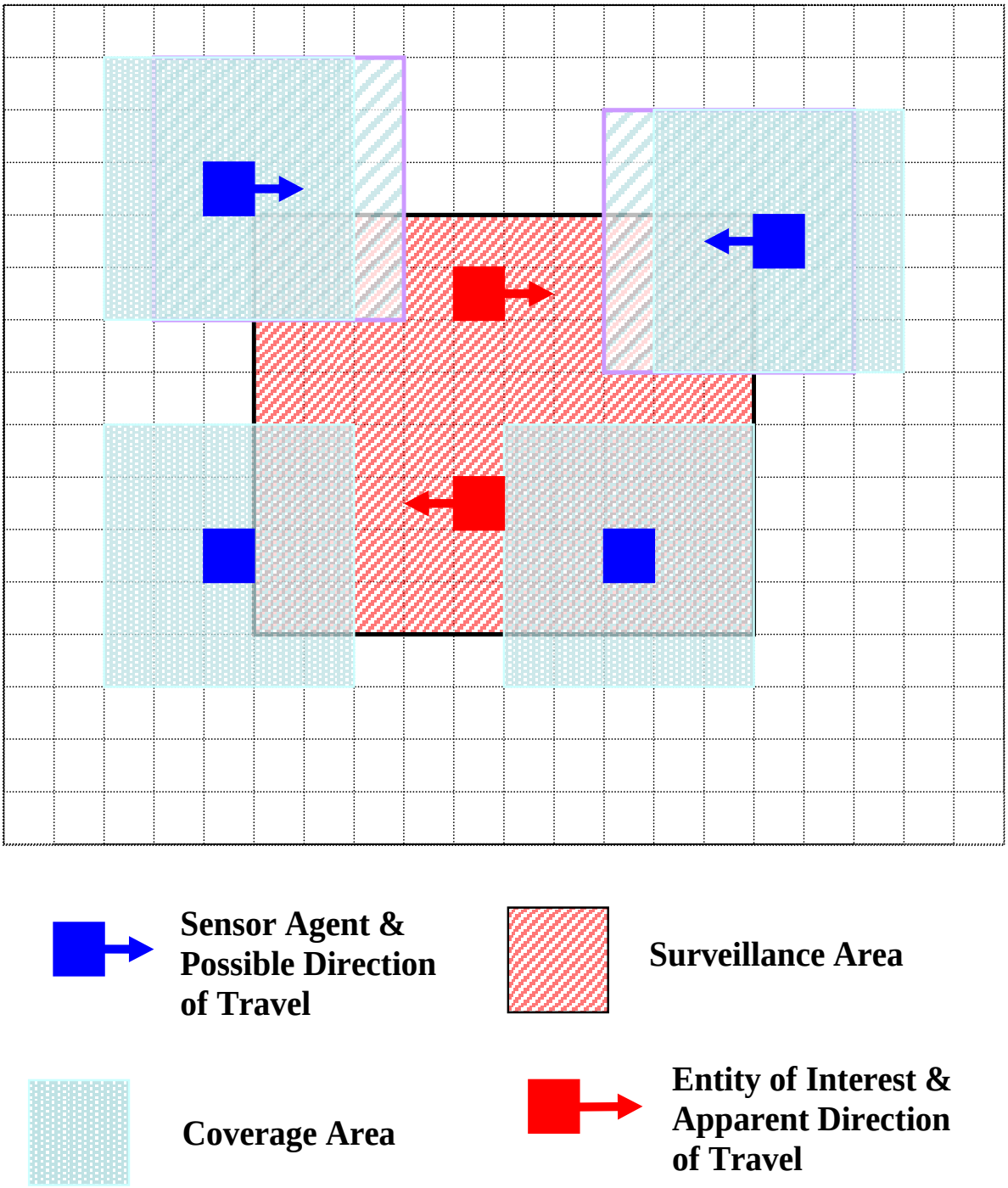

Figure 59 – Intelligent Mobile Sensor Network

To model this system, one needs to define what is meant by the state of the agent, what are its goals, and what are the actions and rewards which determine the agents' behaviour. The state of a sensor agent can be described by its position with respect to the intended surveillance area, and its position with respect to the object of interest.

Additional complexity would result from describing the agent's position with respect to multiple objects of interest, a possible solution for which is to have the system focus only on the nearest, or nearest not-otherwise-monitored, object of interest. This should make the model more tractable but at the cost of making it even less likely that the overall system will be able to identify a globally optimal solution.

In this simple case, a grid coordinate system attends to the portion of the state information relating to the individual agent's position, and a candidate for the latter would be a 2-vector describing, admittedly coarsely, the approximate distance and bearing of the proposed target from the current position (or multiple such 2-vectors in the multiple targets case). This gives a state of the form $\underline{s} = (x, y, r, \theta)$ for suitably defined coordinates, say $x, y \in \{0, \pm 1, \pm 2, \ldots \pm n\}$ for the local grid coordinates and $r_i \in \{1,2, \ldots, 2n\}$ and $\theta_i \in \{0,45,90, \ldots, 315\}$ for the approximate range and bearing of the i^{th} target from the current position.

The action in this model is just a move, represented as a vector addition to the current grid coordinates. In this situation however, the state transition is stochastic, because the state of the target is outside of the control of the sensor agent, and it will not be known ahead of making the decision what the range and bearing of the target will be.

The goal is to maintain coverage of both the whole area and of particular targets, which are not necessarily complementary goals. A reward scheme that might reflect this situation, working on a cellular lattice model as illustrated above, would be to give a sensor agent a reward of +1 for each cell under surveillance that is in the target area and -1 for each cell under surveillance that is outside this area, in theory incentivising remaining within the target area. Additional reward of +2 could be gained whenever a target of interest is under surveillance, incentivising actions that achieve this and thereby creating the possibility of conflict when this leads to an overlap with another sensor agent or movement outside of the target area.

There are of course many alternative ways of weighting the rewards that could influence the resulting behaviour; whether the optimal policy that results from this particular suggestion for a reward scheme produces desirable behaviour from the sensor agents is something to be investigated through implementation. In practice, parameter-tuning should be anticipated.

B.3.2 The System Perspective

In the autonomous systems context, it is implied that there is at least some degree of human-free decision making in the overall coordination and allocation of sensor assets. The system perspective in this case means the software monitoring or coordinating the network, depending on the level of control required of the system in question.

The state of the whole system must describe the current coverage of the target area. This is trivial to formulate in this case; the number or proportion of monitored cells will suffice.

The actions taken by the whole system could be of the form of an overrule of a particular sensor agent, which would be formulated the same way as for the agents.

One could impose a reward of -1 for any cell under surveillance within the target area, and -2 for any cell outside the target area, which is under surveillance by more than one sensor agent, to incentivise efficient use of resource. A reward of -2 could also be imposed whenever any target of interest is not being monitored by any agents.

Note that to implement this, some indexing of which cells are monitored by which agents will be necessary. Also assumed throughout is that the sensor agents have knowledge (or can learn) where targets of interest are and where the boundaries of the total area of interest are. In operational circumstances where either of these things can change, more interesting behaviours can be expected to result.

Annex C

C. Bounded State Spaces and Safe Operating Regions

C.1 Novel Approaches to Safety Assurance

Safety assurance involves analysing a system for weaknesses inherent in the design. To certify a system as safe, a systems safety engineer needs to gather evidence for a safety case, which is defined by Bishop and Bloomfield [12] as:

> *"A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment."*

Numerous techniques exist in conventional safety case analysis, including Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA), which examine what can go wrong with the system's operation. FTA identifies a failure that could occur, then shows which components need to fail to cause this failure, then which subsystems need to fail to cause that, and so on until a tree of causal links exists which captures the likelihood of that failure occurring. FMEA takes an alternative approach of considering the system as a block diagram of system components, and tabulating the anticipated consequences associated with each block in the system failing. All such methodologies are based on the fundamental principle of the *predictability* of the system's behaviour under the circumstances the system is designed for and expected to operate in.

Such methodologies can give high degrees of assurance that systems will operate as intended. However, the systems in question are typically closed systems in the sense that the system design constrains to some extent how the components interact with each other. The phrase 'to some extent' reflects the fact that when systems fail it could well be because of consequences implicit in the design of the system that were simply not anticipated – for example, the critical resonant frequency of wobbly bridges was inherent in the design, but was not anticipated prior

to some of the infamous bridge collapses. The same could potentially be true of any engineered system. However, one would in general expect well-designed and thoroughly tested systems to behave essentially predictably and to allow the safety engineer to determine to an extremely high degree of accuracy what the possible types of system failure are and what circumstances will bring them about.

In the world of autonomous systems, including agent-based systems, the situation is less straightforward. Notwithstanding the potential complexity of the systems themselves, the applications one might envisage in difficult and dynamic environments will typically feature significant external or environmental influence on systems. Furthermore, if operations are foreseen with multiple autonomous systems interacting with one another, or with changing mission plans, variable weather or hostile or disruptive action, then the operation of an autonomous system cannot be said to be closed (or even necessarily predictable) in the sense that one can control or reasonably anticipate virtually everything that can affect the system. This would be especially true for a system permitted to adapt to such problems as it encountered them. Traditional safety assurance methodologies may struggle with inherently unpredictable and potentially chaotic operating environments anyway, but the notion of autonomous systems making decisions under their own authority and adapting to situations they find themselves in would seem to be incompatible with the notion of predictable behaviour. With safety certification typically predicated by predictable behaviour, it is not obvious what could contribute to a formal safety case for a system intended to operate autonomously in such environments.

The premise underpinning this thesis was that safety assurance could be undertaken with an assessment of whether or not the actions an autonomous system is allowed to take will result in bounded behaviour; whether the control policy will result in the system remaining within some bounded, ‘safe’ region of the state space. Essentially, the idea is to remove the explicit need for absolute predictability of what will occur and replace it with a mathematically sound process for ensuring the overall behaviour of the system will not be unsafe. If such assurance can be given, then autonomous systems could be deployed with at least some degree of adaptability for operations in dynamic environments, and the designer has some assurance that the overall system behaviour should not stray into operational states that are considered unsafe. The following subsections consider how this problem might be tackled.

Note that throughout this work, ‘safe’ is loosely defined as those operating parameters the system is designed, permitted or desired to remain within – precisely what is meant by ‘safe’

will depend entirely on the system in question and the operational context, and a formal definition of safety will not be given.

C.1.1 Applying Novel Approaches to Safety Assurance

The key concept to outline in this final annex is the proposal that geometric techniques for safety certification could be applied in practice. The analytical approach would be to say that, given some initial starting point in the state space, a series of control actions defined by some policy produces an orbit through the state space. The (typically continuous) trajectory through the state space that corresponds to this orbit will have to be assessed against whatever safety criteria have been specified (note that even when the state space one is using in a control model is a discretisation, in reality the whole of the continuous path the system takes will likely have to remain within the state space and not just the discrete end points).

It would be helpful if the behaviour of the system can be shown to converge to a steady and predictable pattern; for example, in the experimental example developed as part of this PhD, if there were a solid body of evidence that agents have learned to trace out their required arcs, with any perturbation to avoid collisions being within an acceptable tolerance. Analytical solutions might be useful if the steady and predictable patterns were an attractor of a dynamical system, but not all problems will meet this criterion. Notwithstanding the problem of classifying whether an attractor (if it exists) is in a safe or unsafe region (and what to do if straddles both), a potential problem foreseen with this approach is that the action functions may not always be contractions (meaning in this context actions of sufficient impact to cause perturbations in patterns of behaviour to grow more pronounced). Although assuming action functions are contractions is not unreasonable in traditional control problems (since one would not typically allow actions that are likely to lead to unstable behaviour) this might not continue to hold when adaptive behaviours are allowed – indeed, temporarily allowing state transitions of greater magnitude might allow a system to move away from an unanticipated local disturbance and might actually be safer in the long run. However, allowing for such adaptation could lead to reachable state spaces expanding over time, and so potentially not converging to any bounded subset, safe or otherwise, complicating the application of techniques based on convergence to attractors even when such an attract can be defined.

In effect, allowing autonomous systems to adapt to unpredictable environments (using the terminology from Chapter 4.5.1, moving from an approach (b) to an approach (c) level of

control) potentially complicates the use of traditional analytical approaches. As an alternative, as part of the precursor work, Williams [68] considered the application of Iterated Function Systems (IFSs), further extended by Deeks (the author of this thesis) and Williams [23]. An original motivation for this PhD was for the author to continue to develop this research, into IFSs or a wider range of candidate solutions from computational geometry, but as noted earlier in this thesis this aspect of the research was parked. What follows is an update of the earlier work from [23] with as much revision and extension as the author had undertaken by that point. However, the novel proposal, which is for a geometric rather than dynamical systems approach to forming a 'novel safety assurance' process, has not changed.

The following flowchart in Figure 60 describes the main stages in the kind of safety assurance process envisaged. The IFS techniques described herein for a traditional system (or amended versions of these techniques for adaptive systems) could then serve as a subsequent verification stage for the assurance and certification of the finalised optimal control policy.

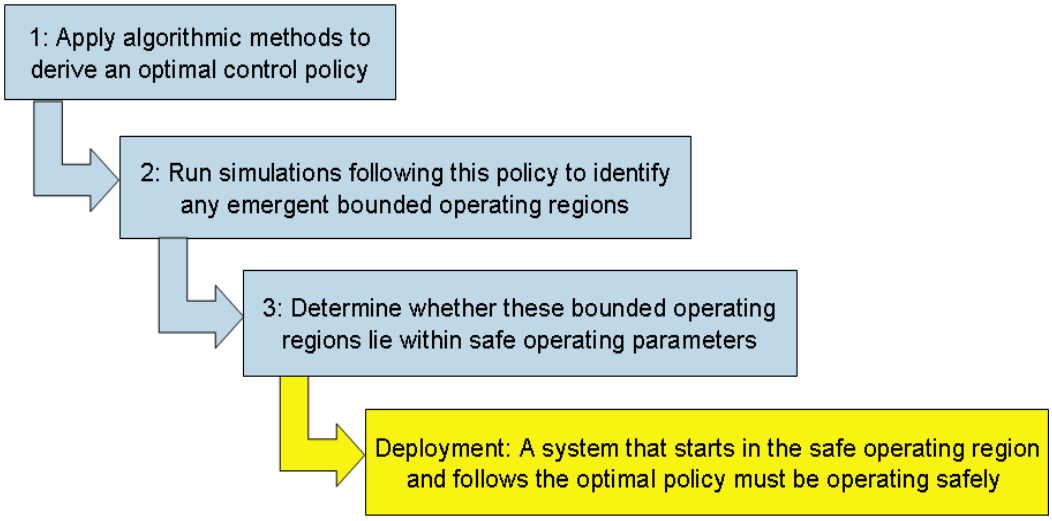

Figure 60 – The Proposed 'Novel Safety Assurance' Process

Working with a toolset similar to that developed in the main body of this thesis, stage one would involve generating an optimal policy for the system to follow, perhaps based on training data gained from an artificial environment (it would not have to be a Q-learning algorithm in general – one should pick a solution method suitable to the problem in question). Stage two would be primarily a simulation exercise, to highlight where any of the issues discussed might arise (concerning safe operating regions and ensuring the operating policy remains safe), to facilitate any subsequent (iterative) improvement of the control policy or refinement of the

model (depending on which is determined to be more appropriate). Stage three would then be a performance analysis of how well the system performed against the necessary safety criteria when following its optimal control policy. Note that, in practice, there might be many more stages than this, with more thorough training and assessment of control policies, and an investigation into the performance of such methods at identifying and classifying safe behaviours. The process described above and the theoretical background that follows are intended to give a broad overview of the key stages that are envisaged.

C.1.2 Safe Operation, Boundedness, and Optimal Control Policies

The question considered in this annex is whether the mathematical models and toolsets developed for the complex systems control problem in the main body of this thesis can be taken further to also provide an analytical basis for the subsequent assurance problem – demonstrating and proving that the observed behaviours are safe and appropriate as part of the novel safety certification process proposed. The end goal would be to have developed novel certification techniques for the purpose of verifying the bounded behaviour of an autonomous (or multi-agent) system with a hierarchical decision controller, with additional consideration given to the challenge posed by decision-making-under-uncertainty problems. The connection with MDPs is that if an optimal policy for an assignment has been determined, using the techniques described in this thesis or otherwise, then that policy defines all the actions the system should take and thus determines the possible dynamics of the system in state space. Analysis of these dynamics should reveal whether or not the system can be said to be bounded within any particular region of the state space.

Translated into the language of MDPs, the proposition is that the actions specified by an optimal control policy correspond to a safe policy when:

a) The appropriate safe operating parameters define a bounded region of the state space, known hereafter as a safe operating region. [Subsets of this are also safe operating regions, and they could be disjoint from each other in certain contexts.]

b) The system operates within this safe operating region.

c) The actions selected when in the safe operating region and following an optimal policy are bounded (in the sense that they do not take the system out of this region).

Condition (c) means that, given a system's current state is within a safe operating region of the state space, the state transitions likely to follow from selecting the optimal action keep the system within this safe region of the state space. Combined with condition (a), this means that an optimal policy that leads to state transitions being bounded within a safe operating region can be described as a safe policy. The easily-overlooked condition (b) is required to ensure the system is in the safe region in the first place – the boundedness conditions are redundant if the system is never even in that region. If these three conditions can be verified, this could be a valuable contribution to or even the basis for a safety case.

Note that despite the comment that safe operating regions could be disjoint, in practice this would often not be useful since most applications will involve continuous movement of an autonomous system. Passing through an unsafe region on the way from one safe region to another is still unsafe. Since the possible applications for disjoint safe regions appear rather limited (software agent applications that deal with logical states are one possibility), these discussions focus on connected safe operating regions – connected meaning that all points in the safe region can be reached without leaving the safe region, analogous to the formal definition of a connected set from set theory.

The next step is to consider these conditions applied to the simple MDP examples described during the development of the mathematical framework in this thesis (Chapter 5). Referring back to the example of the grazing cow from Chapter 5.6.3, one could define a safe policy to be one which does not involve wadding across the stream (imagine the stream is now a much deeper, wider river following a torrential rainstorm). If one starts on the right hand side of the river, the optimal and safe policies agree – you should stay there. Being on the right hand side of the river and following the optimal control policy together satisfy conditions (a) and (c) – assuming condition (b), that you are indeed on that side of the river, then behaviour can be said to be safe. However, if one starts on the left hand side of the river they disagree – the optimal policy suggests you should risk a crossing and the safe policy says you should not. In this case the system cannot be declared safe without first altering either the safety requirements or the optimal policy (the latter would typical involve redefining the system goals, via changes to the reward policy).

The other simple example, featuring the crossing of a rickety bridge, has an optimal policy which implies that one should stay on the left hand side of the bridge. The logical safety criteria would be to specify the left hand side of the bridge as the safe region, and then the optimal and safe policies agree and the system could be declared safe. Problems arise with this example if,

for some reason, the right hand side were to be declared the safe region. It would then be impossible to satisfy the safe and optimal policies simultaneously. Although this is acknowledged to be a rather artificial example as the right hand side of the bridge in this simple example is a terminal state, and it would be unusual to specify that a system is only safe once it has terminated. The key feature being highlighted is the potential for inconsistency or disagreement and thus necessary trade-offs between safety and optimality.

Figure 61 below illustrates a more general example of an optimal policy applied to a region of the state space being bounded within that region. The arrows are the possible actions that are identified as optimal actions (i.e. those which maximise the expected return from the current state, or maximise the value of that state). If the system is within the bounded region, where the bounds correspond to some prescribed safety criteria, then the bold purple arrows represent the safe and optimal actions corresponding to the safe region in question.

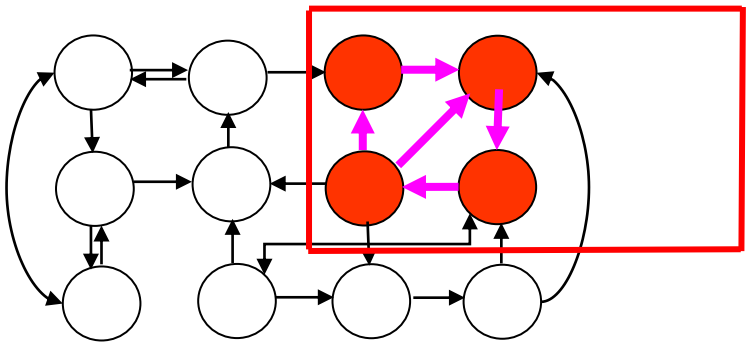

Figure 61 – A Safe or Optimal Subset of the State Space

The safe and optimal policies here do not match up exactly – there are other optimal actions identified that are labelled as potentially unsafe (black arrows out of the safe region). If these can be explicitly identified then they can be deleted from the optimal policy (perhaps using an action filtering technique such as was introduced and demonstrated in the experimental activity within this PhD reported in Chapter 8), suggesting that in some cases the intersection of safe and optimal policies will suffice. Note this will not always work, as the intersection would be an empty set if every optimal action were declared unsafe. However, in many cases (such as the example illustrated by Figure 61), restricting a system's choices just to those actions represented by purple arrows is both safe and optimal, and under such a policy the system could be declared safe (provided of course that the easily overlooked condition (b) is also satisfied – the system must be in the safe region in the first place for any of this to apply).

In practice, it should be expected that state space dynamics and the corresponding geometry will be more complex than identifying the particular state one might want to remain within (as with the two simple examples) or boxing off a corner of a discrete state space lattice (such as in Figure 61). Higher-dimensional geometries would be characteristic of many problems, depending on which state space representation is most suitable for the system in question. Systems will also typically have continuous state spaces for most applications, and they should remain safe during state transition as well as in the state at each end of the transition (as noted above in the discussion about disjoint safe regions). Depending on how the state space is defined, this could be an important and challenging consideration during the safety assessment.

C.2 Iterated Function Systems and Fractal Geometry

The vision of earlier research to which the author contributed [68],[23] was that the effects of action selections on the state space of a system should be modelled, within the MDP framework, as an Iterated Function System (IFS). Each action selection is then an iteration on the system's state space, leading to a corresponding geometric transformation of that state space and the geometric (typically fractal) pattern associated with that IFS emerging. Analysis of these patterns should then help determine the safe operation of the system (or otherwise). In this section, the key concepts behind IFSs and their fractal representations are summarised.

[It should be noted that the background thinking and the innovation to apply the IFS formalism is not a contribution from the author directly. The content of this subsection is based on work conducted in parallel to the author's own precursor work and also draws from the well-known text by Barnsley [4], but it is necessary to explain it so that the author's own contributions covered in the subsequent sections make sense.]

An IFS is a set of functions $\{w_i\}_{i=1,2,\dots n}$, where each w_i is a contraction mapping from some region of the state space to itself. A contraction mapping is a mapping where, for all w_i in the IFS and for some $s \in [0,1)$ (called the contractivity factor), $d\big(w_i(x), w_i(y)\big) \leq s \cdot d(x,y)$ for any two points x and y in the state space.

Since $d(\cdot,\cdot)$ means the distance between the two points and $s < 1$ by construction, the name contraction refers to the fact the mappings w_i are bringing all the points closer together, literally contracting the state space. As a series of mappings w_i are made, the state space continues to contract towards an attractor, which can best be described as the set of points that remain

reachable after infinitely many mappings. Typically, this attractor will be a fractal pattern in the state space.

A formal definition is given by Barnsley [[4], p80, definition 7.1] along with much more background material on metric spaces that it is not necessary to repeat here. (Technically, one has to ascertain that the state space in question and the metric with which one calculates distance is properly defined as a metric space in order for any of the following to be valid.)

A series of mappings from the IFS on an initial starting point p results in an orbit through the state space - a sequence of points $\left\{p, w_1(p), w_2\left(w_1(p)\right), w_3\left(w_2\left(w_1(p)\right)\right), \ldots..\right\}$ where each of $w_1, w_2, w_3, \ldots..$ are chosen from the possible w_i in the IFS. Each combination of mappings $w_1, w_2, w_3, \ldots..$ selected results in motion around the state space that may seem arbitrary but does in fact correspond to that particular sequence of control actions in a deterministic way (sometimes uniquely).

If the appropriate conditions are satisfied, then left to run indefinitely the orbit of some point in the state space will end up at or arbitrarily close to the attractor of the underlying IFS, which is the set of limit points for the orbits of any possible combination of w_i that could have been selected. A starting point on or sufficiently near to the attractor (formally, a starting point within the basin of attraction for that attractor) will lead to bounded behaviour, as the orbit of the point is guaranteed to converge to the attractor.

To illustrate an IFS and its fractal attractor, the following is an adaptation of a common example of the Cantor set, which is an example widely used in the literature on fractals and more general set theory (Barnsley [4] and Falconer [31] are two of many examples).

Consider the IFS over the two-dimensional state space $S_0 = [0,1] \times [0,1]$ (for convenience) defined by the functions:

$$\begin{cases} w_1(z) = \frac{1}{4}z \\ w_2(z) = \frac{1}{4}z + \frac{3}{4} \\ w_3(z) = \frac{1}{4}z + \frac{3}{4}i \\ w_4(z) = \frac{1}{4}z + \frac{3}{4} + \frac{3}{4}i \end{cases}$$

Each of these four functions is a contraction (since the state space is contracted by a factor of a quarter each time), and maps the state space to one of four corresponding but smaller subsets

of the original space. Choosing another of these mappings and transforming the state space again, mapping the state space to one of sixteen even smaller regions of the state space.

The two-dimensional Cantor set S_n generated by this IFS is the set of all points that can be reached following a series of *n* such mappings. Illustrated in Figure 62 below are S_0, S_1 and S_2 – the set of points one could be in after zero, one or two mappings. What further iterations of the system will look like from here following further such mappings is hopefully clear.

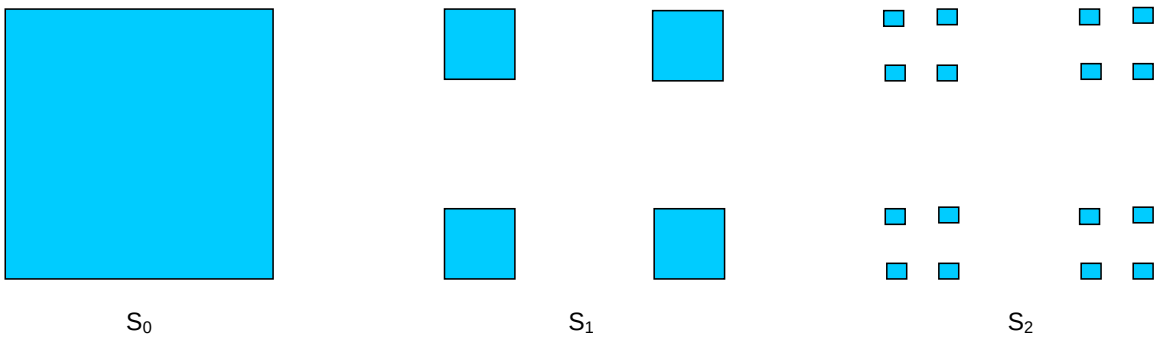

Figure 62 – Two-Dimensional Cantor Set Generated by an Iterated Function System

The attractor of this system is the set of reachable points after infinitely many of these mappings, which will be a totally disconnected (i.e. no point in the attractor connected to any other). This is the same set as the set of all the limit points of the IFS. This is impossible to illustrate, but hopefully the reader will have in mind what this would look like. Since the mappings in the IFS are all contraction mappings, for any point in the original state space S_0 and any combination of mappings $w_1, w_2, w_3, \ldots$ then the orbit is guaranteed to be arbitrarily close to the attractor after some number of mappings.

The problem that may arise with this mathematically simple construction is that a totally disconnected attractor may not allow for paths between points in the attractor to remain within the attractor themselves. In Figure 62, for example, it is easy to see that the bottom left most point $(0,0)$ and the top right most point $(1,1)$ are both within the attractor (both are reachable by one of the mappings w_i and will remain so at all time steps), but after just one iteration it is no longer possible to draw a path between them without passing through unreachable points.

This is important because of the discussion in the previous subsection about disjoint safe regions – if the eventual aim is to have the IFS modelling the control actions decided by a system controller, and the system is meant to operate safely, then it is not just the attractor of

the system which should be within the safe operating region. Paths between reachable points must also be within the safe operating region. But if the attractor is totally disconnected and spread across the entire state space (as in this example), then the safe operating region must either be a set of disjoint safe operating regions or be one safe operating region covering much if not all of the state space. In the former case, state transitions (mappings from point to point) cannot be said to be safe even when they appear to be (because the trajectory between two points in safe regions of the state space passes through potentially unsafe regions) and in the latter case any further analysis is trivial since virtually every state and possible subsequent transition is deemed to be safe.

The proposed solution to the problem of connectedness is related to the concept of code space and the labelling of different orbits with the combination of mappings necessary to reach the current point from the starting point. Barnsley covered this in detail and Williams summarised it [68], and it is not necessary to repeat the detail here. The important point is that the attracting regions need to be made to overlap (which includes simply bordering each other) so that transitions through the state space can remain in safe operating regions. Williams provided an example of this concept, by modifying two of the functions in the IFS defined above:

$$\begin{cases} w_1(z) = \frac{1}{4}z \\ w_2(z) = \frac{2}{3}z \\ w_3(z) = \frac{1}{4}z + \frac{3}{4}i \\ w_4(z) = -\frac{1}{2}z + 1 + i \end{cases}$$

Now, the sets S_0, S_1, S_2 are:

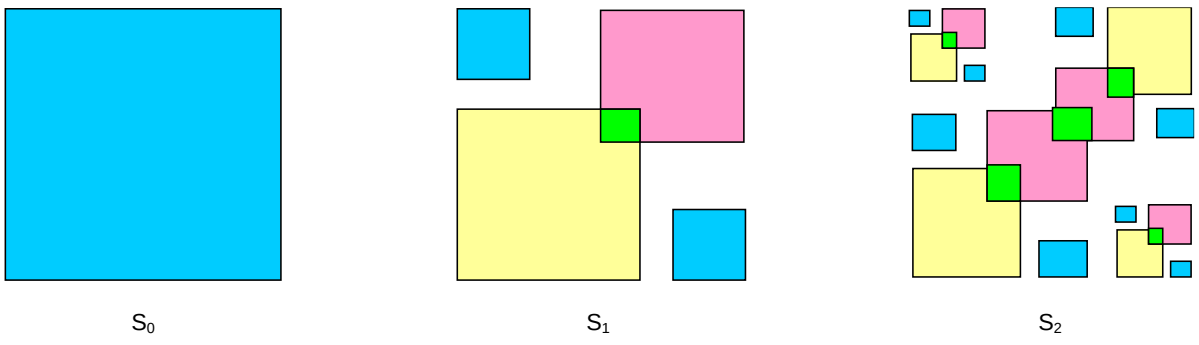

Figure 63 – Modified IFS's Reachable Set (featuring overlap)

This IFS has resulted in a continuous path in the state space from the bottom left corner to the top right corner, and this path will be there in all reachable sets S_n. In the area along the overlapping diagonal, the system has a choice of paths from point to point, just as one would require of any real-world system (albeit a constrained choice of paths). This is more like the shape of attractor that will be needed by any practical control system.

Unfortunately though, because of the ambiguity inherent in an overlapping IFS (there are points that can now be reached by more than one combination of mappings, rendering the behaviour non-deterministic) the system is now considerably harder to analyse. One solution Williams recommended for this problem was to make use of Barnsley's Shadow Theorem [[4], IV section 6]. By imagining the fractal pattern is actually the shadow of several separate reachable sets in higher dimensions (in this case, the two dimensional shadow of a three dimension family of reachable sets) it is possible to define entirely deterministic orbits through the state space even if the paths between points pass outside these sets – these reachable points and the permissible paths between them could all lie in the contiguous shadow of the fractal. This was illustrated with the following diagram:

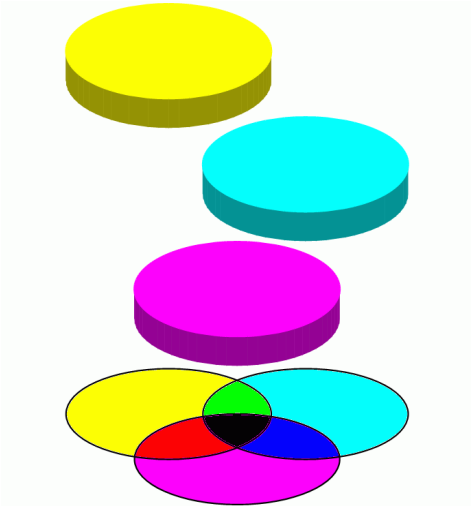

Figure 64 – Illustrating the Shadow Theorem

It was not thought possible to go into any greater detail, because how to construct such shadow projections would be highly dependent on the particulars of the system in question, and would therefore best be illustrated by detailed analysis of a case study. This work has yet to be undertaken. As usual, further detail on the theoretical principles can be found in Barnsley [4]; the key point for this discussion is that an IFS that produces disconnected attractors (and thus

disjoint safe regions) may still be able to produce continuous trajectories between points if appropriate shadow projections can be constructed.

It is important to specify the difference between the *attractor* and the *basin of attraction* for a system. The attractor is the set of limit points to which the series is converging. The basin of attraction is the set of points that will, given enough mappings, converge to this attractor. Ideally of course the basin of attraction will be the entire state space and the analysis will be simpler, but unfortunately this will not generally be the case. It is easy to imagine a system with two or more attractors, where from some states one attractor is reachable and from other states the other attractor is reachable (think about some over-balancing object and the difference between being at a small angle on each side of the tipping point).

The attractors themselves are either safe or unsafe (neglecting the possibility that the attractor straddles safe and unsafe regions). To assess the safety of the system, one must consider whether the basins of attraction for those attractors representing safe behaviour:

- Are entirely contained within the safe operating region of the state space;
- Contain all of the allowed initial conditions.

If both of these are true, then the system can be said to be safe (at least in normal operation).

A slightly weaker form of this assurance is that:

- The attractor is entirely contained within the safe operating region;
- The basin of attraction contains all the allowed initial conditions;
- All trajectories leading from the allowed initial conditions to the attractor are disjoint from any unsafe regions.

If all of these are true, then the system can also be said to be safe (at least in normal operation). Note however the additional condition imposed, namely that if the basin of attraction intersects any unsafe regions, then one must restrict the system only to those trajectories that remain in the safe region, which will require additional analysis. It would in general be preferable to have the entire basin of attraction within a safe region, but whether this is possible will depend on the particulars of the system in question.

Note that if an attractor straddles safe and unsafe regions, it will be more difficult to follow these safety criteria. It is unclear how to deal with this eventuality within the IFS framework, and typically one would have to consider using a different set of mappings and/or a different

state space representation that better incorporates the safety criteria from the outset. Indeed, designing the system's controller in such a way that it is more amenable to safety assessment will be important, although it is unfortunate that it is not always clear from the outset how to do this – one possible approach to engineering safe behaviours into the controller is one of the features demonstrated by the model developed in this research (see Chapter 5). Avenues for further investigation remain regarding how to deal with attractors that are not clearly safe or unsafe.

C.3 Assurance of Complex Multi-Agent Systems

The goal that motivated this PhD was to begin to translate these concepts into the context of a complex or multi-agent system. The earlier proposals for 'novel safety assurance' techniques for autonomous systems are a logical place to begin, since agents within a complex multi-agent system are a form of autonomous system. Whether any of the theoretical discussions about convergence of dynamical systems, novel geometric constructions and safe operating regions within state spaces could be translated into this more complex problem domain was not clear, given that the prior theoretical treatment was for a single autonomous system and in any case was only a proposition, but this was one of the original research intentions behind this PhD.

The annex has been structured as it has for a reason. The preceding subsections are a summary with only minor revisions and updates of some relevant mathematical background and some earlier work led by Williams which the author of this thesis only played a supporting role in. The remainder of this annex picks the story up from there, and is the author's own work, revisiting the earlier content, considering how feasible it would be to extend it to the domain of interacting autonomous systems and ultimately into the domain of complex multi-agent systems, and presenting the author's thoughts on where and how these proposals might be applied. Some updated conclusions and recommendations are provided at the end.

C.3.1 The Iterated Function System Model

Traditional control is concerned with two issues: the correct behaviour of the controlled system, and its stability. In relation to safety assurance, the notion of 'correct' behaviour is simplified to 'bounded' behaviour, and the aim is to identify methods which discover the bounds of the reachable region of state space, which can be equated with the bounds of the behaviour of the

system. However, there is a strong connection between boundedness and correct behaviour, and the proposed modelling methodology summarised in the previous sections may provide a means to investigate this.

The IFS approach is a finite set of mappings on a suitable metric space which, after infinitely many mappings, produces a fractal pattern. This seemingly abstract piece of mathematics provides a potential mechanism for constructing models of the reachable state space of a system, if the functions in the IFS are representations of control actions and the space on which the IFS operates is an appropriate system state space. Finite numbers of mappings may not produce formal fractals, but might provide the necessary mathematical structure to analyse the span of reachable states.

The earlier work considered the possible application of this approach to autonomous systems applications, and the integration with the MDP control approach to produce an integrated design approach for autonomous systems. Given the 'fit' of the MDP control approach and the proposed IFS assurance approach, and the generalisation of the MDP approach to support a whole spectrum of complex systems paradigms as covered in this thesis, it is logical to consider whether the IFS approach can similarly be generalised to a spectrum of more complex systems, including complex multi-agent systems.

First however, it is important to document some of the modelling assumptions that underpin the IFS approach (in addition to all those assumptions listed in the previous section). Formally (adapting Barnsley's definition), each possible action selection is required to be a function a_i acting on state space *S*, where (S, d) forms a metric space with a suitable metric *d* and each a_i is a contraction mapping. Note that, for this model to be valid one has to assume that each action a_i can be approximated by an *affine* transform $T_i(s) = A_i S + b_i$, where A_i is a linear transform and b_i is a constant vector in the state space.

Two concepts to introduce for the unfamiliar reader are that of the *metric space* and the *contraction mapping*:

- A *metric* is a distance measure which has to satisfy particular properties (positive definiteness, symmetry, and the so-called triangle inequality), a similar concept to but not the same as a norm. A mathematical space coupled with a suitable metric defines a *metric space*. Familiar Euclidean geometry with the normal 'distance' metric qualifies as a metric space. However, there are many other kinds. Generally speaking, most state spaces in which human-designed systems will operate in will be metric

spaces, and this issue will not be addressed further herein. However, many of the necessary analytical assumptions would require this to be formally verified before any analysis can be considered valid.

- A *contraction mapping* is a particular kind of mapping on a metric space that requires that two points in the state space prior to the mapping cannot be moved farther apart by the mapping. A constant known as the contractivity factor, which is a positive value strictly less than 1, is the maximum ratio of the distance between the images of the two points in the state space after the mapping has taken place compared to the distance between the two points before the mapping took place.

Contractivity does initially seem to be quite a restrictive condition to place on a control function, but what it corresponds to is guidance that all actions selected by the controller do not deliberately cause the divergence of the system in state space. Considering that the state space representation is meant to be a reflection of the real behaviour of the system in normal space, divergences in state space should correspond to divergences in the real world which would generally be undesirable, and therefore such a restriction is not unreasonable. Some loosening of this requirement, in terms of allowing slightly more flexibility from the controller as long as the potentially digression is strictly less than certain parameters linked to the degree of environmental perturbation has been explored previously (Williams' half of [23]).

How does this help with determining the boundedness or stability of the behaviour of an agent? The answer is that, provided all the relevant assumptions are valid and given suitable state and action spaces have been defined, the range of possible actions available to a system controller can be represented as an IFS, and the consequences of the control policy that the controller is following duly correspond to particular trajectories through the IFS. The IFS formulation is particularly useful because the mathematical apparatus for IFS's will carry over directly to the assurance problem. A series of mappings from one of the actions defined as part of an IFS creates a series of subsets of the state space that the agent should now be in.

The boundedness condition can be difficult to demonstrate. Simple algorithms can be found in the literature on computer graphics (for example Edalat [29] and Chu and Chen [19]), but these only attempt to discover simple closed convex spaces: boxes and spheres. While these are adequate for the purpose of displaying a fractal on a monitor, they do not address the more complex issues which arise when a specific *operating envelope* must be achieved. Fractal bounds are proposed as a possible approach to a more general solution for this problem.

C.3.2 Overview of the Proposed Approach

Traditional approaches to assurance involve the detailed analysis of a system's design, to provide evidence of its compliance with those requirements that it is expected to satisfy. This includes detailed analysis of documentation, formal design reviews, techniques such fault tree analysis or others, techniques based on the application of first order logic, and other inputs, depending on what is suitable for the system in question. The goal is to build a body of evidence that demonstrate the acceptable behaviour of a system. Multi-agent systems are an example of an emerging technology area where the systems have become too complex for this to be practicable, complete or reliable (i.e. the system complexity is too big for humans to reliably assess, and to be sure of how reliably they have assessed), and, as discussed in Chapter 4.4, where the risk of complex or emergent behaviours is a significant stumbling block, which is not accounted for at all by traditional approaches.

In practice, techniques to model, understand and analyse complex behaviours are expected to supplement other inputs into the assurance process, rather than be an end in themselves. Note that it is often the performance requirements (as opposed to the procedural requirements) that become particularly difficult with increasing system complexity, and nothing in these discussions is intended to apply to procedural aspects; for example, validating the software engineering methodologies followed. It is specifically understanding the 'holistic' system behaviour and understanding the implications of complex system designs that is the particular 'missing link' that this work attempts to address.

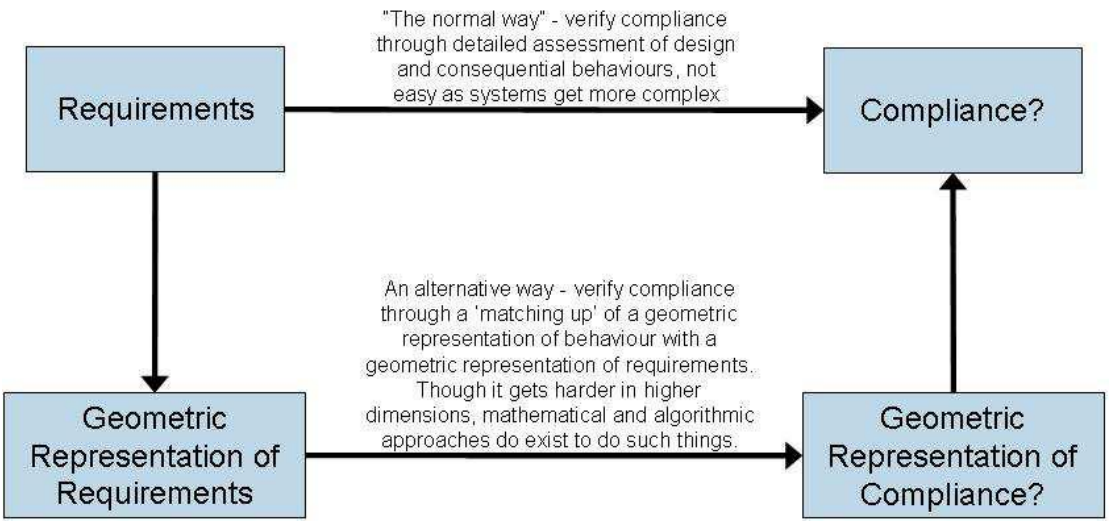

Figure 65 – Novel Assurance for Complex Systems

Figure 65 represents the relationship between the proposed approach to the analysis of system behaviour and 'normal' assurance, supplementing the conventional approaches with additional analysis of the whole system behaviour. This should capture more information about any emergent 'subtleties' in behaviour that might otherwise have been missed, which should provide a higher degree of confidence in the thoroughness and fidelity of the assessment.

C.3.3 Geometric Representation of Requirements

The essence of the proposed approach is to represent the full span of possible behaviours geometrically, generated by an IFS, and determine the compliance (or otherwise) of this span against a geometric representation of the requirements. The IFS will provide the geometric representation of behaviours and some of the analytical tools, but first there is a question as to what is meant by a geometric representation of requirements.

It is important to reiterate the distinction between *procedural* compliance (not addressed by this work) and *performance* compliance. To formally validate system *performance*, it must be possible to appropriately metricate system requirements. Requirements of the form "The system shall be efficient" (which the author has seen in a genuine customer requirements document) do not have any formal or parametric meaning, and are impossible to metricate. Of course, unmetricated requirements are generally not enforceable anyway due to ambiguity in what they really mean. The important point is that this work is focused exclusively on demonstrating compliance with formally metricated requirements; for example, "The operating temperature shall not exceed x degrees" or more pertinently "Controlled deviations from the required trace shall not exceed x metres" (relating back to the experimental models presented in Chapter 8).

Assuming such, the logic of the IFS approach is as follows:

- If the state of a system is described or measured by some set of quantifiable parameters; and
- If the behaviour of the system is the observable or measurable progression between states; and
- If the requirements expected of a system can be defined as a range of allowable states, and hence a set of allowable parameters;

- Then behaviour compliant with requirements means that the span of reachable states is either within, or can be constrained to be within, the range of allowable states (parametrised as appropriate).

Suppose, for example, that the state of a system can be adequately represented by two variables x and y (for example, displacement and temperature, position, and time...). Performance requirements could take the form of bounded or unbounded operating regions, for example:

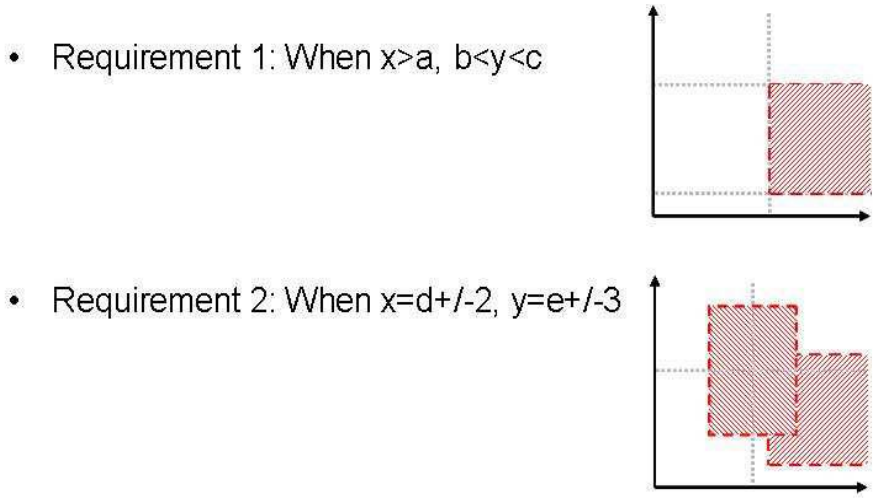

Figure 66 – System Performance Requirement Translated into Geometric Envelopes within the Reachable State Space (1)

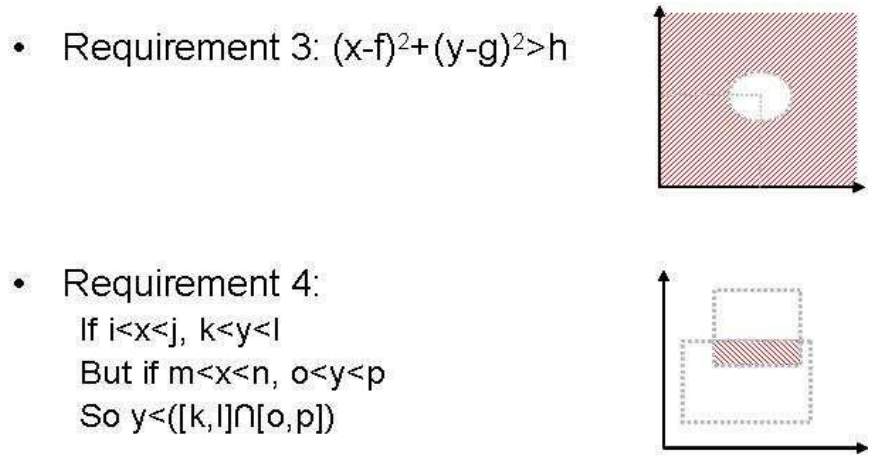

Figure 67 – System Performance Requirement Translated into Geometric Envelopes within the Reachable State Space (2)

Given the full set of applicable requirements, from the customer or from any relevant regulatory authorities, this should be compiled into the overall permissible state space, or the '*operating envelope*'. In theory, it should not be necessary to do any more than to accurately define the starting point and the possible reachable state space implied by the system's control algorithm (by generating the reachable span by interpreting the control algorithm as an IFS), and to compare this against this identified allowable span.

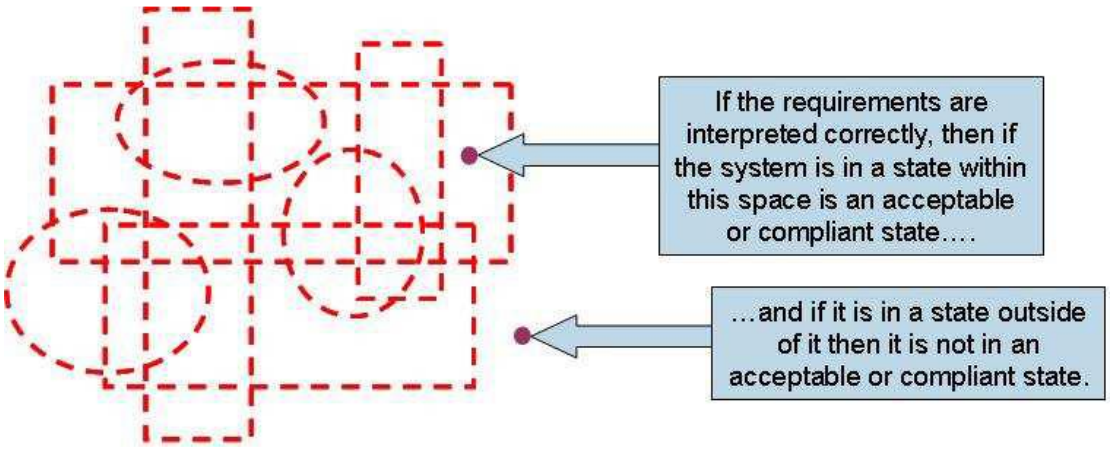

Figure 68 – System Performance Requirements Translated into Geometric Bounds

In reality, state spaces will generally not be convenient two dimensional spaces, and acceptable state space ranges will not generally be a set of overlapping circles and rectangles. Interpreting requirements will often be more complicated and more subtle than this. The actual operating region will in general be a composition of different regions of different combinations of state space variables, with the overall 'operating envelope' described by a set relation consisting of a suitable composition of set unions, intersections and complements corresponding to the appropriate requirements.

In general, requirements may imply a disconnected or even discrete set of permissible states. Not only is this harder to visualise but the allowable state space created from the composition of all the requirements could also be disconnected or discrete. However, this is not inconsistent with the allowable operating region being described by a composition of set unions, intersections and complements, nor is it inconsistent with the representation of a control algorithm as an IFS. That the same representation can simultaneously include continuous and discrete variables as part of an overall operational boundary for the whole system is the primary reason for advocating the IFS approach.

A few other points to note:

- The proposed approach has some logical similarities to the conventional 'formal logic' approach to demonstrating compliance.
- Actually representing the permissible state space graphically is not necessary, presented here for illustrative purposes. Indeed, higher dimensionality and increasing numbers of state space variables would rapidly make a visualisation difficult if not impossible to even produce, never mind for a systems safety practitioner to draw meaning from. In practice, as which much modern geometry, analysis would have to be done numerically, by computer.
- 'Geometrically representing the requirements' really means 'A geometrically inspired method for compiling the requirements'. The logical nature of the requirements themselves, and what they imply for the allowable behaviour of the system, should not be changed by their geometric compilation (assuming that they are well formulated, internally consistent, and measurable requirements to begin with).

C.3.4 Geometric Representation of Compliance

Compliance, in the behavioural or performance sense, means that all initial and reachable conditions of the system are within the allowable space (i.e. that all possible choices of action lead to a state change that remains within the allowable space and, where applicable, transits only through allowable regions of the state space). The essence of the proposed approach is to 'match up shapes'; in other words, to construct a geometric object that represents the span of reachable behaviours given a particular starting state and control algorithm definition, and to compare this span against the operating envelope corresponding to system requirements, to demonstrate the extent to which behaviour will be appropriately bounded.

With assurance being about the provision of evidence, the ideal outcome would be to show that the preferred control algorithm for a system can only lead to system behaviour that is fully compliant with the (measurable) performance requirements of the system. However, it would still be a useful output to be able to identify any non-compliances, and specifically the circumstances or chain-of-events that would cause these non-compliances.

The use of the IFS method for generating reachable state space spans provides a means to generate this kind of evidence. This section discusses the key steps that would need to be taken

in order to apply this method to the generation of evidence to support assurance. Recall that the primary motivation for using an IFS model is to generate representations of state space spans, and the corresponding notion of code space. The reachable state space is the set of points that can be reached by an allowed series of control actions, and the *orbit* or *trajectory* of the system in state space is a representation of the series of control actions taken, connecting the system's starting state to its anticipated finishing state (the smoothness of the curve dependent on the frequency of actions been taken).

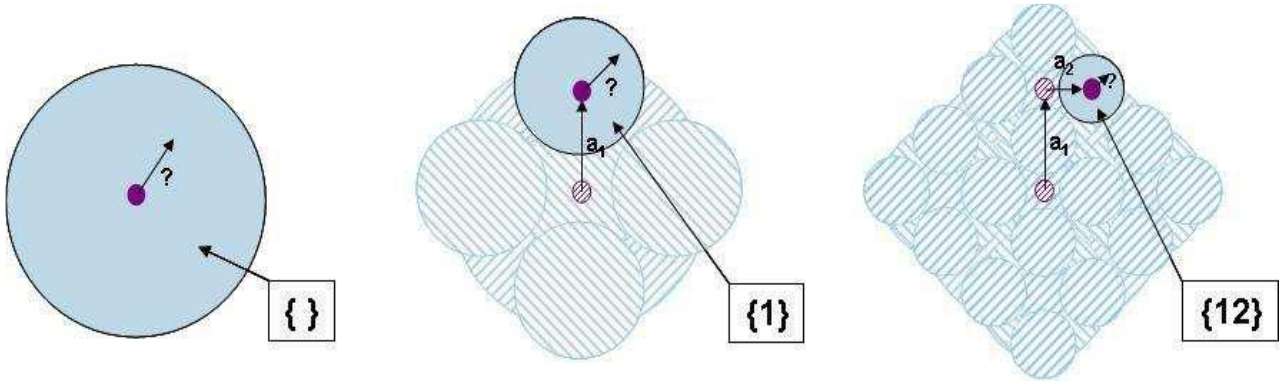

Figure 69 – IFS Representation of a System's State Space Trajectory

As illustrated in Figure 69 above, this trajectory can be concisely described by a code space signature, identifying the series of actions to which that trajectory corresponds. A code is a sequence of numbers labelling the series of actions taken, where finite substrings of the code correspond to regions of the reachable state space, i.e. given some initial condition (starting point, initial state) and a set of possible actions $\{a_1, a_2, a_3 \dots, a_n\}$ then a series of actions (e.g. a_1 then a_2 then a_1 again) defines a subset coded by the substring (e.g. $\{121\}$), which geometrically corresponds to the region in which the system's state should be found following that sequence of actions from a given starting state.

In the limit, a code such as $\{12132531242 \dots\}$ will describe the precise state of the system (barring significant system reconfiguration or environmental change. In practice, being able to identify the finite substring that corresponds to an undesirable region should be sufficient evidence of things not being right. In other words, code space provides a simple and concise mathematical representation of 'non-compliances' corresponding to particular strings (for particular non-compliant points) or substrings (for sets or regions of non-compliant points).

By reversing this process (i.e. firstly identify the undesirable regions of the reachable state space span and thereby identify the undesirable substrings and the corresponding sequences of non-compliant actions that would lead into those undesirable regions) it might be possible to create a mechanism for identifying those aspects of particular implementations that are those expected to cause undesirable behaviour, represented by Figure 70 below.

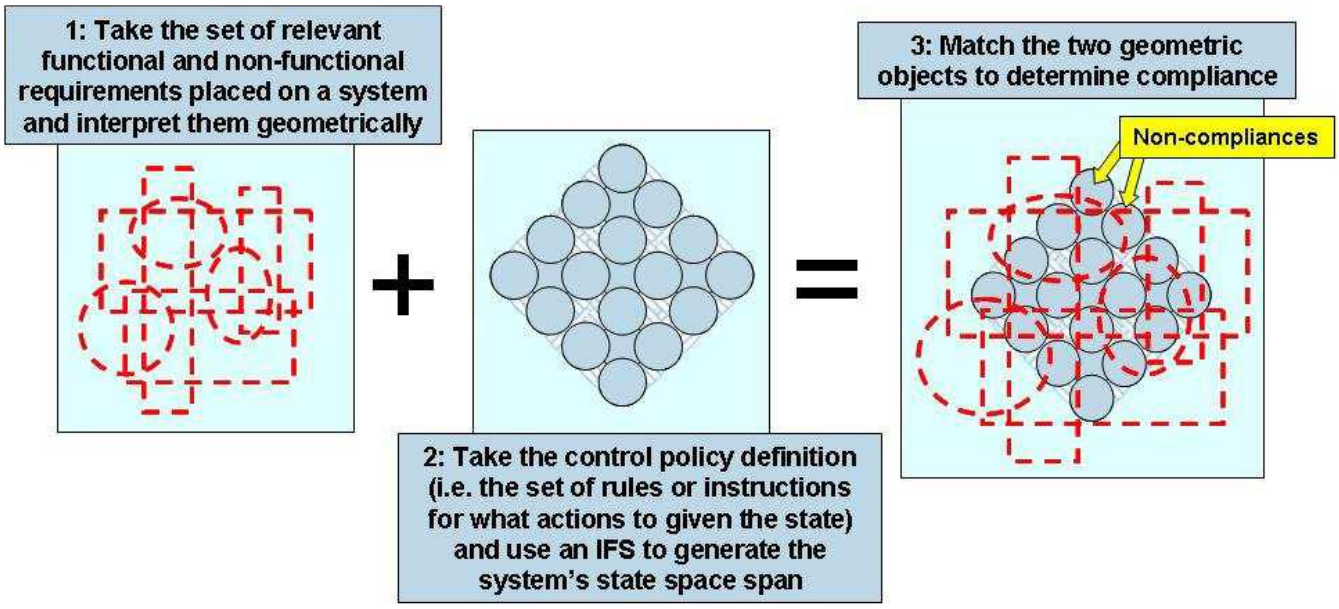

Figure 70 – Geometric Approach to Comparing Behavioural Span Against System Requirements

Identifying these non-compliances, and more importantly identifying the control substrings these non-compliances correspond to, gives the engineer 'options':

- Ban the implementation of undesirable substrings in particular circumstances (which is another instance of action filtering).
- Alert a user when particular substrings are (or are about to be) implemented.
- Change the control policy so that these substrings are no longer permissible.

Which of these responses, if any, is the most appropriate course of action is inevitably context specific. The key point is the early provision (ideally at the controller design stage) of evidence of any anticipated non-compliances implicit in the design, so an early decision can be made regarding mitigation of the problem, with justifying evidence to support to subsequent assurance.

Note that this process is analogous to a generalised model checking technique such SMV (Symbolic Model Verification, see Huth and Ryan [37], section 3.6) and CSP (Communicating Sequential Processing, see Roscoe [57]). It might be asked what particular benefits there are in proceeding with such an analysis of system behaviour when there are existing techniques that are logically quite similar. The primary benefits are the easy integration with the MDP control model, to provide an integrated and streamlined design process, and also the anticipated scalability to complexity discussed in the next section.

This method naturally supports the analysis of the acceptability of expected behaviour under a range of conditions, including variable starting conditions and external perturbation, with only relatively minor parametric changes required. This would be particularly useful were analysis to be conducted with a specialised or bespoke software toolbox. For example:

- Assessment of the acceptability of behaviour in the event of change in the system's starting configuration could be represented simply by a transposition of the state space span within the requirement envelope, as illustrated by Figure 71.

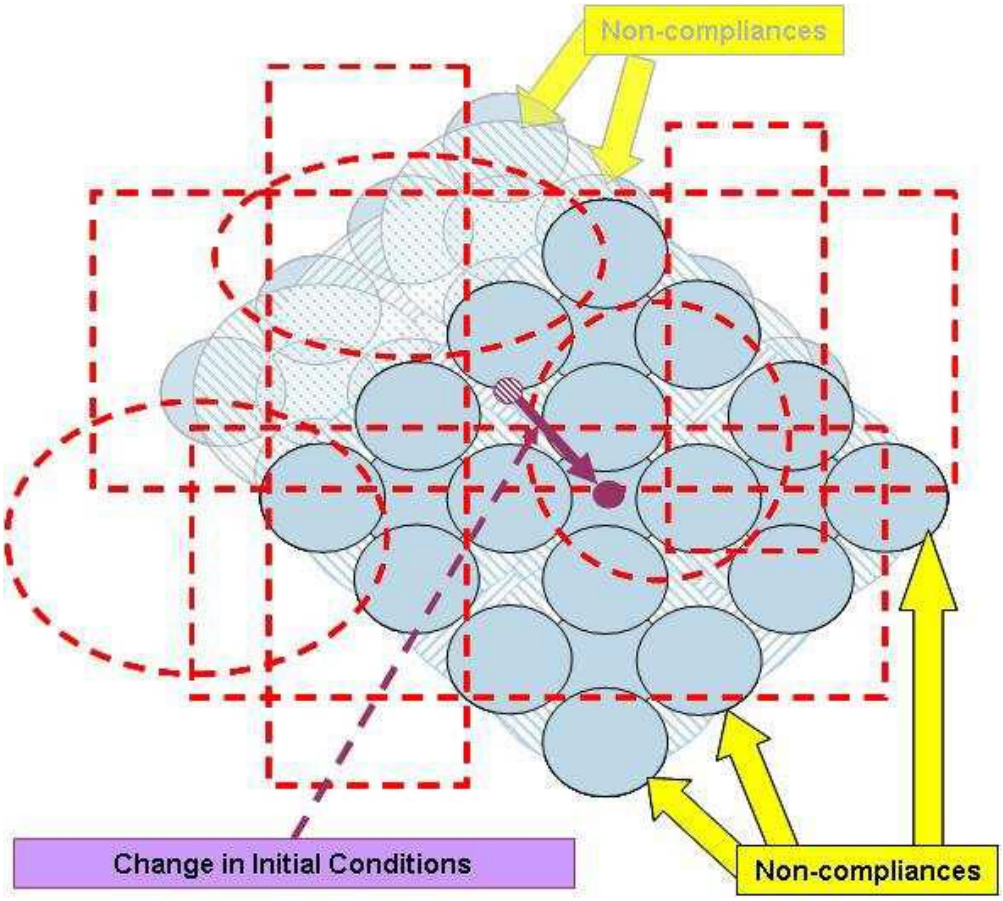

Figure 71 – Behavioural Compliance Under Changing Initial Conditions

- Assessment of the acceptability of behaviour in the event of significant external (e.g. environmental) perturbation can be represented as a distortion of the state space span, as illustrated by Figure 72. (The requirement envelope could also be distorted in some contexts, although this has not been illustrated.)

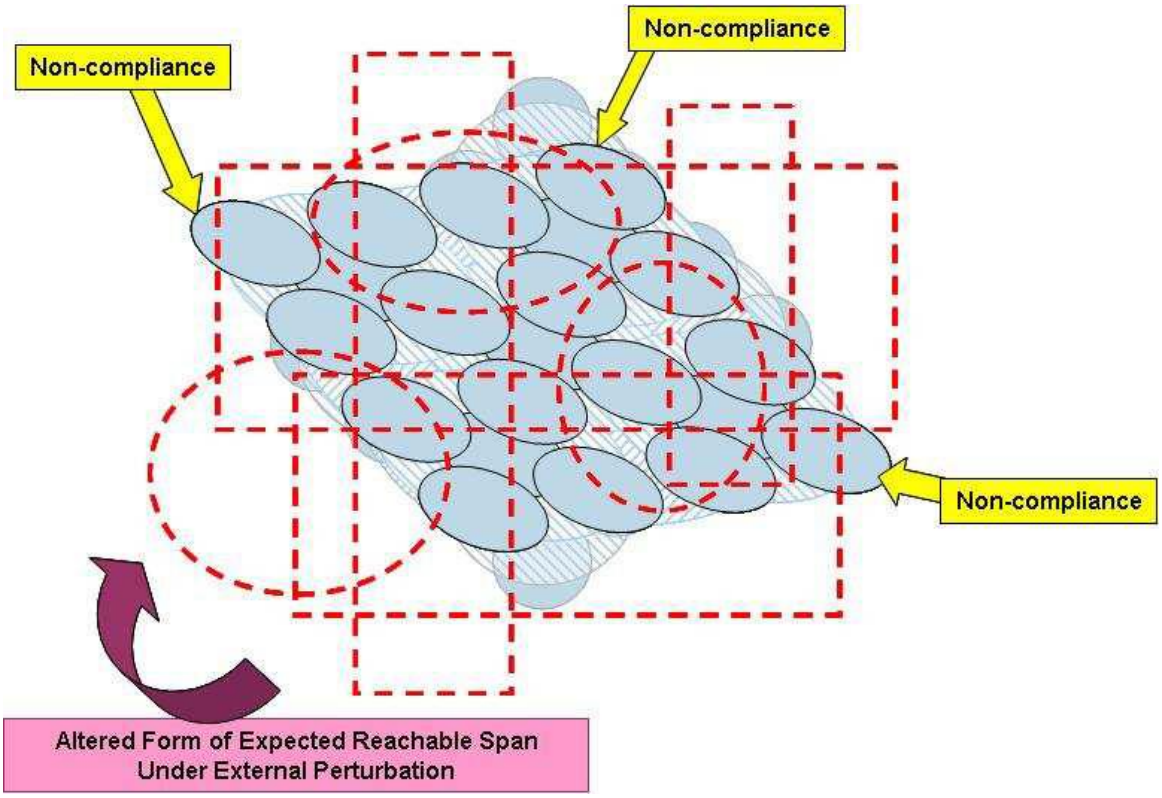

Figure 72 – Behavioural Compliance Under External Perturbation

It is important to note that behaviour does not necessarily need to be totally or finitely bounded. If certain variables are non-critical, then allowing some fluctuation in those variables may be acceptable, requiring the designer only to demonstrate the boundedness of those variables that are deemed to be critical. In other cases, proving certain bounds may be unnecessary if they are in some way implicit (e.g. it would be unnecessary to demonstrate that the temperature of a system component is greater than absolute zero; leaving an infinite lower bound is sound in such cases, since physics mandates the bound is there without needing to have it verified).

The important thing is to show that the expected span of system behaviour is suitably bounded by those appropriate operational requirements; whether these bounds are finite or infinite, discrete or continuous, depends on the context. The only requirement that cannot be avoided is the need for the bounds to be quantifiable or parametrisable, as discussed previously. This

is the reason for the continuing interest in the IFS approach – techniques already exist for demonstrating the boundedness of a fractal, even though such objects are characteristically complex, discrete shapes.

As noted above, there are some existing approaches to demonstrating the boundedness of an IFS, for example Edalat [29] and Chu & Chen [19]. Typically the approach is to demonstrate that there is a simpler geometric object (such as a box or a sphere) that contains the entirety of the fractal pattern generated by the IFS. Proving the overall boundedness of an IFS may be a useful input, but such techniques are not likely to be sufficient by themselves. The challenge is to demonstrate the system is bounded by its operational envelope, which is probably not going to be a convenient box or sphere.

However, the apparent limitations of these existing tools and techniques are not considered to be an insurmountable problem:

- If branches of the reachable span generated by the IFS (limited subsets of the overall behavioural span) can be shown to be bounded within, for example, a box or a sphere, and this box or sphere can be shown to be within the operational envelope, then a piece-by-piece transitive approach to assessing all branches of the fractal could be conducted. It is not so elegant as a single, whole system solution, and computational burden may become an issue if a particularly fine granularity is necessary to obtain 'pieces' of the reachable span that are can be definitively determined to be 'in or out' of the envelope, but there is the possibility that in at least some cases a piecewise approach using existing tools could be sufficient.
- Fractal bounding methods are available, and it may be possible to implement a 'boundedness checker' using techniques from this field. A more thorough investigation of the potential application of such approaches is recommended.

What practical benefits might the IFS approach bring to the assurance process?

- The anticipated range of behaviour is represented by the state space span, and so as long as appropriate mitigations for any non-compliances with the operational envelope are identified, then, strictly from the point of view of demonstrating suitable boundedness, this should be good evidence to support assurance. (If behaviour within the boundary needs to be more strictly regulated within these overall bounds, then demonstrating appropriate stability of behaviour might also be necessary.)

How does this relate to the earlier discussion about faults and failures in Chapter 4.4.1?

- Part of the assurance process is to consider what the system will do in the event of a fault condition. Fault conditions will be caused either by the controller definition being incorrect (i.e. not identifying the correct optimal action, a policy error), or those actions identified by the controller being implemented incorrectly or not having the anticipated effect (i.e. identifying the action that should be correct, but factors external to the controller cause it not to have the desired effect, perhaps because of a physical defect or damage, or because of greater-than-anticipated environmental perturbation).
- In the case that the control policy itself is wrong, the IFS approach is an advantage because the full behavioural span of the system implicitly contains all possible actions, with the assessment of possible non-compliances based on an analysis of the full spectrum of reachable states. If appropriate mitigations for potential non-compliances are identified, then the controller definition that results is already 'conditioned' against this type of fault, because, while a suboptimal choice of action may not be an ideal operation, it should at least be compliant, so it does not present a problem for safety assurance. This 'holistic' approach implicitly incorporates consideration of these types of failure conditions, without needing further assessment of any possible faults.
- In the case that the implementation of the control policy is wrong, the cause of any identified failure modes here will have to be considered on a case-by-case basis. However, this is no worse than the normal situation. The proposed representation may at least make it easier to identify and evaluate hypotheses such as the environment perturbation factor being larger than expected (i.e. generating another state space projection with specific parametric changes would be required to evaluate such hypotheses). Note that non-contractive functions in the IFS could lead to unstable behaviour in this case (digressions, potentially unconstrained, away from the stable operating configuration).

One final point to note is that if systems are to be designed with fail-safe modes, or with some degree of reconfigurability in the event of unexpected failure modes, then there is nothing apparent in the theory that would preclude the proposed techniques being used to assess the likely consequences. It has been beyond the scope of this initial work to consider such instances specifically. However, in the event of a fundamental change to the system's control policy, the effect this would have on the proposed approach would be for the system's state

space projection to experience a *fractal bifurcation* – in other words, as the control policy abruptly changes, and more importantly as the system parameters that inform this control policy change, the mathematical formulation of the IFS changes, and so does its geometric projection. The sudden change in qualitative behaviour of a dynamical system as a result of a change in parameter values is a *bifurcation*; reconfiguration in control systems would then be represented mathematically as an abrupt change in the shape and structure of the reachable state space, hence the description fractal bifurcation.

This is not a simple problem to analyse, using methods described here or otherwise, hence the perennial difficulty of not knowing how to safely manage or control reconfiguration in systems. The partial solution that arises from this work is that, while existing analysis of the compliance of a system would become invalid and need to be re-examined in the event of a significant system change, potentially with a different set of non-compliant substrings identified and mitigations to be determined, the process for conducting the analysis would be similar, so long as the new formulation of control policy is (at least approximately) understood or identifiable. The theoretical scalability of the approach even to such 'hard' problems like reconfigurable systems is a significant motivation for researching such an approach, although it is acknowledged that computational limitations would need to be considered before being sure of their efficacy to these envisaged applications.

C.4 Application to Complex Multi-Agent Systems

In the case of complex multi-agent systems, not only must the set of initial and reachable conditions for the system lie within the operational envelope identified for the system, but the initial and reachable conditions of subsystems and of individual components must lie within appropriate subsets of this operating envelope, and therein lies the challenge. If a system is complex, and the behaviour and hence the reachable span are different from those implied by the expected behaviours of the components and subsystems, then the operating envelope will not in general be a composition of the simpler equivalent 'sub-envelopes'. A particular consequence of this is that demonstrating the compliance of the individual components (i.e. that their reachable spans lie within their particular operating envelopes) does not imply that is true of the whole system, and vice versa. Assurance will require all relevant levels of granularity within the system to be demonstrably compliant, and it is neither obvious how to do this nor will it necessarily be tractable to do so by conventional means.

The proposition that this annex has been building towards is that if the system model and controller definition are:

- Designed for complex systems, with a decomposed state space structure and reward signals that can be shown to represent complex behaviour;
- Capable of representing subsystems as separate entities within a system using a state space projection, with an appropriate model for the relationship between local and global behaviour;
- In a form suitable for representing the control algorithms as transformations on a state space; then:
- The IFS approach applied individually to relevant combinations of components and subsystems should produce a family of operational envelopes for each combination, each of which can be assessed (automatically or manually) as described.

The essence of this proposed approach, and the reason for surmising a link with the three-axis MDP-based modelling paradigm, is that this structure, with the necessary modifications to model the differences between local and global state spaces, seems like a promising candidate for approaching this task, and the application of the IFS approach designed for simpler, interactive autonomy applications would hopefully carry over to the complex multi-agent systems domain in the logical way.

This appears to be a deceptively simple step, but given an appropriate model for complex multi-agent systems, then the subsequent step of projecting the reachable span and assessing this for compliance against (parametrisable) requirements is logical. Put another way, the assurance challenge arises from basic assumptions about the relationship between components and systems, and faults and failures, that is invalid in the case of complex systems, so the root cause of the problem and thus the source of the solution lies in identifying how to adapt the modelling and control steps to reflect this change, and then how to amend the solution tools to apply to these models.

What in practice would the application of the proposed process to a complex multi-agent system consist of?

- The engineer should do all those things described in the previous section, but for different subsystems and combinations of components as well as for the whole system. The three-axis model is specifically designed to support such segmentation. At a

minimum, the engineer should generate a suitable 'requirement envelope' and behavioural span for each desired subsystem, including where appropriate any variation in initial condition or external perturbation.

- The aim is to (presumably algorithmically) demonstrate either that the generated behavioural span is bounded by the appropriate operational envelope, or to identify where it is not and identify the relevant substrings corresponding to particular control sequences for further consideration. The only difference is that this process is conducted, perhaps concurrently if computing resources allow, for a multitude of relevant system component combinations.
- Theoretically, this could be every component (i.e. every agent in a multi-agent system) and every grouping of them, up to including the whole system behaviour – the power set of the multi-agent system. In practice, a lot of combinations will be unnecessary, because those particular components or agents could not influence each other, and so consideration of that system combination adds little value. Exactly which combinations of components are relevant is inevitably context-specific, and will be a design choice particular to individual systems.
- The result is a family of so-called '*mini-fractals*' that are (improper) subsets of the overall representation of the system's behaviour, each analysed against the appropriate operational envelope for that combination of components. It would be useful to represent this graphically, but it is not at all clear how to represent subsets of multi-dimensional reachable spans that are potentially 'distorted' (due to complex interactions) from proper subsets of the full system span. In practice of course this should not be a problem since the analysis would be conducted numerically.
- The result is the potentially concurrent use of one approach to analyse the behavioural compliance of the systems at all relevant 'levels' (theoretically, if needed, at *all* 'levels') within the system, based on the three-axis decomposition model. Such an approach should generate evidence for where system failures (the non-compliances) due to complex interactions might occur, and the circumstances that cause them (sequences of control actions given particular starting states), to inform possible mitigations. This is just the sort of evidence needed for assurance, only generated without relying on the fundamental assumptions of a causal link between fault and failures that is invalid for complex systems.

C.5 Recommendations

The recommended next step is to apply the proposed method to a simple multi-agent exemplar problem. The aim of this will be to demonstrate the practical application of the proposed steps and explore some of the issues that may arise. In Annex B, two exemplars were developed based on areas of possible application of multi-agent technology (logistics management and intelligent sensor management), in addition to the exemplar of synchronised production line automata developed in the main body of this thesis. Hierarchical MDP models based on the three-axis decomposition principles have been defined, and it was originally proposed to use one of these as modelling exemplars for this purpose. The opportunity to return to this second phase of work, making use this material or alternative models, should be taken up.

Whether the proposed method is scalable to increasingly large state spaces is a valid concern. As is usual with such problems, the fidelity of the model is compromised if the state space is too course, but the computational burden could make the entire process infeasible if the state space is too fine. Important design choices will have to be made early on, which could have unforeseen consequences on the practicality of the proposed approach. A recommended future activity is to consider the issue of scale and the computational burden of the proposed approach, probably in the context of one of the exemplars noted above.

Due to the focus on the assurance aspects in this annex, 'control' aspects have only been addressed where necessary, this having already been covered in depth in the main body. However, an apparent application for the techniques developed might be to inform adaptive, responsive control in increasingly autonomous systems, i.e. systems detecting and adapting to their own 'failure modes' (undesirable deviations from expected patterns of behaviour). Quite how the proposed method could inform, for example, learning algorithms for intelligent self-adapting systems, might be worth considering further in the future.

Finally, the discussion above also briefly touched on the issue of system reconfiguration, arguably the hardest problem in the whole field of systems engineering, since virtually no part of system engineering theory or practice knows how to cope when the basic building blocks of a system having a recognisable boundary and purpose turn out to be variables. The theoretical basis for this (the so-called '*fractal bifurcation*') has not been considered in depth, but, at first glance, promises much in terms of recognising, and potentially having the ability to adapt to, physical or control level system changes. The potentially transformational nature of such a capability suggests that this possibility is also worth considering in greater depth.

Bibliography

[1] Albert R, Barabasi A-L, *Statistical Mechanics of Complex Networks,* Review of Modern Physics, Volume 74, [2002]

[2] Asimov I, *I, Robot,* Gnome Press, [1950]

[3] Asimov I, *Runaround,* Astounding Science Fiction, [1942]

[4] Barnsley M F, *Fractals Everywhere: Second Edition,* Morgan Kaufman, [1993]

[5] Barnsley M F, *SuperFractals*, Cambridge University Press, [2006]

[6] Barry J, Kaelbling L P, Lazano-Perez T, *Hierarchical Solution of Large Markov Decision Processes*, MIT Computer Science and Artificial Intelligence Laboratory

[7] Becker R, Zilberstein S, Lesser V, *Decentralized Markov Decision Processes with Event-Driven Interactions*, University of Massachusetts

[8] Bellman R, *Dynamic Programming*, Princeton University Press, [1957]

[9] Berger P L, *The Social Construction of Reality*, Penguin Books, [1966]

[10] Beynier A, Mouaddib A-I, *A Polynomial Algorithm for Decentralized Markov Decision Processes with Temporal Constraints,* University of Caen

[11] Bhatia N P, Szegő G P, *Stability Theory of Dynamical Systems*, Classics in Mathematics, Springer-Verlag, [2002 reprint of 1970 edition]

[12] Bishop P, Bloomfield R, *A Methodology for Safety Case Development,* Adelard

[13] Boutilier C, Dean T, Hanks S, *Decision-Theoretic Planning: Structural Assumptions and Computational Leverage*, Journal of Artificial Intelligence Research 11 1-94 [1999]

[14] Bullimore S R, Briggs K M, *Incorporating Tight Colouring into Distributed Heuristics for Wireless Channel Assignment*, [2011]

[15] Cade N A, *Markov Decision Processes for Intelligent Surveillance in Unstructured Environments*, SEAS-DTC Project IF003 Technical Report GSY/060116/109044, [2006]

[16] Cade N, Bray M, Lepley J, *A Language for Intent,* Selex S&AS 1047-S2018, SEAS-DTC Project IF045 Final Report, Issue 1.0, [2008]

[17] Carley K M, Gasser L, *Computational Organization Theory*, Multivalent Systems: A Modern Approach to Distributed Artificial Intelligence pp.299-330, MIT Press, [1999]

[18] Ceglarek D J, *Self-Resilient Production Systems: Closed-Loop Life-Cycle Engineering*

[19] Chu, H-T, Chen, C-C, *On bounding boxes of iterated function system attractors*, *Computers & Graphics* pp. 407–14, [2003]

[20] Corkill D D, Lander S E, *Diversity in Agent Organizations*, Object Magazine 8(4):41-47, [1998]

[21] Deeks C, *Networks of Q-Learners,* MSc Project Report, Centre for Complexity Science, University of Warwick, [2013]

[22] Deeks C, Vitanov I, Williams R, *Mathematical Models for Hierarchical Decision Making*, SEAS-DTC Project SER012 Interim Technical Report, TES101679, [2007]

[23] Deeks C, Vitanov I, Williams R, *Safety Critical Autonomy (Version 2),* SEAS-DTC Project SER012 Technical Report, TES102403, [2008]

[24] Deeks C, Vitanov I, Williams R, *Safety Critical Autonomy: Bounding the Behaviour of Autonomous Systems with Hierarchical Adaptive Controllers*, SEAS-DTC Conference Paper, [2008]

[25] Deeks C, Williams R, *Assurance of Complex and Multi-Agent Systems*, TES106578 [2011]

[26] Deeks C, Williams R, *Inferring Intent from Stimulated Dialogue*, SEAS-DTC Project IF054 Final Report, TES102118, [2008]

[27] Deeks C, Williams R, *Operational Decision Making and Prediction of Intent for Interacting Autonomous Systems*, SEAS-DTC Conference Paper, [2008]

[28] Doltsinis S, Ferreira P, Lohse N, *An MDP Model-Based Reinforcement Learning Approach for Production Station Ramp-Up Optimization: Q-Learning Analysis*, IEEE Transactions on Systems, Man. And Cybernetic Systems, [2014]

[29] Edalat A et al, *Bounding the attractor of an IFS*, Information Processing Letters **64**(4), 197–202, [1997]

[30] Edwards R E, *Functional Analysis: Theory and Applications,* Dover, [1995 reprint of 1965 Holt, Rinehar and Winston edition]

[31] Falconer K, *Fractal Geometry: Mathematical Foundations and Applications*, John Wiley & Sons, Second Edition, [2003]

[32] Gabbai J M E, *Complexity and the Aerospace Industry: understanding Emergence by Relating Structure to Performance using Multi-Agent Systems*, University of Manchester, [2005]

[33] Galbraith J, *Organization Design*, Addison-Wesley, Reading, MA, [1977]

[34] Hauskrecht M, Meuleau N, Kaelbling L P, Dean T, Boutilier C, *Hierarchical Solution of Markov Decision Processes using Micro-Actions*

[35] Hitchins D, *Emergence, Hierarchy, Complexity, Architecture: How do they all fit together? A guide for seekers after enlightenment*, [2008]

[36] Horling B, Lesser V, *A Survey of Multi-Agent Organisational Paradigms*, [2005]

[37] Huth M, Ryan M, *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge University Press, [2000]

[38] Ishida T, Gasser L, Yokoo M, *Organization self design of distributed production systems*, IEEE Transactions on Knowledge and Data Engineering 4(2):123-134, [1992]

[39] Jennings N, *Controlling cooperative problem solving in industrial multi-agent systems*, Artificial Intelligence 75(2):195-240, [1995]

[40] Kappen H J, *Path integrals and symmetry breaking for optimal control theory*, arXiv: physics /0505066 4, [2005]

[41] Kelly T, Weaver R, *The Goal Structuring Notation – A Safety Argument Notation*, Department of Computer Science and Department of Management Studies University of York, York, YO10 5DD UK

[42] Kratovil E W, Barnes R, Ericson C, *The Safety of Unmanned Systems: The Process Used to Develop Safety Precepts for Unmanned Systems*, Unmanned Systems Safety Guide for DoD Acquisition, Version 1.0

[43] Lane T, Kaelbling L P, *Nearly Deterministic Abstractions of Markov Decision Processes*, MIT Artificial Intelligence Laboratories, AAAI-02 Proceedings, [2002]

[44] Leigh J R, *Functional Analysis and Linear Control Theory*, Dover, [2007 reprint of 1980 Academic Press edition]

[45] Lesser V, *Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture*, Autonomous Agents and Multi-Agent Systems 1:89-111, [1998]

[46] Lightfoot T J, Milne G J, *Modelling Emergent Crowd Behaviour*, School of Computer Science and Software Engineering, University of Western Australia

[47] Meyer B, *Object Oriented Software Construction*, Prentice-Hall, [1997]

[48] Mount D M, *Computational Geometry*, Department of Computer Science, University of Maryland, [2002]

[49] Newman M E J, *Modularity and Community Structure in Networks*, Department of Physics and Center for the Study of Complex Systems, Randall Laboratory, University of Michigan, arXiv: physics /0602124, [2006]

[50] Okuda Y, Walsh M, *Measuring Emergence in Cooperative Autonomy*, SEAS-DTC Project SER019 Final Report, TES105495, [2009]

[51] Okuda Y, Walsh M, *Review of Existing Approached to the Prediction and Management of Emergence*, BAE Systems, TES107609, [2011]

[52] Onnela J P et al., *Structure and tie strengths in mobile communication networks*, Oxford University

[53] Ott E, *Chaos in Dynamical Systems*, Cambridge University Press, [2002]

[54] Parr R, Russell S, *Reinforcement Learning with Hierarchies of Machines*, Computer Science Division, UC Berkeley

[55] Pineau J, Thrun S, *An integrated approach to hierarchy and abstraction for POMDPs*, Carnegie Mellon University School of Computer Science, CMU-RI-TR-02-21, [2002]

[56] Puterman M L, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Series in Probability and Statistics, [2005]

[57] Roscoe A, *The Theory and Practice of Concurrency*, Prentice-Hall series in Computer Science, Prentice-Hall, [1998]

[58] Smith R G, *The contract net protocol: High-level communication and control in a distributed problem solver*, IEEE Transactions on Computers 29(12):1104-1113 [1980]

[59] Sutton R S, Barto A G, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge Massachusetts, [1998]

[60] Sutton R S, Singh S, *Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning*

[61] The Royal Academy of Engineering, *Creating Systems That Work: Principles of Engineering Systems for the 21st Century*

[62] Theocharus G, Murphy K, Kaelbling L P, *Representing hierarchical POMDPs as DBNs for multi-scale robot localization*, MIT Computer Science and Artificial Intelligence Laboratory

[63] Thoms J, *Understanding Systems Behaviour*, SEAS-DTC Project MP016 Technical Report, [2010]

[64] Thrun S, Burgard W, Fox D, *Probabilistic Robotics*, Intelligent Robotics and Autonomous Agents series, The MIT Press, [2005]

[65] Van Dyke Parunak H, Brueckner S, *Entropy and Self-Organization in Systems*, International Conference on Autonomous Agents (Agents 2001), [2001]

[66] Vorobeychuk Y, Mayo J R, Armstrong R C, Ruthruff J R, *Noncooperatively Optimized Tolerance: Decentralized Strategic Optimization in Complex Systems*, Sandia National Laboratories

[67] Watkins CJCH, *Learning from Delayed Rewards*, PhD Thesis, Cambridge University (King's College), [1989]

[68] Williams R, *Novel Certification: Final Report*, SEAS-DTC Project IF004 Technical Report GSY/060136/109042, [2006]

[69] Xiao Y, *Random Fractals and Markov Processes*, Proceedings of Symposia in Pure Mathematics, [2004]

[70] Yu S B, Efstathiou J, *An Introduction to Network Complexity*, Manufacturing Research Group, University of Oxford