HOMEOSTATIC ACTION SELECTION FOR SIMULTANEOUS MULTI-TASKING

David Andrew Symons

Homeostatic Action Selection for Simultaneous Multi-tasking

David Andrew Symons

University of St Andrews

This thesis is submitted in partial fulfilment for the degree of

Doctor of Philosophy (PhD)

at the University of St Andrews

August 2019

Abstract

Mobile robots are rapidly developing and gaining in competence, but the potential of available hardware still far outstrips our ability to harness. Domain-specific applications are most successful due to customised programming tailored to a narrow area of application. Resulting systems lack extensibility and autonomy, leading to increased cost of development.

This thesis investigates the possibility of designing and implementing a general framework capable of simultaneously coordinating multiple tasks that can be added or removed in a plug and play manner. A homeostatic mechanism is proposed for resolving the contentions inevitably arising between tasks competing for the use of the same robot actuators.

In order to evaluate the developed system, demonstrator tasks are constructed to reach a goal location, prevent collision, follow a contour around obstacles and balance a ball within a spherical bowl atop the robot.

Experiments show preliminary success with the homeostatic coordination mechanism but a restriction to local search causes issues that preclude conclusive evaluation. Future work identifies avenues for further research and suggests switching to a planner with the sufficient foresight to continue evaluation.

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/K503162/1].

Declaration

Candidate's Declarations

I, David Andrew Symons, do hereby certify that this thesis, submitted for the degree of PhD, which is approximately 52,000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree.

I was admitted as a research student at the University of St Andrews in September 2012.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

Date: 10.06.2020

Signature of candidate:

Supervisor's Declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date:

Signature of supervisor:

Underpinning Research Data or Digital Outputs

Candidate's Declaration

I, David Andrew Symons, hereby certify that no requirements to deposit original research data or digital outputs apply to this thesis and that, where appropriate, secondary data used have been referenced in the full text of my thesis.

Date: 10.06.2020

Signature of candidate:

Contents

List of Figures

List of Tables

ACRONYMS

AI Artificial Intelligence

ASM Action Selection Mechanism

ASP Action Selection Problem

CPU Central Processing Unit

DES Discrete Event System

FSA Finite State Automaton

HMI Homeostatic Mortality Index

ID In-schedule Dependencies

MP Multi-phase

MR Multi-robot

MRTA Multi-robot Task Allocation

MT Multi-task

MTC Minimum Turning Circle

ODE Open Dynamics Engine

ROS Robot Operating System

SMT Simultaneous Multi-tasking

SP Single-phase

SPA Sense-Plan-Act

SR Single-robot

ST Single-task

CHAPTER ONE

INTRODUCTION

This chapter provides an entry point to, and overview of, this dissertation. It states the problem at hand and the motivations for solving it. A specific research question is formulated before discussing the scope of this piece of work and the hypotheses made at the outset. Outcomes and contributions are listed and finally the structure of the following chapters is laid out.

1.1 Problem Statement

1.1.1 Unrealised Hardware Potential

The field of robotics is enjoying ever-increasing popularity due to its many applications ranging from space exploration to home aids as trivial as automated vacuum cleaners. Robot hardware has improved immensely over the years, especially due to industrial interest. Mass production and 3D printing of components has made robots affordable and ubiquitous. Humanoids with twenty or more degrees of freedom (number of actuators, such as joints, that can be independently controlled) are readily available and have the potential to perform almost any task imaginable.

The problem is that we cannot yet orchestrate all these actuators to produce the desired behaviour and harness the power of the available hardware. The difficulty lies in finding a sequence of controls or commands that make the many small motors come to life and coordinate to perform a useful task.

This section examines why we are still far from reaching the theoretical limits of what could be accomplished with the tools we have built. To do so it addresses the gap between *perceived* and *actual* robot competence and gives examples of the state of the art in adaptive AI controllers. A list of remaining challenges leads to a description of the chosen research gap.

1.1.2 The Scripting Illusion

Robots are being perceived as increasingly capable and even autonomous. While this impression is partially justified by genuine advances in the field, it is also contributed to by the illusion of competence created by scripted robots. Dancing humanoids, as featured in this 2017 Guinness World Record [117], are common culprits of creating this deception. The observer is led to believe that the robot is in some way conscious of its actions or is at least planning its moves autonomously. Some robots are even made to look as human as possible, thereby encouraging one to relate to the machine and credit it with human attributes [78].

However, just like in Searle's Chinese room argument [111], a look behind the scenes immediately destroys the illusion. The seemingly impressive behaviours are simply the result of the machine meticulously following a script or macro, which is a detailed sequence of instructions that it is hard-wired to execute. The benefit of this approach is that it allows the full potential of the robot's hardware to be utilised. The impressive results that can be achieved are not proof of the machine's intelligence, but rather attest to the insight and perseverance of the programmer who underwent the arduous task of writing the script.

In spite of their lacking autonomy, scripted robots are undeniably useful. For instance, assembly line robots are often programmed in this way to great effect. Their rigid adherence to the instructions they are given allows them to perform high precision tasks in a short amount of time.

The same rigidity unfortunately also prevents them from reacting to changes in their surroundings. This has a number of disadvantages, one of which is that the robot will keep going even if someone gets in its way. A factory worker who was stabbed and electrocuted by the arm of a welding robot in 2015 [99] is an example of how dangerous this can be. As a precaution "robots are generally kept in cages to prevent contact with workers" [99], but although this is clearly a sensible safety measure, it does limit the use of such robots to a confined setting.

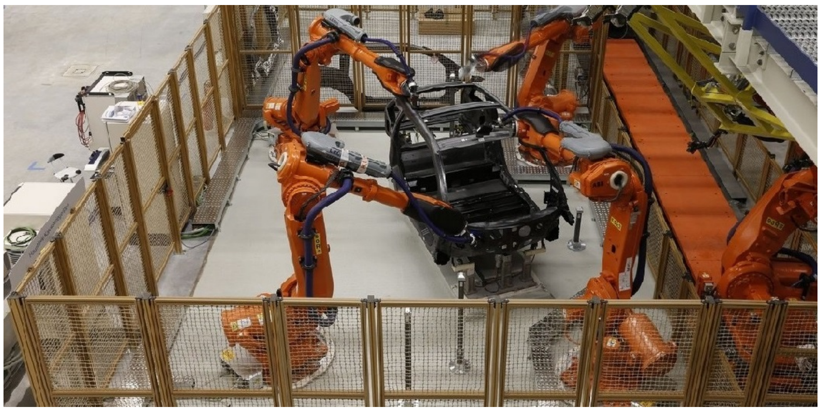


Figure 1.1: Factory robots caged for safety reasons (image taken from [37])

A further drawback of the inability to react or adapt to the environment is that even simple changes to the requirements may call for substantial changes to the script. In the example of dancing robots, this could mean rewriting a script just because the same dance is to be performed on a different surface. Instructions that work when the robot is standing on carpet might cause it to slip and fall on smoother surfaces. Needless to say, this is the antithesis to reusability.

1.1.3 The State of the Art

Prospects for Artificial Intelligence (AI) are not as bleak as the prevalence of scripting may suggest. Adaptive controllers are being developed to take advantage of the also growing range and accuracy of sensors. Capturing information about the environment allows controllers to implement a sensory motor feedback loop to calibrate their actions and adapt to unforeseen events.

Solutions in the field of mobile robotics most commonly revolved around navigation and obstacle avoidance, with notable inventions including self-driving cars [29] and an array of robots with potential military applications developed by Boston Dynamics [17].

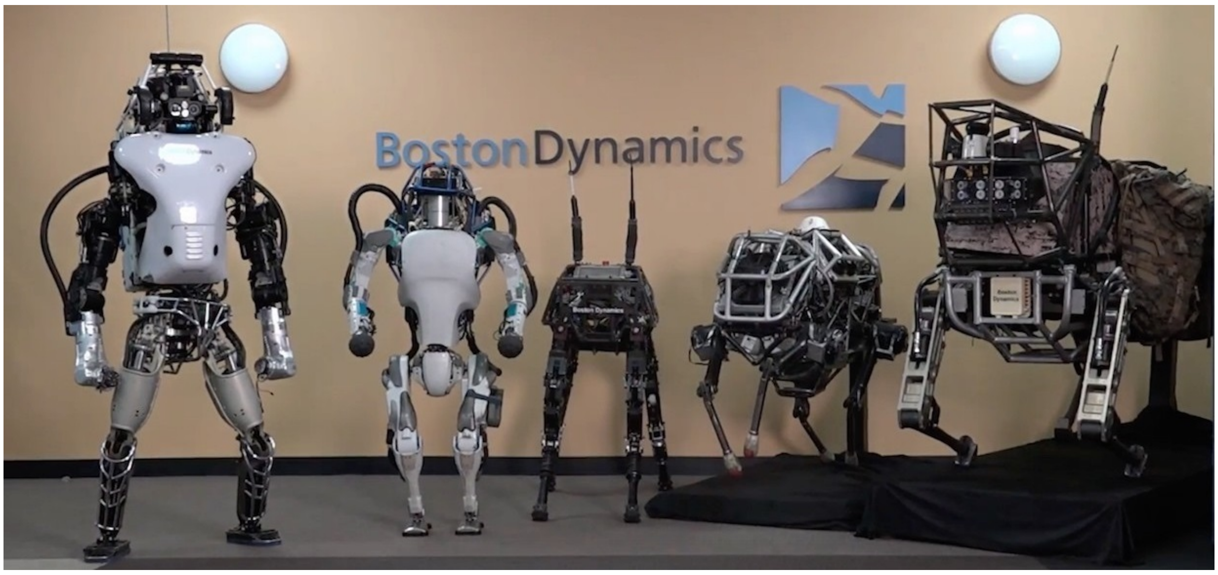


Figure 1.2: Humanoid and quadruped robots developed by Boston Dynamics (image taken from [17])

Of course, these are examples on the cutting edge of technology, and unsurprisingly neither companies nor military are particularly forthcoming with their insights. Even with the knowledge and resources available to these organisations, human intervention is still required. This goes for self-driving cars [16] as well as the Boston Dynamics robots, which, according to Priday [93], rely on human remote control more than the videos would have you believe. So while adaptive control systems, unlike their scripted counterparts, allow robots to be unleashed in the real world, their command of the hardware they operate is still limited.

1.1.4 Remaining Challenges

The following problems continue to present some significant difficulty and are frequently encountered in the development of AI controllers. Issues in adjacent fields of research such as computer vision, active sensing, odometry, speech recognition etc. are also likely to crop up but are not a subject of this thesis and therefore omitted from the list.

1. Given the almost endless possibilities, any reasonably specific task is unlikely to have an existing solution and may require expert knowledge to solve. Even more frustratingly, a solution may exist but not be publicly available. In either case one can expect to invest a significant amount of time and money in the implementation of the required features.

2. The complexity of an existing algorithm might not permit real-time execution. Movement speed has to be severely limited to prevent the robot getting ahead of the planner. In some cases the vehicle may actually need to stop and wait for new controls. Off-line (or pre-) computation of the entire control sequence undermines the ability to adapt to unforeseen events and requires a complete model of the necessarily static environment.

3. Constraints on the robot's motion abilities are not considered by the controller and result in plans that cannot be realised. Simplistic planners may, for example, model the robot as a particle that can move in any direction only to discover during execution that a car cannot move sideways. A more advanced but no less common problem is the AI neglecting to cater for the minimum turning circle of a vehicle getting larger with increasing speed.

4. Solutions to individual tasks can seldom be merged to elicit their combined behaviours. Being able to set a robot multiple tasks simultaneously can, however, be very desirable. It would, for instance, allow a robot waiter to navigate a restaurant and avoid collisions while also balancing a tray. Given solutions to each of the three tasks involved, one may hope to achieve all desired behaviours with little additional effort. Attempts in this direction will, however, be foiled by disparities between the controls best suited to progressing the different tasks individually. An action that benefits one task may undermine another, thus creating a conflict of interest that has competing behaviours vying for control over the robot. With no compromise in sight, the only option is to build a new controller specifically for the selected combination of tasks. Often, complex formulae and human insight are required to resolve contentions and achieve coordination.

1.1.5 Research Gap

As suggested by its title, this thesis is primarily interested in task coordination. The focus is therefore on the last of the aforementioned challenges, although all of them have to be addressed to some extent. Specifically, the research gap lies in overcoming the problem of deciding the relative importance of the tasks being coordinated by a *single* robot and doing so in a task-agnostic manner. The distribution of tasks among teams of collaborative robots is a different form of coordination not considered here. Existing work on task coordination can be categorised into interleaved-, parallel- and simultaneous multi-tasking (see Action Selection Taxonomy).

Interleaved multi-tasking comprises all techniques that give different tasks alternating control of the robot. Brooks' Subsumption Architecture [18] is a foundational contribution to this area. Improvements to his rule-driven approach include variations using reinforcement learning [77][55], behaviour graphs [84][28] and scheduling [100][47]. Simultaneous execution of multiple tasks cannot, however, be achieved by any of these inherently sequential methods.

Parallel multi-tasking allows for concurrent execution of *independent* actions not involving the same robot actuators. Rather than giving a task exclusive control of the whole robot, the task is given control of individual manipulators. Those not required by one task may be controlled by another. Different approaches exist to resolving dependencies and isolating non-conflicting operations that can be parallelised in this manner. For example, Lenser [76] proposes a graph based mechanism. Taipalus [119] casts the issue as a resource allocation problem, while Towle and Nicolescu [121] use an auction based system. Despite improving the capacity for concurrent hardware utilisation, *dependent* tasks are still unable to execute simultaneously due to the absence of a conflict resolution mechanism. The following category deals with tasks contesting the same resource, where interference cannot be resolved by decomposition into serial or parallel actions.

Simultaneous multi-tasking is the main category of interest and contains the only existing techniques dedicated to resolving conflict in order to progress all tasks at the same time with the same action. Khatib [66] and Arkin [1] suggest finding the ideal action for each individual task and then combining those to form an overall solution. The loss of information incurred by discarding everything except a task's highest preference can lead to compromises with which none of the tasks is satisfied. This problem has been addressed by considering multiple alternative actions which are rated by the competing tasks. Rosenblatt [101], Riekkie [98] and Benjamin [9] have made notable contributions to this approach. A shared, fundamental difficulty lies in determining the relative importance of different tasks and thereby the influence they have on action selection [41] [38]. Rather than trying to solve the prioritisation problem, this thesis proposes a way of avoiding it entirely.

Another research gap exists in the limited reusability of available coordination systems. Early attempts at multi-tasking were restricted to handling a fixed set of tasks. Since then, interest in constructing more general plug and play architectures has grown. Arkin and Khatib appear to have been most successful in doing so, but their solutions have been superseded by designs based on rating alternative actions. While these improve on coordination quality, they sacrifice reusability due to their reliance on relative priorities that are once again specific to each set of tasks. Adding to or removing from that set requires a complete re-evaluation of the hierarchy of its members. Finding an alternative to using relative priorities facilitates the development of a more general, task-agnostic coordination framework.

1.2 Motivation

The description of the stated problems already hints at what may be gained by solving them.

Primarily, this thesis aims to improve overall robot competence by extending the capacity for handling multiple conflicting tasks. It is hoped that the same coordination mechanism will eventually allow robots to tap into the full power of today's hardware due to a reduction in unnecessary idle time.

Increased autonomy and the ability to react to sensor information enables robots to adapt their actions appropriately. In this way, their movement becomes safe and predictable, reducing the need for robot cages and allowing people to interact with them.

The development of a reusable coordination mechanism should save having to build custom software for coordinating each new set of tasks. This cuts down on development time and cost.

Finally, the work started here may enable future research. Further progress may be facilitated and encouraged by establishing a common ground and basis for discussion.

1.3 Research Question

Given the remaining problems in the chosen field of research, along with a desire to focus on task coordination, we arrive at the following research question:

Can a general, task-agnostic framework be devised to coordinate multiple tasks being simultaneously executed by a single robot through homeostatic conflict resolution?

The aim is to construct a framework into which existing solutions to individual tasks can be integrated to facilitate their coordination. Integration may require existing mechanisms to be adapted, but doing so is still preferable to the alternative of creating custom solutions for the far greater number of combinations of those tasks. Focusing on coordination in the abstract, rather than on planning individual tasks, also has the advantage of being able to abstract over task-specific details. These include design decisions such as whether a particular task should be programmed to operate in static or dynamic environments.

The two following sections describe the scope of the investigation into the research question and the hypotheses made in an attempt to meet the goals laid out in the motivation.

1.4 Scope

Although most technical definitions are confined to the upcoming content chapters, a clear understanding of certain terms is required to delimit the scope of this thesis. To this end, the following explains how key words and concepts are to be interpreted in the context of mobile robotics and specifically task coordination. Keeping this project within reasonable bounds also involves specifying what can reasonably be expected of the solutions developed.

1.4.1 What is Simultaneous Multi-tasking?

Intuitively, one may assume that multi-tasking is simultaneous by nature, but in fact different types can be distinguished. As should already have transpired from the description of the research gap, the presence or absence of interdependencies determines how a coordination problem needs to be tackled.

Independent tasks that do not undermine each other require no special coordination mechanism. This is only essential when tasks interact or compete. Conflict arises due to dependent tasks trying to manipulate the same actuator(s). More precisely, the problem occurs when competing tasks concurrently attempt to change the same state variable in divergent ways.

Dependencies can sometimes be disentangled by identifying independent actions that can be executed in parallel, but this approach has its limits. To resolve conflict and make sustainable progress, each action must simultaneously consider and progress (or at least not impede) all tasks. Simultaneous Multi-tasking (SMT) is the truly concurrent coordination of multiple dependent tasks with a single action. Proverbially speaking, SMT captures the notion of “killing two birds with one stone”.

Of course, such solutions only exist when tasks allow room for compromise and are not inherently opposed. If there is no common ground at all, one cannot hope to find a satisfactory solution.

1.4.2 What is a Task?

Having established what is meant by SMT, the next step towards defining the scope of the envisaged coordination system is to specify the types of tasks for which it is intended. Although ostensibly benign, the word *task* can make it surprisingly difficult to specify realisable objectives. To curtail its vague everyday meaning, a taxonomy of tasks eligible for coordination is presented as part of the framework in Chapter 3. A brief overview of the categories it defines is given here after situating it within existing classifications.

Taxonomies of robot tasks are often geared towards specific domains. Quispe et al. [95] present a taxonomy of benchmark tasks for bimanual manipulators. A slightly more general classification of tasks involving robotic manipulators is attempted by Leidner et al. [75], but focuses on interactions with external objects. The robot's internal state seems not to be considered and there is no mention of classical mobile robotics tasks such as navigation. A very specific example is fruit picking in an agricultural setting [14]. None of these taxonomies covers a wide enough spectrum to be of aid in defining tasks for a general coordination system that aims to abstract over both robot hardware and the domain of application.

A general and widely recognised taxonomy of Multi-robot Task Allocation (MRTA) problems is that of Gerkey and Matarić [44]. Although seemingly intended for a Multi-robot (MR) setting, their taxonomy also includes a category for Single-robot (SR) tasks into which this PhD fits. More importantly, it makes a distinction between Single-task (ST) and Multi-task (MT) robots. Using the suggested notation, the intersection of the two relevant categories is denoted MT-SR. This is not as close of a match as one may initially think however, owing to the fact that tasks are assumed to be independent. In other words, Gerkey and Matarić's taxonomy is suitable for Parallel multi-tasking, but not for Simultaneous multi-tasking.

The significance of this deficit is pointed out by Korsah et al. [69] who set out to extend the taxonomy to include different types of dependencies. In-schedule Dependencies (ID) come closest to describing conflict between concurrent tasks. The most appropriate category in the extended taxonomy is therefore the one denoted "ID [MT-SR-IA]" [69, p. 1504]. IA stands for "instantaneous assignment" and signifies that the robot is allocated all of its task at the outset, rather than in a "time-extended assignment" (TA) process. This distinction is of little significance here, as we are not concerned with the task assignment problem, but with the coordination of tasks that have already been assigned.

The difference between task assignment and action selection problems is brought out by the taxonomy of tasks presented in this thesis. In addition to delimiting its scope, it also provides a way of further distinguishing between tasks in an otherwise still quite broad class of problems.

Multi-phase Tasks are of an inherently sequential nature, with multiple steps required to attain the desired solution. A detailed description of this category is given in Section 3.1.1, where the task of stacking boxes is used an example.

An even simpler case would be waving a robotic arm from side to side. Each change in direction marks the end of one phase and the beginning of the next as the target position of the arm changes. In this context, conflict comes in the form of scheduling constraints, such as not being able to move the arm left and right in the same phase. This is a task allocation or planning problem and does not involve the type of conflict this thesis takes an interest in. Here, conflict is that arising between tasks that have already been assigned and are competing in the same phase. Even simple actions, such as moving the robot's arm are *not* hard coded behaviours, but single-phase tasks defined only by a target state. This gives the robot the flexibility to adjust solutions as needed to accommodate for other tasks.

The idea is to develop a solution to single-phase task coordination which can be applied repeatedly to also solve multi-phase tasks. An external task allocation mechanism will have to be used for decomposing larger problems into multiple phases. A plethora of contributions on this subject have already been published (see [112, p. 1359] for an overview of approaches to MRTA problems). For this reason, and because everything hinges on being able to solve single-phase tasks, the priority must be to develop a system capable of SMT.

Single-phase Tasks can be seen as the building blocks of multi-phase tasks. Most of these are described in terms of the robot's internal parameters. Together, these parameters form what is referred to as a robot's *configuration*. More precisely what is meant is a collection of variables, a full assignment to which fixes all of the robot's changing aspects at a specific point in time. Simply put, it is a snapshot of the robot that captures information such as its location, pose, orientation, remaining battery life, etc. Tasks may also involve environmental parameters which are part of the external state. The categories outlined in the following cover tasks involving both internal and external state variables. A more detailed description and further examples can be found in Section 3.1.2.

Category 1 : Configuration Transition Tasks require the robot to change the parameters of its current configuration to those specified by a given target configuration. Typical examples are moving to a different location or for a humanoid robot to change posture, e.g. from sitting to standing.

Category 2 : Environment Manipulation Tasks are defined in terms of the robot's external state i.e. the world it inhabits. Such tasks may include moving an object, pressing a switch or kicking a ball.

Category 3 : Configuration Avoidance Tasks are necessary to prevent the robot from assuming undesirable states or poses. A limbed robot could, for instance, lose its balance if its arms and legs were not properly coordinated.

Category 4 : Situation Avoidance Tasks describe dangerous or unwanted relations between the robot and the environment. In the example of obstacle navigation, the robot is kept at a safe distance to obstacles in its path.

Any task that fits one of these categories and fulfils the requirements of the task-agnostic framework is suitable for coordination. Multi-phase tasks also qualify, provided they can be decomposed into single-phase tasks using an external task allocation mechanism. Tasks involving multiple robots do not fit into any of these categories are outwith the scope of this thesis.

1.4.3 Demonstrator Tasks

The performance of the developed system is evaluated in Chapter 7 using a number of different demonstrator tasks. Setting up the required tests involved solving the chosen tasks in a way that is compatible with the proposed task coordination framework. Developing these solutions took longer than originally anticipated. Some task solutions could be adapted form existing ones, while others were constructed from first principles. Despite also making attempts to provide quality solutions for individual tasks, their role is secondary to that of the coordination system which they were designed to verify.

To ensure good test coverage, demonstrator tasks were selected from different categories of the tasks taxonomy. Travelling to a goal location and navigating obstacles are staples of mobile robotics and have to be represented to show that the developed system can coordinate classic benchmark tasks. The goal location task developed in chapter four is representative of category 1. There is intrinsic value in attaining a target location and the robot's position is part of its internal state, i.e. configuration. Contrarily, Obstacle navigation has extrinsic value and involves external state, thus placing it in category 4. Ball balancing showcases the ability to solve category 3 tasks, while also taking on a less studied problem to demonstrate the generality of the developed task coordination system.

Category 2 is the only one not represented. This is because environment manipulation tasks require sensor information to be processed to form a high level semantic understanding of the environment. Recognising objects the robot can interact with is a different field of study and beyond the scope of this thesis.

1.4.4 Proof of Concept

The main aim of this dissertation is to show that multiple tasks can be coordinated in real-time by a generic and reusable AI system relying only on sensor information. The developed theory and presented solutions represent a proof of concept and make no claim to perfection. Being able to generate realisable paths under these conditions, requires concessions to be made. Optimality in particular is a property that is difficult to guarantee when only very little time is available to decide the next action. So although the system makes best efforts, it cannot be relied upon to find the ideal compromise between the conflicting needs of all tasks being coordinated. In fact, it is often difficult to determine, even in retrospect, which controls would have been optimal based on the information available to the robot at the time they were selected.

1.5 Hypotheses

By way of answering the research question, a number of hypotheses are investigated. The assumptions made here are thought to be realistic and constructive in forming the basis for a solution to the stated problem. A satisfactory answer to the research question rests on the truth of these assumptions. Successful vindication of the hypotheses is hoped to inform the development of a generic task coordination system and incite confidence in its validity.

Hypothesis 1 : The chosen definition of *task* is broad enough to be meaningful yet narrow enough for coordination to be feasible.
In order for the task coordination system to be of use and deserving of the term *generic*, a reasonable number and variety of individual tasks must fit the definition given in Chapter 3. If too few tasks are eligible to be coordinated we end up with yet another specialised system. If the definition is too broad or vague, we cannot make enough useful assumptions about the tasks and thereby lose the basis for reasoning about them – at least without falling back on purely symbolic reasoning and drifting off into more abstract fields such as constraint programming where solutions generally suffer from exponential complexities and are ill suited to real-time planning.

Hypothesis 2: A generic task coordination framework can be designed to only rely on state prediction and urgency heuristics.
The proposed framework provides a layer of abstraction that allows tasks to be reasoned about without knowing their particulars. In other words, it is like an interface that defines only the most fundamental properties required for coordination. These two requirements will be discussed throughout my thesis with explicit examples given in the task chapters 4,

5 and 6. In brief, the first requirement is the ability to map controls to state variables and the second is to map those variables to an indication of how well a task is under control. While this is believed to be the bare minimum to make coordination viable, enough tasks must be able to meet these requirements for the system to be generic and reusable.

Hypothesis 3: Homeostasis is a suitable principle to use as the foundation for a task coordination system.

This is the central hypothesis on which the design of the proposed coordination mechanism rests. The assumption is that homeostasis, the attention to threat principle, can be used to resolve conflict between multiple competing tasks and thereby facilitate their simultaneous progression and solution. Chapters 3 and 7 are devoted to investigating this claim theoretically and empirically in order to answer the research question.

Hypothesis 4: Existing solutions can be found in real-time using only sensor information and local search.

While a multi-step planner considers a sequence of controls leading all the way to the goal, a single-step planner bases its decisions on the more immediate future. The disadvantage of local search is that it may not find solutions requiring foresight, and the solutions it does find can typically not be guaranteed to be optimal. However, the higher precision of multi-step planners comes at the cost of computational complexity and this can quickly become forbidding, especially when multiple tasks are involved. Given the necessity of reacting to sensor information in real-time, a single-step solution is attempted. The question of its sufficiency is addressed in Chapter 7.

1.6 Outcomes and Contributions

This thesis presents five contributions relating to task coordination, the individual tasks being coordinated and the system built to evaluate the proposed homeostatic action selection mechanism.

Contribution 1 : A new taxonomy of action selection mechanisms

The Action Selection Taxonomy presented as part of Chapter 2 categorises existing task coordination mechanisms into interleaved-, parallel- and simultaneous multi-tasking. Sub-categories of these further distinguish between the different techniques employed.

Contribution 2 : A general, task-agnostic framework for simultaneous multi-tasking

The framework consists of a taxonomy of tasks eligible for coordination together with an interface that abstracts over the details of those tasks. The plug and play architecture is built on the abstraction provided by this interface, which captures the minimum requirements for conflict resolution to be possible.

Contribution 3 : A homeostatic conflict resolution mechanism
The proposed Homeostatic Mortality Reduction mechanism rates and compares the merit of candidate actions on the fair, universal scale it defines. Problems connected to determining relative task priorities or weights are avoided, thus facilitating the selection of compromise controls capable of balancing the needs of competing tasks.

Contribution 4 : A testbed for evaluating the constructed task coordination mechanism
The testbed comprises a robot simulator and solutions to a number of individual tasks to act as demonstrators for task coordination. The classical robotics tasks of reaching a goal location and navigating obstacles are among those adapted for compatibility with the task-agnostic framework.

Contribution 5 : The introduction of ball balancing as a novel demonstrator task
Ball balancing was devised as an example of a non-standard task eligible for coordination. A physics model is developed to simulate and predict the behaviour of a ball rolling in a spherical bowl mounted on a moving vehicle. By minimising the ball's energy, an AI controller can counteract external forces to keep the ball balanced and contained within its bowl.

1.7 Thesis Structure

The following gives a brief summary of each chapter's contents and links them together by outlining the logical structure into which they fit.

Chapter 2 introduces the reader to task coordination and related literature. After defining key terms, literature on *action selection* is presented in the form of a new taxonomy that distinguishes between the available mechanisms based on the type of multi-tasking they support. Homeostasis is reviewed as the basis for the proposed task coordination system.

Chapter 3 addresses the research question by designing a general framework for task coordination. The requirements for the desired system are laid out and the tasks it should be able to handle are classified into four well-defined categories. Finally, a homeostatic control system is developed following the structure prescribed by the framework.

Chapter 4 concerns itself with the task of reaching a goal location by generating a realisable path the robot can follow to its destination. This is the first of four demonstrator tasks used to evaluate the coordination system. For compliance with the framework and to satisfy the second hypothesis, the forward kinematics for a differential drive robot are cited and an urgency heuristic is developed that draws the robot towards its goal.

Chapter 5 provides a solution to obstacle navigation or more precisely the task of following a contour around an obstacle at a safe distance. Ideas from existing solutions are combined with results from Chapter 4 to guide the robot steadily and gracefully around obstacles in its way. The desired behaviour is achieved by producing heuristics for two separate tasks: maintaining a safe distance to obstacles and smoothly circumnavigating them.

Chapter 6 develops the novel task of balancing a ball in a shallow bowl atop a moving robot. This more versatile alternative to the inverted pendulum problem was designed to demonstrate the ability to keep balance both literally and in the more abstract sense of balancing the needs of competing tasks. A specially created physics model fulfils the coordination framework's requirement for state prediction. The ball is kept inside the bowl by minimising an energy based heuristic.

Chapter 7 conducts experiments using the tasks developed in the preceding chapters to evaluate the homeostatic coordination system. Results are recorded and displayed using a custom-built robot simulator, the design of which is described. Experiments are set up to collect data for different combinations of tasks. The problems become progressively harder until all demonstrator tasks are coordinated simultaneously.

Chapter 8 concludes this thesis with an evaluation of the work undertaken. It assesses the feasibility of using homeostasis as the guiding principle for conflict resolution and task coordination. Solutions to the demonstrator tasks are also discussed. Finally, the dissertation culminates in a section on future work.

Chapter Two

Background and Action Selection Taxonomy

This chapter provides a literary background on task coordination and techniques relevant to achieving it. The review covers existing work pertaining to different types of multi-tasking as well as principles on which successful conflict resolution may be based. The context survey begins by placing this work within the literature and addresses inconsistent use of terminology that may lead to confusion or incorrect classification. Following placement into the field of action selection, a more precise classification within that area is achieved by providing a taxonomy of different types of Action Selection Mechanisms (ASMs). Each of the major categories in the taxonomy is examined together with a review of selected contributions that fall into interleaved-, parallel- and simultaneous multi-tasking. A review of homeostasis, its relevance to robotics and an outline of how this *attention to threat principle* can facilitate SMT completes the chapter.

2.1 Placement

The research gap and corresponding challenges have already been outlined in the introduction. Solutions to the type of problem described fall into the field of *behaviour-based control* and typically rely on a closed-loop control system. A basic form of such a system is known as the Sense-Plan-Act (SPA) architecture and involves maintaining a centralised representation of the world by processing incoming sensor information. The accumulated knowledge is used to form a plan which, when put into action, changes the environment and with it the sensor readings – thereby closing the loop.

When, as in this case, multiple tasks are involved, a decentralised model known as the *behaviour-based architecture* is more appropriate. This architecture also implements a feedback loop akin to that used in SPA, except that it requires a separate planning module for each task. These modules independently produce plans most suited to the behaviours they represent, thus giving rise to the problem of having to combine the often diverging plans to select a mutually beneficial action [9, p. 2]. This is known as the Action Selection Problem (ASP) and is the primary field in which this thesis is situated.

Similar works sometimes mention a related field called *multi-objective optimisation*. But while there are clearly multiple-objectives involved in multi-tasking, the optimisation aspect should not be emphasised in behavioural robotics where it is often hard to even define optimality, let alone achieve or prove it [112, p. 313].

In summary, the area this PhD is concerned with is the intersection of behaviour-based control and action selection with some aspects of multi-objective optimisation. Despite giving name to a whole field of research, the definition of *behaviour* is surprisingly elusive and can be seen to vary significantly between different works. Behaviours range from simple reflexes (if-then interactions) to forming increasingly sophisticated long term plans. A number of additional terms have been introduced in an attempt to specify more precisely the complexity of the behaviours to be elicited from the robot. Such terms include job, project, goal, mission, task, etc. but are rarely defined or consistently used. This thesis uses the term *task*, a definition of which has already been outlined in the introduction to delimit the scope and pre-empt any misconceptions. A full exposition of the four types of task can be found in 3.1.

Following through with the chosen task-centric terminology also means talking about task coordination or multi-tasking rather than behaviour coordination or action selection. In any case, the term *action selection* does not accurately describe more recent approaches to coordination: While earlier approaches did select from a given set of available actions as the term implies, today's methods tend to synthesize, construct or search for a compromise action rather than selecting from a pool of candidates. For this reason, action selection is only part of the overall taxonomy of task coordination methods presented in the following section.

2.2 Action Selection Taxonomy

A number of taxonomies for categorising action selection problems and mechanisms have been proposed, most notably by MacKenzie [83], Saffiotti [105], Pirjanian [92] and Benjamin [9], with the latter being the most recent, despite dating back to 2002.

Here, a new taxonomy is presented from the perspective of multi-tasking. It combines and reorganises categories of the existing taxonomies, some of which focus only on one specific branch of development, but also adds more recent contributions. Reorganisation and label changes are justified as each category is discussed.

Three major categories distinguish between different types, or interpretations, of multi-tasking while their sub-categories divide ASMs based on the techniques they employ. Excluded from this taxonomy is only the very limited notion of multi-tasking sometimes used to describe multi-purpose hardware. In that interpretation, multi-tasking capabilities may be ascribed to a robot that can, for example, switch between different end effectors or tools. Due to the (sometimes even manual) switching process, the tasks that can be performed are mutually exclusive, meaning there is no coordination involved as there is no conflict to be resolved.

The structure of this literature review follows the visual representation of the new taxonomy shown below. Contributions to each of the major categories of interleaved-, parallel- and simultaneous multi-tasking are examined in the following.

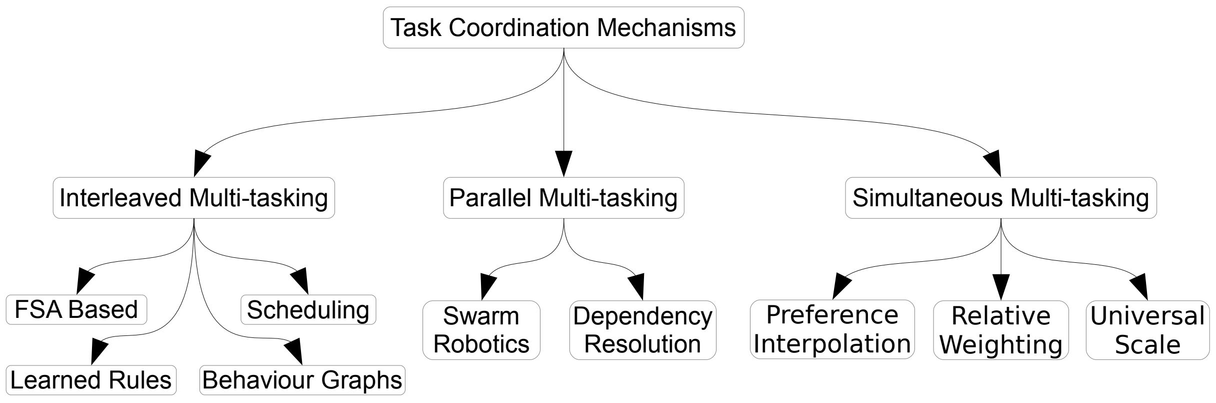


Figure 2.1: Taxonomy of action selection mechanisms for the coordination of multiple tasks on a single robot

2.2.1 Interleaved Multi-tasking

This category includes all action selection mechanisms designed to determine the order in which different tasks are permitted to execute (part of) their preferred actions. It should be noted that despite being in the field of *action selection*, techniques in interleaved multi-tasking in fact selects tasks, not actions. Once selected, a task is free to execute whichever action is most beneficial to its progression without paying heed to the impact its execution may have on the other goals being pursued.

Seeing as the order of execution is determined by repeatedly selecting the most eligible from a set of tasks, this class of ASMs was labelled “Arbitration” by Saffiotti [105] in 1997. At that time the category was equivalent to what MacKenzie called “state-based” [83], but was later presented as one of three sub-categories of Arbitration by Pirjanian in his 1998 thesis [92].

The concept of choosing between available options is however such a fundamental principle that it plays, at least a minor, role in most ASMs today. That includes mechanisms for simultaneous multi-tasking, such as *voting*, which is why a separation based on the use of arbitration no longer appears meaningful.

Benjamin’s 2002 taxonomy conflates all approaches based on alternating which task is in control by branding them “No Compromise” [9, pp. 8–9]. This view is supported by the fact that controls selected at any given time reflect only the needs of a single task. It does, however, neglect to acknowledge that time is the contended resource in these approaches and sharing it does constitute a form of compromise.

The term *interleaved multi-tasking* is suggested as an alternative since it better conveys the notion of tasks being granted full control over the robot for short periods of time. Sub-categories distinguish between four different strategies for determining which task is to be granted control in the next discrete time step or interval.

2.2.1.1 FSA Based

This category corresponds to MacKenzie’s definition of *state-based* ASMs, but was relabelled due to the ambiguity of the term *state* which in this context refers to the state of a Finite State Automaton (FSA) and not to the physical state of the robot or that of the environment it inhabits. Most early discoveries in the field of action selection fall into this category due to heavy reliance on *if-then* style production rules which map percepts to simple, reflex-like behaviours.

Famously amongst them is Brooks’ Subsumption Architecture [18], which laid the foundation for behaviour-based robotics. At the time of its publication in 1986, most research was aimed at building a comprehensive model of the agent’s environment and reasoning about it using symbolic search algorithms [19]. Brooks pointed out that such complete knowledge could only be realistically assumed in simulated environments [20]. Solutions relying on such a model are slow, unresponsive and ill-suited to solving real world problems [21]. The Subsumption Architecture addresses these issues by using a decentralised representation of the world in which a hierarchical system of behaviours controls the robot’s actions. Each layer in the hierarchy provides a fairly simple, and therefore fast, mapping between incoming sensor information and the controls to be executed in response. Finite state automata lend themselves well to this rule-driven approach. The modular design allows each behaviour to be implemented by a separate FSA which is incorporated (or subsumed) into the FSA of the behaviour in the layer above it. Only one behaviour can be active at any given time but may be interrupted by others if need be. Higher level tasks such as mapping the environment may, for instance, be interrupted by lower level behaviours such as obstacle avoidance. A central model of the world is not necessary because each behaviour receives the information it requires directly from the sensors [18].

Another FSA based solution put forward by Košecká and Bajcsy [71] in 1994 follows the Discrete Event System (DES) design. Their method can be seen as the event driven equivalent of the Subsumption Architecture. State transitions are triggered by events, which are meaningful changes to the environment, as opposed to any change in the sensor readings. When a new state is reached, a pre-defined open-loop behaviour is activated. A second FSA, called the "supervisor", implements a closed-loop system that guides the first FSA by enabling or disabling certain events, thereby effectively modifying the state transition table.

Arkin and MacKenzie [6] propose a comparable technique which replaces events with "perceptual schemas" and behaviours with "motor schemas". Schemas can be seen as small, well-defined processes that perform a specific functionality. Perceptual schemas, which are of particular interest in their work, implement an efficient sensor management strategy which allows for selective perception of only the relevant regions of the environment. Motor schemas are notified of environmental changes pertaining to the behaviour they represent and may be activated as a consequence. A detailed explanation of schemas can be found in Arkin [1] although the method described there does not belong in this category.

A 1997 revival of the DES approach by Pirjanian and Christensen [26] implements a three layered "hybrid" architecture consisting of an FSA based supervisor controlling a DES behaviour module, which in turns sits on a reactive module that handles unexpected, low level events. The supervisor is intended to "reconcile AI planning with reactive modules" [26, p. 2], which the authors find lacking in the approach suggested by Brooks. The supervisor is however still limited to sequencing behaviours.

Gat presents a paper on "Three-layer architectures" [43] in which behaviours call on a top layer "sequencer" to determine execution order. Bennett and Leonard [12] employ a similar hierarchical control system in order to perform feature detection using an underwater vehicle.

2.2.1.2 Learned Rules

With FSA based approaches, most of the logic controlling task priorities and activation is implicitly contained within the fixed structure of the automaton. To change the emergent behaviour of such systems, they have to be redesigned [90, p. 11]. Even if only one behaviour is changed, the alteration may have a knock-on effect on other tasks which then also have to be recalibrated. Despite various tools for generating FSAs from a description, maintaining and adjusting complex systems can be difficult and time consuming.

An alternative to hard-wiring production rules is to use reinforcement learning. Such systems are trained to produce similar reactions to the same stimuli that cause state transitions in FSA based approaches. Examples of this are a Q-learning technique by Lin [77] and a W-learning approach by Humphrys [55]. Of course, these systems still have to be retrained each time their behaviour is to be changed. Doing so requires a new reward function to be constructed, which can also be a challenge.

2.2.1.3 Behaviour Graphs

Behaviours can also be modelled as a collection of interconnected nodes that form a graph along the edges of which activation signals are propagated. When the sum of all inputs to a node overcome an activation threshold, its associated behaviour is activated and gains control of the robot. In other words, this method follows the same principles as neuron activation in artificial neural networks.

Maes [84] and [85] demonstrate an implementation of this idea in which activation levels are determined by the difference or disparity between the robot's current and goal configurations. More specifically, nodes in the "Activation Network" receive energy when the execution of their proposed actions brings the robot closer to its target(s) or enables another action that does. Conversely, energy is deducted where the action might hinder or reverse progress - be it directly or by blocking another useful behaviour. The amount of energy disseminated upon attainment of certain states, as well as the threshold values for node activation, have to be hand tuned on a case-by-case basis.

Decugis and Ferber [28] expand on Maes' work by constructing a hierarchy of several Activation Networks which effectively sequence higher level tasks that are inherently conflictual and cannot be tended to at the same time. Although behaviour selection is more sophisticated, it still elects a single behaviour in a winner-takes-all fashion, meaning that other tasks have no bearing on the action chosen by the behaviour put in charge.

Arkin and O'Brien [88, p. 149] introduce the "circadian rhythm function" which makes an additional contribution to each node's activation levels. The additional element represents a forecast of future consequences that may ensue from present choices. This addresses a weakness in previous methods of this type which are unable to account for uncertainty, e.g. relating to sensor error or changes in a dynamic environment.

2.2.1.4 Scheduling

Scheduling is an area that frequently comes up in searches for the keyword *multi-tasking* but is absent from the existing taxonomies because ideas that originate in the design of operating systems have only recently been transferred to robotics.

One of the earliest approaches in this direction, although not yet linked to operating systems, was presented by Roeckel et al. where "The Task Manager prioritizes the Behaviors requesting control and grants control to the one with the highest priority" [100, p. 322]. The prioritisation scheme appears not to be automated as it "uses a downloaded set of orders" [100, p. 322], which is taken to mean that priorities are static and pre-determined.

A more interesting contribution to the scheduling category was made by Groth in 2013 [47]. The control system proposed is derived from scheduling and context switching techniques used in multi-threading. By adding the ability to interrupt and resume tasks, the robot's reactivity is improved, allowing it to interact with a dynamic environment or a human operator. The focus in Groth's work is on environment manipulation tasks, as described in category 2 of my task taxonomy, and is not directly applicable to this PhD. This is also due to the limited notion of concurrency being used. The Concurrent Sequential Process (CSP) model which Groth builds on is only capable of simulating or approximating concurrency by interleaving tasks; it does not provide SMT capabilities. Ideas presented here may be relevant in future work however, e.g. in the design of a high level planner to break down category 2 tasks into a sequence of category 1 tasks.

2.2.2 Parallel Multi-tasking

Parallel multi-tasking is chosen as the term to describe all mechanisms designed to distribute work amongst *independent* entities being coordinated concurrently. These entities may be individual actuators mounted to and controlled by a single robot, or they may be entirely separate agents working towards a common goal. Swarm robotics is the field concerned with the latter case while the former relies on a type of dependency resolution to select appropriate controls for each of a robot's motors. Both have in common that they strive to achieve a larger purpose by decomposing the necessary actions into sets of conflict free operations which can be performed in parallel.

The main difference is the scale or granularity at which they operate. Since this PhD focuses on fine grained coordination of a single robot, swarm robotics is only briefly mentioned for the sake of completeness.

More attention is given to mechanisms which provide the ability to find and execute non-conflicting controls in parallel using different actuators of the same robot. Such techniques have the potential to better utilise available hardware by reducing the idle time inherent to interleaved multi-tasking. Approaches based on this strategy are reviewed under *dependency resolution* below.

2.2.2.1 Swarm Robotics

Swarm robotics is a large field that deals with many different issues aside from action selection such as networking, communication protocols, collaborative modelling of the environment etc. Its categorisation into parallel multi-tasking is due to the fact that coordinating robots in a swarm has a lot in common with coordinating different parts of the same robot. The planning methods required also have to disentangle conflicts and are incentivised to improve efficiency by minimising time spent waiting for other actions to complete [49].

A central idea behind solving problems in this category is to identify actions that can be taken without impeding or obstructing others. In short, the aim is to coordinate multiple robots with multiple actions. This differs from the main aim of this thesis which is to coordinate multiple tasks with a single action. Hence why the focus in the category of parallel multi-tasking is on approaches designed for individual robots from which very similar lessons can be learned more directly.

Distributed, swarm, team and collaborative robotics fall into the MR category of Gerkey and Matarić's [44] taxonomy. Depending on whether individual robots are assigned parts of one task or given separate tasks, these MRTA problems are classed as MR-ST or MR-MT respectively. Further reading on the former category can be found in [61] and [135], while [125] is on the subject of the latter. Key concepts in these areas are *distributed problem solving* and *coalition forming*, which are not applicable to the SR domain for obvious reasons. A more general overview of research in the field of swarm robotics is presented in [132].

2.2.2.2 Dependency Resolution

When it comes to parallelisation of the work-flow on a single robot, there is no way around addressing task dependencies. This may be achieved by a very fine-grained method of action selection, whereby individual control variables are selected one at a time. In this way an entire control vector is assembled to make efficient use of the robot's hardware.

Lenser [76] implements a hierarchical control system that incorporates a mode-based approach called "Dual Dynamics" [59] into Brook's Subsumption architecture. A "conflict graph" facilitates dependency resolution. Nodes in the graph represent the tasks waiting to be executed. Those which require use of the same actuators are deemed to be in conflict and are linked by an edge indicating that they cannot be simultaneously progressed. A “combinator” extracts all non-conflicting sets of tasks from the conflict graph and evaluates them in terms of the combined reward which their activation will predictably yield. The candidate task sets are whittled down by process of elimination until only the set promising maximum total reward remains for execution. Optimality can be achieved provided the reward for each task is accurately predicted.

A different way of casting the issue of parallelisation is to model it as a resource allocation problem where the robot’s actuators represent the contested resource. Such an approach is presented by Taipalus [119] [118] who, like Groth [47], takes inspiration from process scheduling. Unlike Groth’s sequential scheduling technique, Taipalus mixes “the order of execution for multiple tasks so that parts of tasks which utilize the interdependent resource of the robot can be executed in parallel” [119, pp. 1–2].

The “ActionPool” [118, p. 38] mechanism is described as working in a similar way to Apple’s Grand Central Dispatch (GCD) system. Each resource, which is a physical subsystem of the robot, manages a separate pool of actions waiting for execution. Parallel multi-tasking is made possible by creating actions in such a way as to guarantee their atomicity, i.e. independence of other actions. This allows each resource to process exactly one action from its pool at any given time and since there are several action pools, this means that multiple actions run concurrently. Action priorities allow more important or urgent tasks to skip the queue and receive the attention they require. The mechanism used to create atomic actions is not very well described but uses the Beliefs-Desires-Intentions (BDI) agent paradigm [96] in a similar way to Joyeux, Alami and Lacroix [62]. Human insight is also used in the formation of plans to a considerable extent, which detracts from the value of the controller in terms of its autonomy.

Towle and Nicolescu [121] present an auction based system in which competing behaviours bid for the use of individual actuators. The auction mechanic turns out to be a thinly veiled metaphor for selecting the most important task. Just as in Maes’ Activation Network, importance is quantified in terms of an “activation level”, the magnitude of which is, however, determined by environmental factors instead of “inter-behavior communication” [121, p. 159]. The authors point out the problems with using static priorities to determine task precedence in case of conflict and also reject deliberative methods due to the complexity of dynamic environments [121, p. 158]. Instead they claim to be able to resolve contentions by using time constraints together with information about the environment and the robot’s workload. Essentially, behaviour priority increases as its deadline approaches [121, p. 163]. Most importantly the system sets estimated completion times itself and does not rely on the operator to supply these parameters.

Knips et al. [68] present a neural dynamics architecture for reaching and grasping. Their process model relies on Dynamic Field Theory [110] to handle everything from computer vision and shape classification to pose estimation and movement generation. Despite the overall complexity of the system, task coordination is limited to sequencing behaviours that are activated when certain preconditions are met. If it were not for the fact that the robotic hand can be opened while approaching the object to be grasped, this technique would be classified as Interleaved Multi-tasking. Once the robot arm is in place, the grasping behaviour is activated, followed by a lifting behaviour. The controller does react to unforeseen changes and can adapt its actions, but at no time are the same actuators simultaneously contested. The most likely application of the techniques described would be in sequencing multi-phase tasks or adding a scene interpretation module for category 2 tasks.

2.2.3 Simultaneous Multi-tasking

While the field of action selection has its roots in interleaved multi-tasking, techniques in that category no longer represent the state of the art. Even though many of the reviewed ASMs serve their purpose well, they simply lack the capacity to pursue multiple goals concurrently. Arkin was one of the first to point out the problems with ignoring all but one task and many others followed [4] [1].

Parallel multi-tasking makes progress towards reducing the idle time of individual actuators but sidesteps rather than solves the problem of coordinating dependent or conflicting tasks.

Simultaneous multi-tasking finally commits to confronting the issue of resolving contentions between competing tasks concurrently. Different terms have been used to describe ASMs that do so. MacKenzie uses the term “continuous” [83, p. 5] but only gives a very vague description, a fact lamented by Pirjanian [90, p. 8], who proceeds to equate it with Saffiotti’s “command fusion” [105, p. 12]. Safiotti defines command fusion as the task of combining “the results from different behaviors into one command to be sent to the robot’s effectors” [105, p. 12] but contradicts himself on the following page by explaining that “The simplest way to fuse the commands from different behaviors is to use a switching scheme: the output from one behavior is selected for execution, and all the others are ignored”. The latter is a description of interleaved multi-tasking while the former, which later turns out to be the intended meaning, corresponds to *Preference Interpolation* as it is presented here. Another subcategory in my taxonomy, *Relative Weighting*, is quite well described as “voting” by Pirjanian [90, p. 18] but is more often mistaken for command fusion. Simultaneous multi-tasking encompasses all of these terms and sorts contributions in this area into three clearly distinguished subcategories.

2.2.3.1 Preference Interpolation

This category corresponds to the literal interpretation of *command fusion*, whereby a compromise control vector is formed by amalgamating individual preference vectors representing each task's ideal solution.

Khatib unwittingly made one of the first contributions to SMT in a 1986 paper on "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots" [66]. His "Potential Fields" method was not originally conceived as an action selection mechanism, but was, as the title suggests, a contribution to obstacle avoidance. Almost any book on AI meanwhile includes a well illustrated explanation of potential fields (e.g. [112, p. 146–148]), which is why only their role in ASMs will be highlighted here. The technique requires each task to define a potential (cost) function over control space, which describes a surface called the potential field. Descending on this surface by following its steepest gradient, a process called "gradient descent", leads to the controls associated with least cost for that task. Additive superposition of all tasks' potential functions yields a combined surface which can be used to find the overall preferable action. Equivalently, one may express each task's preference in terms of a control vector pointing in the direction of steepest descent on its own potential surface. Averaging the individual preference vectors produces the same compromise as found using the combined potential field. In brief, a mutually agreeable action is generated by interpolation between task preference vectors.

Arkin was the first to realise the potential of this technique in the context of action selection and made extensive use of it in "Motor Schema Based Navigation" [1]. A brief explanation of "perceptual schemas" and "motor schemas" has already been provided in the review of Arkin and MacKenzie. While the concept of schemas remains the same, here motor schemas are implemented using potential fields. When a perceptual schema is sufficiently certain it has located an obstacle, a corresponding motor schema is instantiated to create a repulsive force that keeps the robot at a safe distance. Another schema attracts the robot towards its goal location using a different potential function. In this way, each active motor schema (i.e. behaviour) contributes a potential field and associated control preference. The system produces a compromise action by linear combination of the individual task control vectors.

Another approach using additive superposition was contributed by Schöner [109] in 1992. The details of the technique are quite different but the underlying principles are the same as those already described.

Even if the technical issues (e.g. local minima) typically associated with potential fields can be solved, there is an even more fundamental problem with combining preferred action vectors. The problem stems from the loss of information incurred by discarding all but a task's ideal action [9,

p. 10]. This leaves nothing but an erroneous assumption of linearity on which to base the quality estimate of linear combinations of those ideals. In fact, even monotonicity cannot be assumed, meaning that "arbitration via vector addition can result in an averaged command which is not satisfactory to any of the contributing behaviors" [102, p. 44].

Low et al. [80] use a self-organising neural network in an attempt to bridge the gap between high-level task planning and low-level motion control. The highest level in the four-tiered architecture generates a sequence of checkpoints for the robot to pass through on its way to a goal location. Lower levels are seen as reactive components with the responsibilities of transitioning between checkpoints (using controls generated by the neural network), avoiding obstacles and monitoring the robot's internal state. The "command fusion module combines the control commands from the reactive components into a final command that is sent to the actuators" [80, p. 3]. This approach does circumvent any possible problems with local minima in potential fields, but does nothing to address the more fundamental issue of preference interpolation.

2.2.3.2 Relative Weighting

To overcome the problems with single preferences, ASMs evolved to take multiple alternative actions into consideration. Benjamin categorises these approaches as "multifusion" [9, p. 10], but the term may give the false impression that the alternative control vectors are somehow convolved. Pirjanian describes the process of compromise formation as "voting" [90, p. 18], which is more accurate – although it should not be taken to mean that each task may only cast a single vote or that votes are binary values. More precisely, ASMs in this category *rate* all alternative actions in a given set on a continuous scale according to their individual and egoistic preferences. Once all candidate actions have been rated, the one that commands the highest total rating is selected for execution. However, before computing the total, each task's preference must be scaled by the relative weight (or importance) of that task. Weighting provides the crucial ability to direct attention towards more critical tasks, such as preventing imminent collision. While other details of the techniques reviewed here may vary, they all provide a relative weighting scheme that gives name to this subcategory.

The method as outlined above already serves as a good description of Rosenblatt's highly influential paper "DAMN: A Distributed Architecture for Mobile Navigation" [101], from which the following illustration was taken.

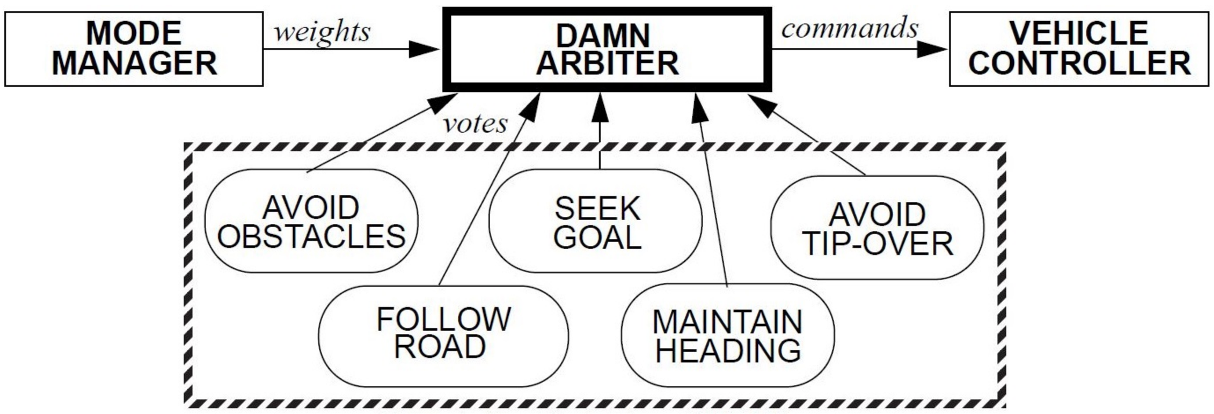


Figure 2.2: Illustration of components in Rosenblatt's "Distributed Architecture for Mobile Navigation" (image taken from [101, p. 168])

The "DAMN Arbiter" weights the votes provided by each behaviour and sends the command with the highest weighted total to the vehicle controller. What is not immediately clear from the diagram is that there exist two sets of weights: "Each behavior is assigned a weight reflecting its relative priority in controlling the vehicle. A mode manager may also be used to vary these weights during the course of a mission based on knowledge of which behaviors would be most relevant and reliable in a given situation" [101, p. 168]. The relative task priorities are set a priori using human insight whereas the mode manager may (optionally) use insights from a higher level planner to modify those weight distributions if necessary. Finer details of the method can be found in Rosenblatt's thesis [102].

Rosenblatt [103] expands on his previous research by adding elements of sensor fusion to the "DAMN" architecture, thereby creating a crossover he terms "utility fusion". Utilities are assigned at regular intervals along candidate control vectors and summed after scaling by a diminishing factor that represents uncertainty growing with increasing distance from the robot. The control vector with highest utility is chosen, but not executed directly. Instead a parabolic interpolation with adjacent controls facilitates smoother transitions than would be possible with bang-bang control where abrupt instruction changes lead to jerky motion. The rather complex algorithm consists of ten steps detailed on pages 4–5 [103].

In the same paper Rosenblatt also comments on work done by Yen and Pfluger [131] to incorporate his previous work into the fuzzy logic framework (a rule based system using multivalued logic). Fuzzy logic approaches were popular around this time with several contributions including [130] [105] [92] [106] [91]. Rosenblatt, however, observes that the entire category of fuzzy approaches still has shortcomings which reduce the system's overall effectiveness [103, p. 2] and by 2002 Benjamin states: "In summary, no defuzzification technique has been put forth that isn't readily open to a flaw-revealing counterexample" [9, p. 106].

Riekkie [98] proposes "action maps" to deal with a flaw in Rosenblatt's method which lies in the fact that control variables are determined sequentially. Specifically, the direction of the next transition is chosen before deciding on a suitable speed. By failing to consider both parameters simultaneously, clearly existing superior solutions may not be discovered. This is well illustrated in [9, pp. 11–12]. In Riekkie's approach, the full n-dimensional control space is searched (where n is the number of controls or degrees of freedom).

Benjamin develops a technique called "Interval Programming" (IvP) [9] to deal with finding solutions in the now exponentially larger search space. Task rating and weighting mechanics remain unchanged with the focus being on a general solution to optimisation of a complex objective function in high dimensional search spaces. The proposed mechanisms is essentially a sampling approach which restricts itself to evaluating a feasible number of control samples and then using linear interpolation to approximate intermediate values. A demonstration of this technique in the context of marine vehicle control attests to the feasibility of the IvP method [10]. Benjamin et al. [11] provide a guide to "MOOS-IvP" and "IvP Helm" solution tools which can be applied to a diverse set of problems. This technique will likely be of use to the work conducted here once it is applied to a robot with more degrees of freedom.

The described approaches have made significant advances, but share a common difficulty inherent to this category, namely that of determining suitable relative weightings. For example, Gadanho and Custódio comment that "One of the most difficult problems was to determine the relative weights of importance" [41, p. 182]. Everaere and Grislin-Le Strugeon echo the same sentiment in stating that such approaches "have shown difficulties in determining the behaviors weights for the vote" [38, p. 1]. The problem is fundamentally that different tasks express their preferences using objective functions with diverging ranges which also produce values in different units. Finding relative weights is an exercise in determining a fair conversion rate that allows these different currencies to be equated or compared. To make the matter worse, relative importance may vary due to scarcity of a particular resource, like in any other economy. This rules out the use of static, predetermined weights. The robot should, for instance, prioritise shorter paths more and more as fuel levels drop. Doing so may, however, require passing by obstacles more closely than is deemed safe. In this example, the conversion rate provides the answer to the question: How many centimetres closer to the obstacle should the robot go per litre of fuel saved? This question is already hard to answer although there are only two tasks involved. The difficulty we have in quantifying priorities is likely explained by the fact that we do not think or plan this way ourselves. An alternative, biologically inspired approach that does not require answering such questions is explored in the following.

2.2.3.3 Universal Scale

Since static weightings are insufficient and assessing relative importance dynamically is problematic, this thesis proposes the use of a universal scale for rating and comparing tasks. Instead of answering the question about relative importance, the use of a common currency is proposed. The difficulty in this is finding a commonality between a diverse range of possible tasks on the basis of which a suitable metric can be defined. Inspiration is taken from biology where even primitive organisms can be observed making sensible trade-offs between their varying needs. The mechanisms that controls decision making is based on an attention to threat principle called homeostasis.

2.3 Homeostasis

This section investigates the possibility of defining a universal scale for rating tasks on the basis of homeostatic urgency. A brief history and explanation of the concept of internal stability leads to a discussion of some of its applications to AI and robotics.

2.3.1 The Attention to Threat Principle

The French physiologist, Claude Bernard, first discovered the "concept of constancy of the internal environment" [46, p. 380] of living organisms in 1854 and developed it throughout his career. From his observations of the regulatory system that controls body temperature, blood sugar levels and many other critical functions, he extrapolated a more general theory about the internal workings of the body [46, p. 383]. In particular, he notes that the "milieu intérieur" (internal environment) seeks to maintain a state of equilibrium necessary for survival.

This principle, by which the body antagonises disruptions to its internal parameters and returns to a stable state, was named "homeostasis" by Cannon. In his book "The Wisdom of the Body" [23], Cannon investigates the mechanisms by which equilibrium is achieved through a series of experiments. It is these self-regulating mechanisms that gave rise to the field of cybernetics founded by Norbert Wiener [127].

A breakthrough in this newly established field came in 1948 when Ashby created the "Homeostat" [7, p. 100], the first mechanical device capable of replicating homeostasis. Key in the development of this adaptive "Ultrastable System" [7, p. 80] was Ashby's more concrete and practical interpretation of homeostasis as "the maintenance of essential variables within physiological limits" [7, p. 63]. Expressing the state of the internal environment in terms of variables enabled a more proactive engineering approach to what was previously a fairly abstract biological process that could merely be observed. Using the abstraction afforded by this algebraic interpretation allowed the essential variables to be plotted as points in a "phase-space" [7, p. 95]. The task of imitating the body's natural tendency to restore and maintain stability is then modelled as keeping those points within a safe region of phase-space. Taking inspiration from nature, Ashby observes that "Some external disturbances tend to drive an essential variable outside its normal limits; but the commencing change itself activates a mechanism that opposes the external disturbance" [7, p. 61]. Stability and equilibrium can be achieved the same way mechanically – by favouring actions that counteract environmental changes through an equal and opposite force.

This thesis investigates a novel way of applying homeostasis to simultaneous multi-tasking. The basic idea is to recast task objective functions as homeostatic indicators of success in balancing the needs of tasks competing for attention. When a task comes under threat due to the external influences of its rivals, its homeostatic indicator will be pushed outside of its safe range and thereby trigger a corrective mechanisms for restoring balance. While analogous to Ashby's view of homeostasis, it should be noted that the outlined approach is an application of a principle for internal stability to an external measure of a task's overall success. Obstacle clearance for instance is defined in relation to obstructions in the external environment and is not equivalent to the regulation of internal variables such as body temperature. To capture this extended notion of homeostasis, its more general interpretation as the *attention to threat principle* is conducive.

In this very general sense the solutions in the *Relative Weighting* category of SMT could already be seen as homeostatic. For the reasons pointed out in that section, a different approach is required however. Interestingly, Ashby aptly states the very essence of the problem that these control systems are still wrestling with today. In "Design For a Brain"[7, p. 43], he describes the issue with the non-uniformity of the essential variables as well as the need for ordering them according to their importance in spite of this:

> The essential variables are not uniform in the closeness or urgency of their relations to lethality. There are such variables as the amount of oxygen in the blood, and the structural integrity of the medulla oblongata, whose passage beyond the normal limits is followed by death almost at once. There are others, such as the integrity of a leg-bone, and the amount of infection in the peritoneal cavity, whose passage beyond the limit must be regarded as serious though not necessarily fatal. Then there are variables, such as those of severe pressure or heat at some place on the skin, whose passage beyond normal limits is not immediately dangerous, but is so often correlated with some approaching threat that *is* serious that the organism avoids such

> situations (which we call "painful") as if they were potentially lethal. All that we require is the ability to arrange the animal's variables in an approximate order of importance.

This PhD addresses the described ordering problem from a new perspective. Instead of trying to compare variables in different units and with different ranges, a fair comparison in terms of a universal measure is attempted. A suitable metric is already indicated in the quote above: **Urgency** allows even the most diverse maladies in Ashby's extensive list to be ordered in terms of their severity. A framework and mechanism for conflict resolution through fair compromise based on homeostatic urgency is developed in the next chapter, but not before taking stock of existing attempts to apply homeostasis in robotics.

2.3.2 Homeostasis in Feedback Loops

Although it only maintains one homeostatic variable, the lighting control system presented in [63] displays some of the basic principles behind counteracting undesired change in the environment. A robotic chandelier, consisting of multiple spotlights that can be moved to shine in different directions, is programmed to maintain desired lighting conditions specified by the user through speech and gestures. When the brightness, recorded by a camera in the ceiling mounted light fixture, deviates from the target value, the controller compensates by moving its spotlights or changing their intensities. While luminosity can be said to be under homeostatic control, the mechanism to maintain the desired state is, essentially, a simple feedback controller. The most effective positions for the lights is determined using depth-first search and is of little relevance to SMT.

A more interesting use of luminance control is presented by Hernández et al. [51] to increase the quality of camera images used to guide a line-following robot. Luminance and white balancing are kept in acceptable bounds by adjusting camera settings such as zoom, focus and shutter speed. As in [63], the magnitude of deviation from the ideals values leads to a proportionate response. All of these controls are, however, independent and do not involve conflict. The only contended resource is time and that is managed by slowing down the robot if needed. Speed is controlled using fuzzy if-then rules integrated into a hierarchical control system.

Neither of these two examples go beyond using homeostatic variables in a feedback loop. While the aim is to achieve a stable state, the principles of homeostasis are not integrated into the control systems themselves, which follow standard closed-loop designs.

2.3.3 Homeostasis in Neural Networks

Within the field of AI, homeostasis appears to have made the largest impact on artificial neural networks. One of the earliest ideas to apply Ashby's notion of internal stability to neuron activation was put forward by Di Paolo [30] in a paper titled "Homeostatic Adaptation to Inversion of the Visual Field and Other Sensorimotor Disruptions". As the title suggests, it aims to overcome perturbation in sensory inputs by adaptation. For demonstration purposes, a simple robot is made to approach a light source using two light sensors mounted on opposite sides of a circular vehicle. The key feature is that the sensors can be inverted, i.e. the signals emitted by the left and right sensors can be crossed over. Methods for recovering from this extreme form of disruption are refined over a series of publications in 2002 [31], 2003 [32] and 2008 [56]. The basic idea is to use a neural stability indicator (the homeostatic variable) in addition to the usual fitness function to allow the system to withstand and adapt to sensorimotor perturbation.

Following Di Paolo's work, there are a number of contributions aimed at improving neural networks and, in particular, their robustness. Timmis and Neal [87] present an artificial endocrine system based on a fairly literal interpretation of biological homeostasis that includes emotional responses. Vargas et al. [124] [123] and Moioli et al. [86] draw and expand on work done by Timmis and Neal. In order to build a more complete and biologically accurate homeostatic system, the existing neural network approach is incorporated into a larger framework together with an artificial immune system. The aim, especially of [124], seems to be the recreation of some form of cognition, while [86] and [123] are more focused on neural networks called "GasNets". Finally, Williams [128] applies homeostatic plasticity to continuous-time recurrent neural networks in order to improve signal propagation.

2.3.4 Homeostasis in Artificial Life

Work on artificial neural networks seems to be gravitating towards cognition, swarm robotics and more generally Artificial Life. Schmickl et al. [108] apply homeostasis to the field of Artificial Life, where the adaptive principle is to aid in the generation and evolution of artificial organisms. An argument for the use of homeostasis in swarm robotics is made by Timmis and Tyrrell [120]. Stovold combines research from Vargas, Schmickl and Timmis (among others) to investigate "the fundamental basis of collective intelligence by considering distributed cognition and its role in adaptivity and homeostasis in robots" [116, p. 1].

While very effective in these areas, neural networks and evolutionary algorithms are not well suited to multi-tasking. One of the main reasons for this is a limitation pointed out by Stovold himself stating that "the range of behaviours is limited to those already trained into the neural network, and retraining takes time" [116, p. 2]. Approaches to SMT, building on neural networks would inherit the same issue and therefore fail to provide the aspired real-time computation and/or generality: The neural network would have to be retrained for every new combination of tasks and that is assuming that training data is even available. The same holds true for other learning approaches, such as the goal-directed imitation system presented in [36].

There may be applications for artificial neural networks and machine learning at the task-level, but a different mechanisms is required for top-level coordination. The following summarises the few attempts that have been made towards using homeostasis in that context.

2.3.5 Homeostasis in Task Coordination

In "Homeostatic Control For A Mobile Robot: Dynamic Replanning In Hazardous Environments", Arkin [3] seeks to enhance robot autonomy by facilitating self-preservation of internal state through homeostasis. His proposed mechanism follows an analogy to the endocrine system, which is responsible for homeostatic regulation in the biological world. The endocrine system communicates through the release of hormones (such as adrenaline) which can, among other things, control energy levels and thereby increase or decrease activity in certain parts of the body. Arkin introduces "signal schemas" which mimic the hormone transmitters and receptors to provide a means of disseminating information about the robot's internal variables and provoke the appropriate adaptive reactions in the motor schemas.

The outlined approach comes to full fruition in a follow-up publication [2], which demonstrates visible adaptation in a control system sensitive to the robot's fuel reserves. The implementation builds on Arkin's previous work on motor schemas in 1987 [1] (see review above) to which an additional layer, the "homeostatic control subsystem", is added. Feedback concerning the homeostatic variables is passed (via signal schemas) to the motor schemas where the information parameterises the potential field used to produce each behaviour's control vector. Summing all schema vectors, yields an overall control vector which represents a compromise between following a shorter path (to save fuel) and going a longer way around obstacles (to avoid collision). Experiments show success but the method is still in the same category as the approach it builds on (Preference Interpolation) and suffers from the same limitations already described in the above taxonomy of ASMs. A summary of Arkin's contributions to behavioural control can be found in his book "Behavior-based Robotics" [5] published in 1998.

Gadanho [40] [41] uses homeostasis to train a control system to switch between predefined behaviours as and when appropriate. Disruption to the homeostatic balance of certain internal parameters triggers an "emotional" response, which guides reinforcement learning. The approach falls under the *Learned Rules* category of interleaved multi-tasking.

Yoon et al. [134] make a rather weak contribution to parallel multi-tasking which can be described as performing interleaved multi-tasking for independent sets of behaviours in parallel. The behaviour sets appear to be pre-defined, meaning the controller does not perform the dependency resolution step autonomously. As is typical for parallel multi-tasking, only one behaviour in each set can be active at the same time since there are no means of coordinating the dependent behaviours within the set. The principle by which behaviours are activated is of interest however, owing to the fact that it is based on homeostasis. The concept is quite simple: Each behaviour is assigned a "satisfaction level" and the least satisfied behaviour executes its desired controls [134, p. 3160]. Unfortunately, the paper does not explain how satisfaction levels are determined, but from the perspective of this thesis the main contribution of this paper (although not presented as such) is its generalisation of homeostasis and not the mechanism itself. The key difference to other approaches is that homeostasis is applied to entire behaviours, not just individual variables such as fuel levels. Since behaviour satisfaction is not even part of a robot's internal state, the approach can be seen as extending the concept of homeostasis to the stabilisation of external or derived variables. This generalisation fits with the interpretation of homeostasis given above, where it is described as a general *attention to threat principle*.

The allostatic control system constructed by Fibla et al. [107] for emulating a rodent is of significant interest to this PhD. It combines homeostatic principles with what is arguably the most sophisticated action selection mechanism in the Relative Weighting subcategory of SMT. This thesis does not make a distinction between homeostasis and allostasis, as homeostasis is taken to encompass both the idea of keeping a variable within its bounds as well as the mechanism of antagonising change to achieve balance. In the article, homeostasis is used to describe a simple feedback loop whereas allostasis involves deliberate behavioural changes with the aim of maintaining stability at a meta level. Behaviours are "decomposed into minimal homeostatic subsystems" [107, p. 1935], each of which is associated with a gradient vector field that is generated as part of a complex learning process. Once created, the allostatic control control system, a kind of supervisor, can set desired values for each behaviour's objective function. Allostasis manifests itself in the behavioural change induced by setting different targets. The gradient vector field is used to determine whether to increase or decrease the wheel speeds of an E-puck robot so as to bring the actual objective value closer to its target. Gradients of multiple behaviours are combined using a weighted sum, thus placing this method in the Relative Weighting category. Since there is no description of weights being dynamically computed, it can only be assumed that these are manually selected, highlighting one again the problems with this category of solutions. Another limitation lies in the fact that behaviours are rather basic and only include category 1 tasks. Cue following, path planning (towards a target destination) and environment exploration are essentially different instances of the Goal Location task. It may also prove difficult to apply the technique for adjusting controls to other robot architectures, since it seems to be specifically designed for a differential drive robot.

CHAPTER THREE

HOMEOSTATIC TASK COORDINATION

This chapter describes the proposed framework for task-agnostic SMT and the homeostatic mechanism developed for conflict resolution. A taxonomy of tasks is provided to specify the types of problems the framework can handle and represents the first step towards its design. Clearly defining how the word *task* should be interpreted, allows the system's desired features and requirements to be laid out. A diagram of the framework illustrates the components that are needed to satisfy these requirements and shows how different parts of the mechanism fit together. The internal workings of each component are laid out before the final section presents Homeostatic Mortality Reduction as the core mechanism for conflict resolution. Central to the presented solution is the universal scale on which task priorities can be rated and compared in terms of their homeostatic variables. This rating scheme is integrated into the framework by changing the functionality of some of its key components.

3.1 Taxonomy of Tasks

The word *task* can easily be misconstrued to mean almost anything that needs to be done. The unbounded nature of the work required is due to a somewhat recursive definition: a task may be described in terms of a sequence of actions which can be tasks in their own right and could be broken down further still. For example, the task of baking a cake involves buying ingredients, mixing them together, greasing a cake tin, etc. – but buying the ingredients is a task itself that requires another sequence of actions (go to a shop, get a shopping cart, find the flour, ...) and so on. Some tasks are even too abstract or open ended to be broken down. Well meant but impractical advice such as "make something of your life" comes to mind. Using such a loose definition of task would make it impossible to set achievable objectives since the diversity of potential problems would require a universal AI as it only exists in science-fiction.

To avoid being taken to task for an imprecise definition, a more practical interpretation is established with a view to specifying achievable requirements for the task coordination framework. More than that, the following taxonomy of tasks is already part of that framework since it defines its purpose and scope. It also addresses the first hypothesis in as far as it demonstrates the ability to capture a reasonably broad spectrum of tasks, the coordination of which seems both interesting and plausible.

The taxonomy expounded in the following distinguishes between the types of tasks that can be *simultaneously* pursued by a *single* robot. As already explained in Section 1.4.2 of the introduction, its closest correspondence in literature is the "ID [MT-SR-IA]" category of the "iTax" [69, p. 1504] taxonomy for Multi-robot Task Allocation (MRTA) problems. The taxonomy presented here focuses on the coordination of tasks, rather than on the process by which they are assigned. It also introduces additional divisions to split up what would otherwise be a very broad category of tasks. To do so it contrasts Multi-phase (MP) tasks and Single-phase (SP) tasks before sub-dividing the latter even further.

3.1.1 Multi-phase Tasks

Multi-phase tasks involve inherently sequential elements and must be solved in stages. As previously described, tasks are often recursively defined or may otherwise imply requirements not explicitly stated. This can make it difficult to decide if or how a task needs to be decomposed into consecutive phases.

For example, it may seem as though a lawn-mowing robot only pursues one continuous task, when in fact it must pass through several waypoints to cover the entire lawn. The robot cannot drive towards all waypoints simultaneously, but must visit them one after another. Switching targets from one waypoint to the next marks the beginning of a new phase, making this a multi-phase task.

A more involved example is that of box stacking. Let there be three boxes labelled A, B and C. To stack these in alphabetical order, with A being the bottom box, the robot must perform a sequence of actions in the correct order. First, it must travel to the location of box B and pick it up. Then, while holding the box, the robot must locate box A and lower box B onto it. Finally, this process is repeated for box C.

Consider also the additional tasks that are only implied. It is taken for granted that the robot must navigate obstacles when travelling between boxes and counter balance to compensate for their weight when lifting them (e.g. a humanoid could fall forwards when picking up a heavy box). The following schedule illustrates the different phases involved in this task and also shows other supporting tasks being coordinated alongside the main box stacking activity.

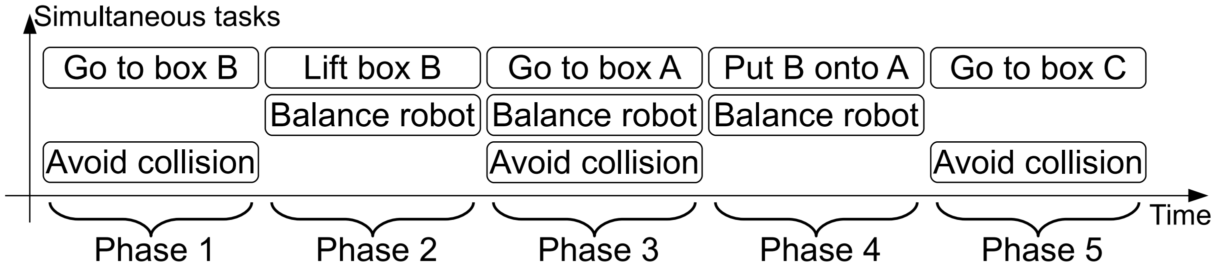


Figure 3.1: Excerpt of a schedule for the multi-phase task of box stacking including the supporting tasks of obstacle navigation (while travelling from one box to another) and balancing the robot (while carrying a heavy box)

Task allocation or planning techniques are required to unpack an implicit task specification and reformulate it in terms of an explicit sequence of smaller, single-phase tasks. While scheduling conflicts may arise in the creation of a multi-phase plan, such as shown above, the resolution of this type of conflict is not a subject of this thesis. The type of conflict we are interested in is that between tasks already scheduled to take place in the same phase. As such, the proposed framework and homeostatic action selection mechanism do not come with an inbuilt scheduler or *native* support for multi-phase tasks. That does not, however, mean that such tasks cannot be coordinated using the developed system. On the contrary, it is designed to do so in combination with an external task allocation mechanism. Single-phase task coordination is the first priority, because it is a necessary requirement for multi-phase coordination. With the ability to simultaneously coordinate multiple conflicting tasks within the same phase, any sequence of such phases can be solved by repeated application of the developed technique.

Furthermore, there is already an abundance of work on task allocation [69] and planning [112] whereas there are very few contributions in the area of simultaneous multi-tasking (see 2.2.3). Logistics can be worked out using existing symbolic reasoning, scheduling or constraint solving techniques. There are also specific solutions to well known problems such as travelling salesman or the "Tower of Hanoi" [70] puzzle, which may be seen as a more complex version of box stacking.

3.1.2 Single-phase Tasks

Single-phase tasks have already been identified as the components of multi-phase tasks (see individual phases in figure 3.1). They represent units of work for the robot being used. Further sub-division into sequential elements is either impossible, impractical or simply of no benefit.

In the context of mobile robotics, the single-phase tasks one can reasonably expect to solve are, broadly speaking, concerned with changing either the state of the robot itself or that of the world around it. A further distinction is possible based on whether the tasks have intrinsic or extrinsic value. The former are characterised by specific, well-defined goals while the latter seek to avoid undesirable conditions that may arise as a result of pursuing those goals. Table 3.1 summarises the proposed taxonomy and outlines the types of tasks that belong into its four categories. A few paragraphs are given over to describing each of these in more detail with the support of concrete examples.

	Tasks concerning internal state	**Tasks concerning external state**
Tasks with intrinsic value	Category 1 State transition between given initial and target configurations	Category 2 Manipulation of the robot's external environment
Tasks with extrinsic value	Category 3 Avoidance of a region of configuration space representing undesirable conditions	Category 4 Avoidance of situations defined with respect to the environment

Table 3.1: Taxonomy of single-phase tasks distinguishing between tasks concerning internal or external state variables and possessing intrinsic or extrinsic value

3.1.3 Category 1: Configuration Transition Tasks

Configuration transition tasks have intrinsic value and concern the internal state of the robot.

Tasks with intrinsic value have a clearly defined purpose that can be expressed as reaching a specific target. The target is given in terms of a desired state which, in this category, is a description of a robot's internal parameters or, more precisely, its configuration. This target configuration, together with the robot's initial configuration, sufficiently specifies any task in this category. The initial or start configuration describes the state of the robot when commencing a task and may or may not be freely chosen. Since the robot can only be in one state at a time, there can be no unknowns remaining in the initial configuration, i.e. the values of all variables are known. The target is a desired future configuration and as such need only specify values for the subset of configuration variables that a task is concerned with. Of course, when the robot arrives at the target all variables will be fully assigned – even those we did not care to specify.

A typical example of tasks in this group is travelling from A to B, where A and B are locations that can be expressed as coordinates which are part of the robot's configuration. Say you start in St Andrews in a car facing south. This specifies an initial configuration of (56.34°N, 2.8°W, SOUTH). Assuming you want to drive to Edinburgh and do not mind in which orientation you park, your target configuration is (55.95°N, 3.2°W, ?) with the question mark denoting the freedom to choose the orientation at the destination. Once you arrive, you can of course only park facing one direction, thus fixing the final orientation, e.g. to (55.95°N, 3.2°W, EAST). Solving such tasks automatically involves an AI controller finding a sequence of controls to make the transition between the initial and target configurations. In our example the control sequence would be a detailed description of how to drive from St Andrews to Edinburgh. Generally speaking, tasks in this category are concerned with state transition of the robot itself.

In addition to attaining a target location (using any propulsion system), tasks in this category include moving parts of the robot (flexing an arm, bending a knee, turning the torso), orienting a sensor to face in a particular direction or regulating internal parameters (e.g. adjusting fan speed to regulate CPU temperature).

Another internal state variable is the robot's remaining battery life. Recharging will typically be a multi-phase task involving driving to a charge point (category 1) and then connecting to the power supply (category 2, see below).

3.1.4 Category 2: Environment Manipulation Tasks

Environment manipulation tasks have intrinsic value concern the external state of the robot.

The external state of a robot can be formalised in a similar way to its internal counterpart. While the configuration describes the state of a robot itself, the external state describes the environment around it. Unless a map of the surroundings is provided, a robot must rely on sensor information to build its own model of the world it inhabits. Whether provided or constructed, this model contains information about all known features of the environment. Typically, these will include the locations of obstacles, people, objects the robot can interact with, other robots, etc.

Tasks in this category seek to interact with, or manipulate, the perceived environment. Examples include grasping an item, moving an object (e.g. shunting boxes by pushing or pulling), kicking a ball, pressing a button, connecting a cable (e.g. the robot's charger) or opening a container. Some tasks may be defined in terms of the environment without seeking to change it directly. Examples of this are target-chasing or acting as a goal keeper by blocking off a region of the environment.

3.1.5 Category 3: Configuration Avoidance Tasks

Configuration avoidance tasks have extrinsic value and concern the internal state of the robot.

Tasks with extrinsic value are given rise to by tasks with intrinsic value. The need for solving the former is a direct consequence of pursuing the specific goals of the latter. While extrinsic tasks lack such specific targets, they are by no means less important. Rather than attaining a certain state, these tasks aim to avoid a whole range of states that are deemed undesirable.

In the context of internal state, we are concerned with configurations that need to be avoided. If the robot were to remain stationary in an acceptable configuration, there would never be any danger of encountering an undesirable configuration. Unfortunately, a static robot is rarely of much use. We have already established that solving tasks with intrinsic value necessitates state transitions, and it is as a result of those that the robot may pass through an intermediate configuration that is off-limits.

Tasks in this category aim to prevent the robot from entering any dangerous configurations on its way to attaining the goals set by other tasks. Examples of dangerous states to be avoided include unstable or imbalanced poses which cause the robot to fall over, getting one actuator tangled in another and preventing the robot from damaging itself with its own tools (e.g. a welding attachment). Energy or fuel rationing is also part of monitoring internal state. A task could, for instance, prioritise the shortest path or make sure the robot travels at the most energy-efficient speed to make sure that it can reach the next charge point before running out.

3.1.6 Category 4: Situation Avoidance Tasks

Situation avoidance tasks have extrinsic value and concern the external state of the robot.

Tasks in this final category are very similar to those in the previous in that they also arise due to the pursuit of other goals. Solutions likewise require constraints on the robot's motion. The main difference is that, rather than avoiding a region of configuration space, these tasks are designed to avoid more complex situations that cannot be defined in terms of a robot's configuration alone. Instead, the situations to be avoided are specified with respect to features of the perceived environment.

Obstacle avoidance is representative of the tasks in this category. In this example, the robot must be kept at a safe distance to any objects with which it might collide. Measuring and maintaining an appropriate distance requires knowledge of the locations of potential obstacles which are part of the external state. Another example may be for the robot to slow down or stop moving when it senses that a human operator is dangerously close to it.

3.1.7 Solution properties

The task coordination framework that these definitions are leading up to is not designed for tasks that do not fit into one of the four categories of the expounded taxonomy. One may however be tempted to add modifiers, attributes or requirements to eligible tasks. For example, the task of transitioning from an initial pose to a target stance qualifies as category 1. A solution may already exist but exhibit the cliché jerky motion often associated with robots. Should this not be acceptable, one might seek to add an adverb such as *gradually* or *smoothly* to the task description, which may then read "smoothly transition from sitting to standing". The following explores how tasks like these may be interpreted and how their solutions can be approached.

The most fundamental question to be answered is whether this is still the same task or if "move smoothly" is a separate, additional task. Either interpretation is tenable, but taking the latter view of splitting the task into two would lead to disqualification on the grounds of no longer fitting the taxonomy: The additional task of moving smoothly can be classified as having intrinsic value, but neither internal nor external state can capture the notion of smoothness. That is because smoothness is a quality which cannot be seen from, or ascribed to, a single state which is, after all, just a snapshot of one moment in time. Many solution qualities, smoothness included, are by contrast defined over an interval of time. They describe properties of the solution as a whole rather than an intermediate state that is part of a desired solution.

This last insight, that qualities are properties of the solution, supports the alternative interpretation that does not require the introduction of an additional task and thus avoids being disqualified. By shifting the burden of dealing with the desired qualities from the task specification to the solution, we can retain the original description of the task. We now view the original task as having multiple solutions that are distinguished by their properties. This is in contrast to the previous view that there are different tasks with exactly one solution each. In our example there might be solutions for standing up quickly, gradually, jerkily, etc. and we select or create whichever solution we desire. The task remains the same.

More abstract solution properties such as completeness and optimality may also be specified. Completeness is the quality of always being able to find a solution, providing one exists, while optimality guarantees that the best solution will be found. Such properties pertain to the solution method or algorithm, not just a single solution produced by it.

Of course, the freedom to specify any desired property does not imply the existence of a solution that provides it. Some combinations of tasks and solution properties simply cannot (yet) be solved. However, this is no longer a problem associated with the task specification. It is simply a sign of the limitations on what can be achieved. Realistic requirements are discussed next.

3.2 Requirements

Motivations for gaining the ability to perform SMT have already been laid out in the introduction. The focus now shifts to designing a system with the desired features and properties. Specifically, this means ensuring that the coordination mechanism developed here is generic and capable of producing viable controls for a real robot in real-time. For this purpose the following requirements must be considered in its design and implementation. It should be noted that the high level of realism required of the solution contributes significantly to the difficulty of its development. The motivation behind this is to prepare the system for the difficult transition to working with physical robots, as suggested in the future work section.

3.2.1 Generic Architecture

Having lambasted existing techniques for being limited to solving a fixed set of tasks, the suggested alternative must deliver on its promise of being generic. The aim is to allow any task fitting the taxonomy to be coordinated. By abstracting over the specifics of the tasks in question, they become interchangeable, thus allowing a *plug and play* style architecture to be implemented. The controller need not even know which tasks it is solving and is therefore generic by design. Of course, this precludes the use of task-specific knowledge or equations – at least within the control mechanism itself.

3.2.2 Hardware Abstraction

It is also useful to abstract over the robot's hardware. This allows the system to be used with different types of robots whether limbed, wheeled or otherwise propelled. The difficulty with this requirement lies in dealing with the different sets of controls and commands applicable to the large variety of available robots. Many approaches avoid this problem at the cost of reusability.

3.2.3 Realisable Controls

Generated controls and control sequences must be realisable by the robot executing them. Planning geometric paths is insufficient as they can only be followed by omni-directional vehicles travelling at low speed. A realistic model of motion must consider a number of constraints.

To begin with, the robot cannot be modelled as a particle. Its heading must be considered and consequently planned paths must have continuous tangents. Some vehicles have the ability to turn on the spot when stationary, but coming to a complete stop is often inconvenient or altogether unacceptable. Most vehicles, including cars, are simply unable to change their heading without performing a manoeuvre such as a three point turn.

Contrary to another popular simplification, real vehicles are unable to change speed instantaneously. This would require infinite acceleration or deceleration when, in reality, both are clearly finite. As a corollary of this, paths must have continuous curvature, since a discontinuity would also require an infinitely fast change in controls. Imagine driving in a figure-eight motion consisting of two circular pieces of track. At the joining point of the two circles one would have to instantaneously turn the steering wheel from hard left to hard right or v.v. and a differential drive robot would have to instantly change its wheel speeds.

Finally the vehicle's Minimum Turning Circle (MTC), i.e. the radius of the sharpest possible U-turn, must be considered. The turn radius depends on the vehicle itself as well as on its speed of travel. When moving at very low speeds, the limiting factor for most wheeled vehicles is axle length (or the distance between the wheels in case of differential drive with a restriction to forwards-only motion). As speed increases, so does the turn radius. This dynamically changing constraint must be considered during path planning.

3.2.4 Real-time Computation

To allow a robot to travel at reasonably high speeds, its controller must produce the next controls in real-time, i.e. while the robot is moving. The faster the robot travels, the less CPU time is available for planning a move of a given length. When multiple tasks are involved, the time interval available for planning each step must be shared among them. As a result, time is scarce and efficiency all the more critical. This requirement sets a low threshold on the permissible complexity of algorithms, thereby ruling out a number of possible solutions.

3.2.5 Autonomous Adaptation

The robot must be able to adapt and react to changes in its environment using feedback from its sensors. Without this ability, the controller would not be sufficiently autonomous causing it to fail in the same sorts of scenarios as scripts and macros. When possible, the robot should be able to solve the tasks it is given despite perturbation or changes in the outside world.

3.2.6 Unknown Environments

The robot and its controller are to rely on sensor information alone. While sensors are already required for adaptation, a map (or model) of the environment is often supplied in addition. Denying the AI such information makes it harder to find a solution but will ensure it is applicable in completely unknown terrain. This does not mean the robot is debarred from recording sensor information or building up its own model, just that it is not given any a priori knowledge.

3.3 A General Framework for SMT

The first step in constructing a suitable task coordination mechanism is to design a framework that can provide the features needed to satisfy the above requirements. The components making up the proposed framework are illustrated below. The diagram gives an overview of all top-level modules and the relationships between them. A detailed explanation of each component can be found in an eponymous section below.

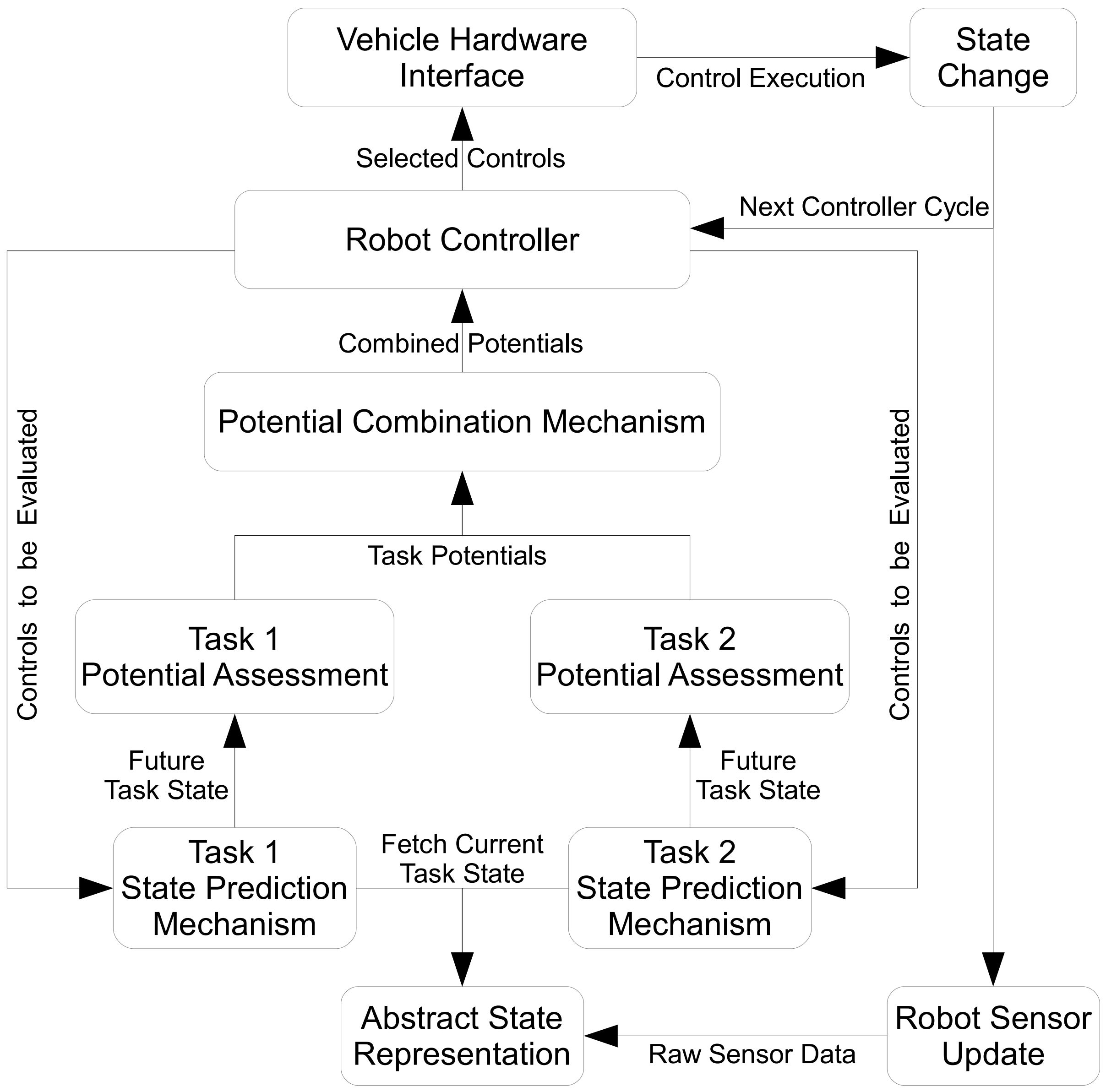


Figure 3.2: Framework for task-agnostic SMT with reversed flow of control compared to the classic behaviour-based control loop (any number of tasks can be added, but only two are shown in the interest of clarity)

Particular emphasis is placed on the requirement of keeping the system generic. The easiest way to ensure genericity and interchangeability of tasks is to design the components of the system from an abstract, top-level view. Using a high degree of abstraction prevents any low-level, task-specific details from seeping into the coordination mechanism itself. It also leads to a modular design with interchangeable components that can be modified and improved without having to change unrelated aspects.

Task-specific details deserve attention as well, but will be addressed separately in the upcoming chapters that deal with each of the selected demonstrator tasks. For now we assume that *conceptual tasks*, as one may find described in a problem specification, can be decomposed into tasks that fit the given task taxonomy.

In line with the second hypothesis, we further assume the existence of (or at least the possibility of creating) a state prediction and potential assessment mechanism for each taxonomy task. Access to these two functions provides the sole basis for decision making in a system that is otherwise completely decoupled from the tasks it is coordinating. Consequently, coordination success heavily depends on these functions to guide search towards controls that constitute a suitable compromise between all competing tasks. The potential function, especially, must be a reliable indicator of how to progress and ultimately solve the task it represents. No other indicator of success can be made available without breaking the abstraction that enables generic task coordination.

The chosen architecture is loosely based on the classic behaviour-based control loop, but also makes some significant modifications. Most notably, the flow of control between some components has been reversed. Normally, State Change triggers a response in the behaviour modules, which then submit their preferences concerning a variety of possible actions to a central action selection unit. Here, the Robot Controller is notified of the state change and initiates the rating process. Possible actions are generated by the controller and passed through the task modules for rating. The result is returned back to the controller, which now has the power to coordinate and guide the search as opposed to taking a passive role and processing only the information it is provided. In this way, search can be directed towards or concentrated on regions of control space which are more likely to hold good solutions.

Other ASMs, specifically those reviewed in the Relative Weighting category, could also benefit from being integrated into the outlined framework.

3.3.1 State Change

State change occurs continuously as the robot executes the controls it is passed. The controller works in discrete time steps however, as it requires a certain amount of time to process sensor information and select controls. A new time step begins when the control system is first initialised and, from then on, every time new controls are submitted for execution. As can be seen from the framework diagram in figure 3.2, the start of the next controller cycle is an event that causes two things to happen.

Firstly, the robot's sensors will be updated to reflect the new state of the perceived world (see Robot Sensor Update). Secondly, the action selection mechanism is set in motion to adjust the robot's actions in line with the updated sensor information (see Robot Controller). The framework's components will be described starting with the train of events caused by the sensor update and will then return to the action selection mechanism, i.e. the robot controller.

3.3.2 Robot Sensor Update

At the beginning of each time step, a snapshot of all sensor readings is taken. The raw sensor data is passed to the Abstract State Representation where it is used to build a temporary model of the environment. The robot relies on this information alone, i.e. it is *not* provided with a priori knowledge of any kind. Consequently, the robot must be programmed to cope in unknown environments as per the requirements.

Satisfying the requirement of being able to adapt to sensor information is the responsibility of the developed AI, but the provision of said data is a prerequisite for it to function. Since this project is not concerned with perception related issues, the existence of a sensor for all relevant state variables is assumed. In reality, it may, for instance, be unrealistic to expect a GPS sensor to provide localisation with sufficient accuracy without the additional support of odometry. For the purpose of evaluating the coordination mechanism, the details of how the current state is determined is not of import however.

3.3.3 Abstract State Representation

The raw sensor data recorded is used to construct an *abstract state representation*, which is a model of the perceived world including both internal and external state variables. Depending on the implementation, the sensor data could simply be collected, but more often it is useful to process it to some degree. For example, sonar sensors are used to measure the distance to obstacles, but do not directly indicate the position of the obstacle hit point (where the sonar ray is reflected by the obstacle). In this case, it can be useful to compute and store the hit point instead of, or in addition to, the raw distance measurement. Additionally, one may want to occlude gaps between obstacles that the robot is too large to pass through. Storing metadata to indicate which gaps should be closed off saves having to repeat potentially costly computations.

Once all raw sensor data has been processed, a classical behaviour-based design would disseminate that information by pushing it to the task modules, which would in turn initiate communication with the controller. Here, we return control directly to the controller, merely saving the data to be queried later. This is why there is no arrow leaving the Abstract State Representation component. Instead, there is an incoming arrow from the Task State Prediction Mechanism, which actively retrieves any data it requires.

The state representation is made available to all components of the framework on a read-only basis. Doing so allows the model to be queried from any number of threads or processes without causing concurrency issues.

An alternative to such a centralised data repository would be to use a separate model for every task. Each model would then only comprise the state variables relevant to it. This follows the more traditional behaviour-based architecture, but was found to be an unnecessary complication. Specifically, it leads to duplication of information and some derived values having to be recomputed for every task that is concerned with the same state variable.

3.3.4 Robot Controller

The *Robot Controller* is the central component of the proposed framework. Its job is to find and select the controls representing the best realisable compromise for progressing all tasks simultaneously. It does so by minimising an indicator of conflict that combines the different individual costs ascribed to candidate controls by each task. The Potential Combination Mechanism is responsible for computing this overall indicator of quality (or lack thereof). For now, it suffices to assume the existence of such a mechanism, as its details would only distract from the description of the controller itself. The following discusses possible search algorithms and the discretisation of continuous time into time steps. Since the framework can be implemented in different ways, the discussion takes a top-level view. A specific implementation of this component can be found in Chapter 7.

3.3.4.1 Search Algorithms

Controls must be produced in every time step, i.e. each cycle of the closed-loop control system. As already mentioned, a new cycle begins with the submission of new controls to be executed. Apart from leading to an update of the robot's sensors and the Abstract State Representation, this event also initiates the search for the next controls. In essence, all that is required is an algorithm for finding controls that score low on conflict, i.e. the combined task potential. Search can be implemented in different ways due to the modular design of the framework.

When coordinating relatively harmonious or benign tasks, gradient descent over control space may be a viable option for quickly deciding a suitable action. As the number of tasks and/or the conflict between them increases, the potential surface will become increasingly rugged and unamenable to descent. There are numerous ways of addressing this problem. One very interesting approach would be to try and adapt the "IvP" method proposed in [9], as already remarked in the literature review.

For the purpose of demonstrating the proposed mechanism, a sampling based approach is sufficient however. The implemented sampling controller (see 7.2) generates control samples within the valid region of control space, that contains only realisable actions. Generated controls are passed to the Task State Prediction Mechanism, thereby initiating a process that produces their final combined score.

Each sample represents making a single step forwards. The outcome of this move is evaluated without planning a control sequence that takes the robot all the way to its goal(s). The advantage of this technique is that evaluation is fast, allowing a sufficient number of alternatives to be considered in real-time. A possible disadvantage is the short-sightedness of the plans that are formed. This thesis postulates that a single-step planner is capable of producing satisfactory solutions despite planning only one step ahead. Chapter 7 investigates the truth of this statement in order to verify the fourth hypothesis. After evaluating as many samples as possible in the given time frame, the controller simply selects the controls associated with the least overall conflict.

It should be noted that the requirement for realisable controls is trivially satisfied by only generating samples that respect the vehicle's static and dynamic motion constraints. The controller can only select actions it has sampled and all samples are valid, therefore the selected controls must also be valid.

3.3.4.2 Synchronisation and Time Steps Length

Running the search algorithm will take a certain amount of time, leading to the question of how long a time step should be.

Synchronisation
In fact, the first question to be asked is why a time step is even needed. Theoretically, the system could operate asynchronously, with each task generating its own events that are processed as soon as the controller is notified of them. This model does not involve temporal alignment of

the evaluation of task preferences and hence requires no discretisation of time. Indeed, such an event based system would be ideal for single-task operation. It guarantees that no CPU time is wasted and allows the robot to react to new events immediately.

The simultaneously coordination of multiple tasks requires the preferences of all tasks to be considered in the selection of new controls. When those preferences are recorded at different times, a fair comparison is no longer possible. Many tasks are of a time sensitive or volatile nature, meaning that the validity of their preferences rapidly expires. To maximise the relevance of a comparison and to strike a fair compromise, decisions must be made based on sensor data recorded and evaluated at the same time. This is achieved by the use of synchronisation and discrete time steps.

Time Steps Length
A number of different policies can conceivably be used to decide the length of a time step.

The simplest approach is to wait for the controller to finish its search. This makes for a variable time step length and has the advantage that search is guaranteed to complete with the set precision. The problem is that time-critical tasks may require more frequent adjustments to the controls being applied. If the controller takes too long to complete its computations, such tasks could fail before the AI even has a chance to react.

To make sure this cannot happen, the time step length can be set to a fixed duration that is appropriate for the most time sensitive task. Search would then ideally use all available time to refine its results. When the allotted time expires, the controller selects the best controls discovered so far. Reactivity is maximised by choosing short time step lengths, while accuracy increases with duration. An ideal trade-off could be determined by running the same experiment multiple times with different values. It is, however, to be expected that the ideal value will be specific to the problem the robot is set. For good all-round performance, one would have to compare time step lengths across numerous experiments or manually set a time that suits the most volatile task.

The above policy still uses a fixed cycle length throughout the course of an experiment. Hypothetically, the length of a time step could be changed dynamically, while the robot is active. The difficulty here is the lack of a benchmark for comparison. The robot does not know how well it is doing until it completes all its tasks and compares its performance with previously completed experiments that used different interval lengths (all other parameters being the same). Another consideration is the time one would have to invest in computing the ideal cycle duration. The time it takes to compute that ideal value will be deducted from that available to the actual controller. This will likely negate any potential benefits and is not seen as a realistic option.

The second of the above policies is quite clearly the most promising, but none of the techniques is very conducive to testing. Whether precision is fixed and the time step length is variable or vice versa, there is a nondeterministic element to the controller. For tests to be reproducible, the system must be deterministic. To achieve this, a fixed time step length is set and the precision of the search is adjusted to guarantee its termination within that time. Because the search algorithm must complete even under worst case conditions, more time must be allowed than is required on average. Any remaining time goes to waste but, for the purpose of testing, this is a price worth paying and the reason why this last policy was chosen.

3.3.5 Task State Prediction Mechanism

The provision of a state prediction mechanism is part of the preconditions laid out in the second hypothesis. In order to reason about a wide range of tasks generically, a two-part abstraction needs to be created. State prediction is the first part of this. It provides a hardware abstraction that allows different robots to be treated the same way without the controller even having to know what type of vehicle or actuator it is controlling. This is achieved by defining a mapping from controls to the internal or external state change their execution brings about.

Forward kinematics are an example of state prediction for a moving vehicle. Since other tasks may revolve around different state variables, *state prediction* was chosen as a more general term that does not mislead the reader by implying that all tasks must be concerned with the kinematics equations of a vehicle. Chapter 6, for instance, develops a physics model for predicting forces acting on a rolling ball.

Candidate controls are passed to each task's state prediction mechanism as a first step towards determining how beneficial their execution is likely to be for that task. The mechanism makes an informed estimate of the future state the robot would assume if the given controls were executed and passes this on to the next component, which considers the prediction as part of Task Potential Assessment. Each task only makes a prediction regarding the state variable(s) it is concerned with. In the example of obstacle avoidance, only the controls that influence the location of the robot are of interest. State prediction, in this case, involves using the vehicle's forward kinematics formulae to determine the end configuration of the transition defined by the given action. The predicted end configuration can then be used to calculate the distance to the nearest obstacle – a value this task's potential depends on.

How exactly state prediction works (and how task potential is assessed) is specific to each task. Literature on action selection often glosses over the details of individual tasks, but here a description is necessary as these two processes represent a central part of the theory and mechanism developed. Since it is not possible to describe state prediction in an abstract way, a chapter is devoted to each of the three demonstrator tasks used to evaluate the framework and its homeostatic variant.

For the remainder of this chapter, state prediction is assumed to be provided by a state function S of the robot's current state cs and given controls c. These inputs are mapped to a predicted future state fs as shown.

$$fs = S(cs, c) \tag{3.1}$$

The relevant current state information is obtained directly from the Abstract State Representation.

3.3.6 Task Potential Assessment

The Task State Prediction Mechanism provides the first part of the abstraction required for generic, task-agnostic coordination. The second part of that abstraction is provided by the *Task Potential Assessment* unit, which defines a function P for mapping any predicted future state fs to its associated potential.

$$potential = P(fs) \tag{3.2}$$

The term *potential* is used in the same way as in Khatib's "Potential Fields" methods [66]. Potential, objective or cost functions (used synonymously) map a given action to the utility a task can expect to receive upon its execution.

While state-prediction functions abstract over robot hardware, potential functions abstract over the specifics of the individual tasks being coordinated. Having each task define and evaluate its own objective value allows the controller to judge the progress of any task based on the output of this function. In other words, it need not know about, or consider, the factors which contribute to a task's satisfaction level.

The composition of these two functions provides a mapping from a current state and a set of controls directly to the potential rating for a particular task.

$$potential = (P \circ S)(cs, c) = P(S(cs, c)) \tag{3.3}$$

This mapping allows search to take place in control space, whereas most objective functions are defined over state space.

Ways of assessing potential, or rather its homeostatic counterpart (urgency), are discussed in the task chapters to follow. For now, and in line with the second hypothesis, we assume the existence of an appropriate function.

3.3.7 Potential Combination Mechanism

As already mentioned in the description of the Robot Controller, the role of this component is to combine all individual potentials into a single measure of conflict. Suitable compromise controls can then be found by minimising the conflict with which they are associated.

The framework can be implemented to imitate existing ASMs in the Relative Weighting category. Khatib's "Potential Fields" [66] approach could be trivially replicated by computing conflict as the sum of all contributing potentials. Reproducing Rosenblatt's "Distributed Architecture for Mobile Navigation" [101], requires a weighted sum.

A more sophisticated mechanism is called for to overcome the known problems with these existing methods. The proposed solution, Homeostatic Mortality Computation, is presented in the following section. Here, we explore generally what properties a conflict rating scheme should possess to inform and facilitate the selection of the most promising compromise actions.

Firstly, the rating mechanism must not influence decision making unless the presence of contentions requires it to effect a compromise. When all tasks agree on a desired action, or in the trivial case where only one task is currently active, the controls corresponding to that action must be selected without alteration. It is not acceptable for the compromise finding mechanism to arrive at a needlessly inferior solution due to a distortion in the ranking of candidate actions. In short, the system must be transparent in the case of agreement.

Secondly, a useful rating cannot hinge on the potential contributed by a particular task. The coordination mechanism is responsible for resolving contentions between adversarial tasks, but the presence of a particular task (let alone a synergistic or symbiotic one) cannot be assumed in a plug and play system. The task-agnostic architecture does not expose knowledge of which other tasks are being pursued. Consequently, each task's potential function must be sufficient to solve the task for which it was constructed when no other tasks are active. The coincidental presence of cooperative tasks is beneficial, but cannot be essential to successful multi-tasking.

3.3.8 Vehicle Hardware Interface

The vehicle hardware interface has the sole purpose of actioning the controls it is passed by the Robot Controller. Execution takes the robot to a new state and thereby completes the closed control loop, which then enters its next iteration to repeat the entire process. When the goal state is reached, the controller simply stops sending commands, thus terminating the loop.

3.3.9 Computational Complexity

Different implementations of this task coordination framework will have different computational complexities, depending on the search algorithm used. It is, however, possible to examine how the number and complexity of individual tasks influences overall runtime.

To do so, we assume that the search algorithm will evaluate n control vectors. For every one of these, each of k tasks will perform task state prediction and potential assessment. Finally, the Potential Combination Mechanism is run once per control vector. The runtime of one cycle of the control loop is then as follows:

$$runtime = n \ (S_{T1} + P_{T1} + S_{T2} + P_{T2} + ... + S_{Tk} + P_{Tk} + PotentialCombination) \tag{3.4}$$

where S_{T1} denotes the runtime of state prediction for task number one and P_{T1} is the runtime of the algorithm used to assess its potential.

For worst case analysis, assume that the state prediction and potential assessment mechanisms of all tasks operate at the speed of the slowest, most complex task.

$$runtime = n \ (k \ (S_{worst} + P_{worst}) + PotentialCombination) \tag{3.5}$$

To further simplify, some assumptions can be made that are likely true for most implementations. First of all, the Potential Combination Mechanism will typically involve the evaluation of a fairly simple equation with one term per task, e.g. a weighted sum. The complexity of this is linear in k and negligible compared to the other computations.

Another fairly safe assumption is that state prediction will be less complex than potential assessment. The forward kinematics of a vehicle are, for instance, expressed by a set of equations that can be evaluated in constant time. Potential assessment cannot beat constant time and will often involve some form of iterative refinement. Given these assumptions apply, complexity can be expressed using big O notation as follows:

$$O(n \ k \ P_{worst}) \tag{3.6}$$

Since complexity is linear in k, the system should scale well with the number of tasks. The complexity of potential assessment depends on the individual tasks being coordinated and cannot really be influenced. It may be possible to use approximate solutions to save time however. The biggest issue is with the number of control vectors the search algorithm needs to evaluate. This will depend on the robot's degrees of freedom, i.e. the dimensionality of control space. A sampling approach will quickly become intractable, which is why other methods, such as [9], should be considered in high degree spaces.

Another avenue to improving on runtime is to make use of the ample opportunities for parallelisation. It is possible to evaluate each control sample in a separate thread without encountering any concurrency issues. Even within the evaluation of a single sample, one could use multiple threads to evaluate the potentials of different tasks concurrently.

Finally, there are strategies for reducing the number of samples needed to find a high precision solution. This could be achieved by biasing sample generation towards areas of control space that are more likely to contain solutions superior to those already found. For example, one could first evaluate a small number of widely scattered controls and then focus search in the vicinity of the best of those samples. Optimality cannot be guaranteed in this way, but an exhaustive search of a high dimensional space is out of the question anyway.

3.4 Homeostatic Mortality Reduction

This section proposes a homeostatic mechanism for simultaneous multi-tasking that addresses the shortcomings of existing techniques described in the literature review. Homeostatic Mortality Reduction is a specific implementation of the above framework. It implements a rating based approach that attends to all tasks to a varying appropriate degree as justified by their urgencies. Urgencies are homeostatic indicators, defined on a universal scale, which provides a fair means of quantifying and comparing the threat posed to each task as a result of taking any permissible action. To rate a candidate action in terms of its coordination quality, individual urgencies are combined into a Homeostatic Mortality Index (HMI), which reflects the overall risk of failure associated with executing that action. The mechanism for assessing and reducing mortality is detailed in the following by explaining the modifications made to the framework to integrate the desired features.

Specifically, Task Potential Assessment has been replaced by Task Urgency Assessment and the role of the Potential Combination Mechanism is now performed by the Homeostatic Mortality Computation unit. The homeostatic incarnation of the framework is shown below, with modifications highlighted in red. All other components remain as described in the previous section. The roles of changed components are explained in the following. This section culminates with an Example of Homeostatic Task Coordination to illustrate a simple application of the introduced concepts.

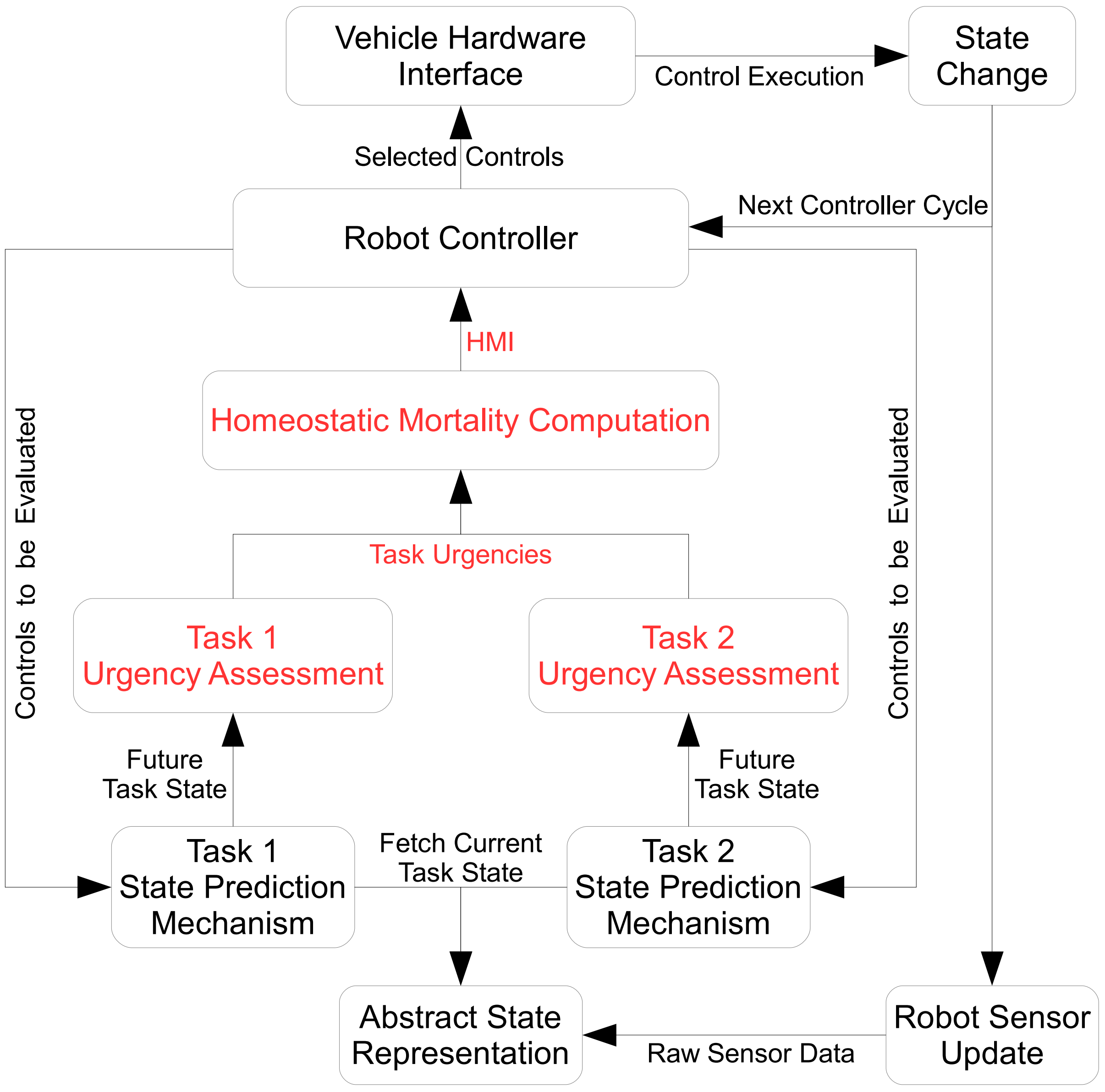


Figure 3.3: Homeostatic implementation of the task coordination framework for simultaneous multi-tasking

3.4.1 Task Urgency Assessment

Each task has its own urgency assessment unit for mapping the state variables it is passed to an urgency value indicative of that state's proximity to task failure. The following explains what urgency is, defines the range of values it can take on and discusses different ways of assessing it.

3.4.1.1 What is Urgency?

Urgency can be thought of as the homeostatic counterpart to a task's potential, cost or objective value. As per the interpretation of homeostasis as the *attention to threat principle*, urgency should reflect the threat a task is under. In the biological world, from which this concept is taken, threat quite literally pertains to the danger of an organism dying. Since the robot is not alive to begin with, the closest equivalent to organ failure is task failure. On the opposite end of the spectrum is perfect harmony. Biologically this means that all organs can perform their intended purposes and are able to keep the organism's vital signs in check. Equivalently, robot tasks are also stable and in harmony when they can pursue their intended purposes. These extremes delimit the range of homeostatic urgency and define a universal scale on which all tasks can be compared.

The challenge is now to find a strategy for projecting task-specific indicators of success (or failure) onto the urgency scale. As long as all tasks rate urgency on the same scale, the range of the urgency mapping can be chosen freely. To make matters easier, the interval $[0,1]$ can be chosen without loss of generality and will be used throughout this thesis. Zero indicates the absence of any threat to the workings of a task, while a value of one represents failure.

The definition of the urgency mapping is, of course, task-specific and therefore a matter to be discussed in the task chapters. General strategies for obtaining urgencies may already be explored however.

3.4.1.2 Direct Urgency Assessment

Broadly speaking, there are two approaches to assessing urgency. The mapping from a state to urgency can be provided by a dedicated mechanism developed from first principles or defined in terms of existing metrics, such as task potential.

There are no restrictions on how to implement direct urgency assessment. One way of going about creating the required mapping is to train a neural network to classify task states given to it as inputs. This would create a direct association between a state and the threat it poses to a certain task. A separate neural network would be required for each task, but would only have to be trained once. The problem with using machine learning at the coordination level, whereby retraining is required every time the number of combination of tasks changes, is not encountered. This is because, at the task level, the system is only aware of individual tasks. Other tasks may as well not exist and could not be considered even if one wanted to. The coordination level will be handled by the proposed homeostatic mechanism, no matter how the task layer is implemented.

Other techniques may be based on probabilities, heuristics or other types of learning. While all of these approaches could provide effective solutions, they would have to be created from first principles.

3.4.1.3 Urgency From Potential

To take advantage of existing solutions, this thesis explores ways in which urgency can be defined in terms of task potential. Functions for assessing potential have already been developed for many popular robotics tasks. Being able to reuse them could save a lot of time and effort.

Whichever path is taken, the computational complexity of the mechanism for urgency assessment must be considered, as this is a likely bottleneck. Real-time control necessitates the evaluation of a large number of samples in a very short space of time. This is a further incentive to use potentials, which are often based on heuristics that can be evaluated in much less time than it takes to find a precise solutions.

A Linear Mapping from Potential to Urgency

The simplest scheme for converting potential to urgency is to apply a scaling or linear mapping. However, the range of a task's objective function cannot be assumed to be bounded. In particular, the maximum value may be unknown or infinite. A direct mapping onto the $[0,1]$ range is then either difficult or impossible, depending on the nature of the potential function.

Urgency as a Distance to the Ideal Case

Another way of defining urgency is as a measure of disparity between a candidate control vector and a task's ideal controls. The ideal controls are found through minimisation of the task's objective function using any appropriate search algorithm together with the Task State Prediction Mechanism. Since these controls represent the best possible action for that task, they are, by definition, associated with zero urgency. The worst possible controls for the current time step can then be defined as those furthest away from the ideal. Control space is bounded, i.e. restricted to the controls that are valid at any given time. This means the maximum distance the candidate controls can be away from the ideal controls is known and finite. The urgency associated with a given control vector is then simply defined as its distance from the ideal controls expressed as a percentage of that maximum distance.

Unfortunately, this approach suffers from the same drawbacks that were found with ASMs in the Preference Interpolation category. Discarding all but the ideal solution means that information is irretrievably lost, which can result in poor solutions or even a compromise that satisfies none of the tasks. Imagine, for instance, that an obstacle can be passed by going either way around it. Going right may be preferable to going left, so the ideal control vector represents a right turn. The alternative left turn is only a slightly inferior solution but, due to being on the opposite side in control space, it is associated with abject failure. This inability to consider alternatives makes task coordination and compromise finding almost impossible. When, as in this example, one task only has a slight preference, another task that has a clear preference should be able to break the near tie in its favour. Doing so is impossible in a scheme where tasks fixate on a very particular solution without good reason.

Urgency as a Distance to the Worst Case

While the described approach does not lead to the desired results, the idea of defining a distance between best and worst cases is not bad in and of itself. The only problem lies in the loss of information resulting from fixing the ideal to a specific control vector. As it turns out, this problem does not occur when instead fixing the worst case and defining it in terms of the maximum value of the task's objective function within the valid region of control space. Returning to the previous example, let us assume the worst action is associated with a cost of 10 units. Going right costs 1 unit and going left 2 units. The right turn is still preferred as it is furthest away from the worst case cost of 10. More importantly though, the left turn is now accurately represented as being almost as good.

Two problems remain concerning the range of urgency values. Firstly, the lower end of the urgency spectrum is hardly used because only controls associated with zero cost also yield zero urgency. Typically, even the best available action incurs some cost, however, and will have a non-zero urgency although, by definition, it should be zero. The second problem is that the upper end of the range is always used because the worst case is defined using a realisable control vector. Samples in the vicinity of this worst case will be associated with very high potential although they do not necessarily represent failure – even if they are among the worst actions possible at that moment in time.

Both problems are solved by the following formula and using a different definition of the worst case.

$$urgency = \frac{sampleCost - idealCost}{worstCost - idealCost} \tag{3.7}$$

By subtracting the ideal (i.e. lowest) cost from that of each sample, the range of values is translated back to zero, meaning that the best sample will have zero urgency. The same is done with the worst case potential in the denominator to ensure the size of the interval between ideal and worst cases does not change. Finally, the definition of the worst case is changed from the cost of the worst possible sample in the current time step to the worst outcome that can reasonably be expected in the near future. This new definition means that only controls that really do pose a lethal threat will be assigned maximum urgency.

Depending on the task, the ideal and worst case costs may be defined very differently. A time consuming search for these values is luckily not always required: The ideal cost can be determined by evaluating the task's objective function for the robot's current state, provided the objective function is admissible, i.e. never an overestimate. Any samples must then have a cost that is greater than or equal to that of the ideal. The worst case is often more difficult to find, but this issue will be addressed for each task separately in the following task chapters.

3.4.2 Homeostatic Mortality Computation

Once all tasks have computed their urgency score for the given controls, the results are passed to the *Homeostatic Mortality Computation* unit. The purpose of this component is to project the whole set of individual urgency values onto a single point in the Homeostatic Mortality Index (HMI). Using this single value, different controls can be compared in terms of the overall risk taken on by executing them. The controls associated with the lowest HMI represent the safest option and should be selected as the most viable compromise.

As previously mentioned, this component replaces the Potential Combination Mechanism. Just like its counterpart, it combines individual ratings into one overall rating. An analogue step is performed in all ASMs in the Relative Weighting category. The difference is that, in those methods, the final rating is nothing other than the weighted sum of individual ratings. Some drawbacks of this approach have already been pointed out, but will be further illustrated as part of the explanation of the more sophisticated and biologically accurate mechanism developed here. The following explores different ways of computing the HMI from an array of urgencies.

3.4.2.1 Simple Addition

To begin with, it is useful to understand the problem with simple addition. Implementing such a utilitarian scheme would indeed lead to least overall urgency and HMI rating. Ironically though, this comes at the cost of potential failure. Pursuing the greater good can mean that individual tasks are overruled and allowed to fail if the momentary benefit to other tasks is sufficiently high. Consider an example with three tasks and two available actions that have the following urgencies and HMI sums:

$$UrgencyArray(action1) = [0.8, 0.8, 0.8] => HMI(summed) = 2.4 \tag{3.8}$$

$$UrgencyArray(action2) = [0.6, 0.6, 1.0] => HMI(summed) = 2.2 \tag{3.9}$$

The second action will be selected on the merit of its lower HMI, thus accepting failure of one of the tasks. If enough tasks favour an action that causes a lone task to fail, the action that serves the majority wins out and the individual task's needs are neglected.

3.4.2.2 Least Squares

Another favourite is the least squares method. In that model, all urgencies are squared before summation, but the problem persists as can be seen using the same example:

$$UrgencyArray(action1) = [0.8, 0.8, 0.8] => HMI(squared) = 1.92 \tag{3.10}$$

$$UrgencyArray(action2) = [0.6, 0.6, 1.0] => HMI(squared) = 1.72 \tag{3.11}$$

A faster growing polynomial may solve the problem temporarily, but more tasks could be added causing it to re-emerge.

3.4.2.3 Exponential Functions

Ultimately, what is needed is a way of mimicking the prioritisation scheme exhibited by all living beings. The survival instinct leads to willingness to take on extraordinary risks in order to keep any vital sign from failing. For example, a gazelle will risk drinking from crocodile infested waters before dying of thirst. In other words, survival may require accepting overall degradation of circumstances in the interest of preventing failure in a single aspect.

To represent the willingness to do anything to keep a task from failing, an exponential function is proposed for mapping urgency values in $[0, 1]$ onto a mortality scale in $[0, \infty)$. Urgency and mortality are essentially the same concept, except that mortality includes the idea of a survival instinct. Any monotonically increasing and continuously differentiable function with the required domain and range is a possible candidate for providing this mapping. What is best is likely to depend on the specific implementation of the robot controller. The following formula (in which u represents urgency) is suggested, but is by no means the only option.

$$mortality(u) = \frac{\mathrm{e}^{\frac{1}{1-u}}}{1-u} - \mathrm{e} \tag{3.12}$$

This function, as well as some of the alternatives that were considered, is plotted in the graph below.

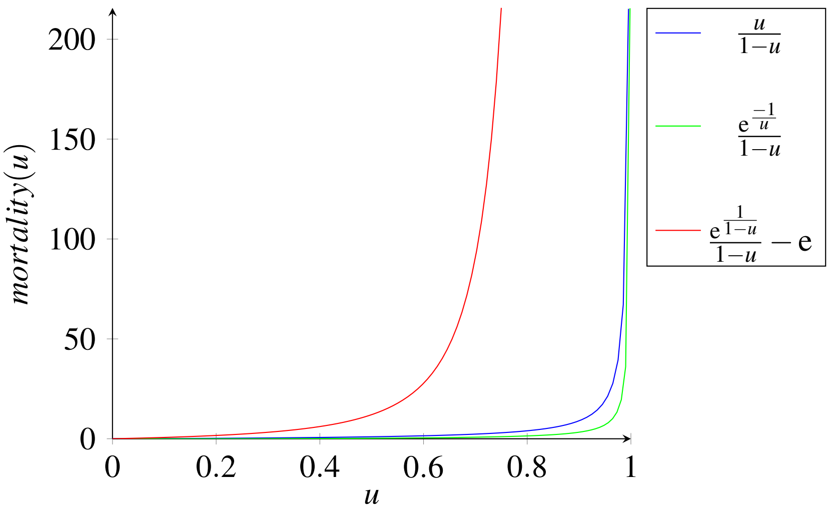


Figure 3.4: Comparison of candidate functions for homeostatic mortality in their [0, 1] domain. All functions tend to infinity as urgency tends to 1. This cannot be displayed without using a very large scale for the vertical axis, which compromises the readability of the graph. For this reason, the displayed range is limited to a value of just over 200.

The blue and green plots are deemed inferior to the red, as they too closely resemble step functions. A gradual transition is required to properly balance the needs of different tasks and progress them simultaneously. With an abrupt jump in mortality, the system would effectively degrade to a task switching behaviour: As long as a task is not about to fail, it is ignored due to its low score. Then, at the brink of failure, the task suddenly demands the robot's full and exclusive attention.

3.4.2.4 Equation for the HMI

Only when urgencies have been converted to mortality, can they finally be summed to yield the HMI. The number of tasks being coordinated is denoted n, and u_i is the urgency of the i-th task.

$$HMI = \sum_{i=0}^{n-1} mortality(u_i) = \sum_{i=0}^{n-1} \frac{e^{\frac{1}{1-u_i}}}{1-u_i} - e \tag{3.13}$$

Of course, there is an HMI for each sample control vector c, the urgency of which also depends on the current state cs. To express this dependency explicitly, the equation for the HMI can be written as:

$$HMI(cs,c) = \sum_{i=0}^{n-1} mortality(u_i(cs,c)) = \sum_{i=0}^{n-1} \frac{e^{\frac{1}{1-u_i(cs,c)}}}{1-u_i(cs,c)} - e \tag{3.14}$$

Once the HMI has been calculated for a given control vector, the result is returned to the Robot Controller, which can now make a fair comparison between all candidate actions. When the controller is satisfied that it has gathered enough information to make an informed recommendation for the controls to be executed next, it passes those controls to the hardware interface. If time runs out before search has come to a natural end, the controls with lowest HMI discovered so far are executed.

3.4.3 Example of Homeostatic Task Coordination

An evaluation of the system described here can be found in Chapter 7. The tasks used for demonstration and proof of concept are developed throughout the next three chapters. Before considering this full scale application, a more compact toy example is presented here. Its purpose is to aid the reader in forming a complete picture of the proposed system and all its components, especially the homeostatic mechanism at its core.

Imagine a therapeutic drug monitoring scenario in which a robot regulates the level of medication in a patient's blood. The correct dosage must be administered and maintained by use of a single linear actuator which depresses the plunger of a syringe. Feedback is provided by a sensor measuring the concentration of the injected solution in the bloodstream. The following describes how this problem can be modelled using two tasks compatible with the proposed framework. While the solution is somewhat over-engineered, this model serves well for illustration purposes.

The first task used in this example is concerned with curing the patient by administering the drug. This is a category 2 task as it has intrinsic value (healing the patient) and concerns a state variable external to the robot (medication level in the blood). Let us further assume that a minimum concentration of the drug is required for it to save the patient and that it becomes more and more effective as the dosage increases. To reflect this, the task's urgency must decrease with increasing concentration.

With only the described task in control, the robot would simply inject the entire content of the syringe as this is the fastest way to combat the illness. Doing so may, however, lead to an overdose, the prevention of which requires a second task. Having arisen as a result of the first task, the second task has extrinsic value and qualifies as category 4, since it also involves the same external state variable. Its objective is to keep drug concentration as low as possible since that is the best way of ensuring there will be no undesired side effects. To reflect this, the task's urgency must decrease with decreasing concentration.

These two tasks are evidently in conflict as they both seek to drive the same external state variable in opposite directions. A compromise must be struck between the extremes of pushing the plunger all the way down and not pressing it at all. In order to determine how much to inject, the mortality reducing controller searches control space for the action with the lowest HMI score. The action selected in this way can be thought of as having the greatest net benefit after considering the different urgencies of both tasks. To come as close as possible to finding the ideal dosage, a reasonable portion of control space would normally be searched. However, in the interest of clarity, this example will only compare three candidate control samples. The following shows how the proposed mechanism evaluates each sample and finally selects the controls to be applied.

The evaluation of a control sample begins with its generation by the controller, which may or may not be able to make an informed guess as to which part of control space to focus its search on. Let us assume that the three control samples to be compared are described by depressing by plunger by zero, one or two millimetres respectively.

The first step in evaluating each of these samples is to make a prediction as to how they will affect the state variable in question. Here, state prediction involves mapping the distance the syringe is depressed (in millimetres) to the medication concentration in the patient's bloodstream (typically measured in moles per litre). This may be achieved in a number of steps. First, the plunger distance is associated with the volume of liquid injected and then that volume is converted to the amount of substance in moles. To get the increase in molar concentration, this amount must be divided by the volume of blood in litres (which may be known or estimated using the patient's size or weight). In the last step, the increase in concentration is added to the current concentration measured in the blood to yield a new predicted concentration level. These steps are shown in the table below which assumes that 2 ml of substance are injected for every millimetre the robot's actuator is extended. Each millilitre of liquid is presumed to contain 0.05 moles of substance and the fictional patient's blood volume is taken to be 5 litres.

Actuator distance	**Volume injected**	**Amount of substance**	**Concentration increase**	**Current concentration**	**Predicted concentration**
0 mm	0 ml	0 mol	0 mol/L	0.1 mol/L	0.1 mol/L
1 mm	2 ml	0.1 mol	0.02 mol/L	0.1 mol/L	0.12 mol/L
2 mm	4 ml	0.2 mol	0.04 mol/L	0.1 mol/L	0.14 mol/L

Table 3.2: An example of task state prediction for the described therapeutic drug monitoring scenario using arbitrary conversion factors (1 mm $\rightarrow$ 2 ml, 1 ml $\rightarrow$ 0.05 mol, blood volume: 5L)

The predicted concentration is then passed to each task's urgency assessment mechanism, which uses task-specific knowledge to rate the threat posed by entering the given state. Depending on the task, rating may be a very complex process, but, thanks to a layer of abstraction, the coordination system need not know any of its details. In this case, a simple solution would be for a doctor to provide a lookup table showing the dangers associated with too low a dose for the first task as well as the risk of an overdose for the second. The relevant extract of such a table could look like this:

Sample concentration	**u_1 (urgency of task 1)**	**u_2 (urgency of task 2)**
0.1 mol/L	0.8	0.1
0.12 mol/L	0.6	0.5
0.14 mol/L	0.4	0.7

Table 3.3: An urgency lookup table is the simplest way of mapping task states to an indicator of the threat they represent to the success of the associated task

Since the tasks are in conflict, the minima in their urgencies do not coincide. To find a compromise both urgencies are combined into a single HMI value that can be minimised. The previously stated mortality formula is used for this purpose.

$$HMI = \sum_{i=0}^{n-1} mortality(u_i) = \sum_{i=0}^{n-1} \frac{e^{\frac{1}{1-u_i}}}{1-u_i} - e \tag{3.15}$$

All in all, there are now three mappings that form a transitive relation between controls and their HMI value. First controls are mapped to a predicted task state. That state is mapped to a set of urgencies which, in turn, are used to compute the HMI. The following table shows the result of applying these mappings to find and select the most suitable compromise action from amongst the candidate controls.

Actuator distance	**u_1**	**u_2**	**HMI (u_1, u_2)**
0 mm	0.8	0.1	740
1 mm	0.6	0.5	40
2 mm	0.4	0.7	97

Table 3.4: Control samples with associated task urgencies and overall HMI rating

The plunger of the syringe will be pressed down by 1 millimetre in the next time step as this action is associated with the lowest HMI and hence represents the safest way of avoiding failure in either task. A balance is struck between the two competing tasks by selecting a medium dose.

4

Chapter Four

Goal Location Task

This chapter is the first of three task chapters. Each takes the reader through the definition of and solution to one of the demonstrator tasks used in the evaluation of the proposed coordination system. Aside from this practical use, the main purpose of these chapters is to show that a diverse range of meaningful tasks fit into the suggested task taxonomy and can fulfil the requirements of the task coordination framework. For improved readability all task chapters follow the same layout.

In this case, the first of the six sections, the task specification, introduces and closer defines the task of travelling between given initial and target configurations. Task-specific objectives are listed, followed by the classification of the goal location task as category one of the taxonomy. An overview of existing approaches to this well-studied problem is provided as part of the background, which also points out some of the difficulties relating to the complexity of cutting edge techniques. The section on task state prediction discusses the use of forward kinematics with reference to the appropriate literature for a differential drive robot, the vehicle type used throughout this project. Urgency heuristics required for coordination by the homeostatic control mechanism are derived in the eponymous section. A sequence of arcs is used to construct a potential function adhering to the requirements laid out in the previous chapter, while also guiding the robot along a smooth, low strain path to its destination. Experiments show successful generation of goal connecting paths for a variety of problems in free space. Despite favouring low execution time over precision, the $O(1)$ solution can be seen to produce visibly smooth paths. The final section briefly summarises the key points of this chapter and evaluates the presented solution to the goal location task in the context of task coordination.

4.1 Task Specification

4.1.1 Task Description

The goal location task is concerned with finding and executing a path leading from the robot's start configuration to a given destination. Variations on this task are distinguished by their definition of the robot's configuration and the path qualities they value.

Here, we consider travel in the 2D plane which means start and goal configurations are specified in terms of (x, y) points. This fully defines the state of an omnidirectional vehicle, but most vehicles also have an orientation which may be specified. In the latter case, the start configuration will be expressed as (x, y, ϕ), where ϕ denotes the heading. The goal configuration may or may not specify a target direction.

Path qualities further narrow down the type of solution desired. Most commonly, the shortest path is sought but other times the fastest or most economical route is preferred.

The following develops a solution that aims to produce smooth paths from an (x, y, ϕ) start configuration to a goal specified with or without a heading. So as not to introduce any unwanted dependencies between the tasks to be coordinated, the task assumes free space travel. Obstacle navigation is a separate task, which is the subject of the next chapter.

4.1.2 Objectives

First and foremost, the goal location task is designed to act as a demonstrator for task coordination. To qualify as such, the mechanism developed here must not prevent any requirements of the overall system being met. The need for providing a real-time solution capable of generating realisable paths is thus inherited.

Beyond constructing just any goal connecting path, the aim is to design a mechanism for generating smooth paths. The motivation for doing so lies in the fact that undulations, kinks and discontinuities make it hard for a vehicle to physically follow the chosen route. Especially when it comes to travelling at speed, smooth paths are highly beneficial as the robot need not slow down as much to manage sharp turns. It is also hoped that gradual changes in the direction of travel will aid in coordination due to affording other tasks more time to react to a commencing change. In terms of human-robot interaction, smooth paths are also safer as the robot's motion can more easily be anticipated and unwanted contact avoided.

Being able to define such a classical robotics problem in terms of the proposed task taxonomy lends credibility to the first hypothesis. The necessary abstraction for supporting task-agnostic coordination must be provided by adhering to the interface laid out as part of the second hypothesis.

4.1.3 Task Model

To elicit the desired behaviour from the robot, its controller must be guided by a suitable urgency heuristic. Encouraging motion towards the goal can be as simple as defining a cost function in terms of the remaining Euclidean distance to the goal configuration, but this is not sufficient here. Apart from the fact that such an approach yields the shortest path, which is typically not very smooth, a distance based heuristic cannot always be reduced. Consider the case where the robot is facing away from a goal located some distance behind it. If reversing is out of the question, as is assumed, the path must circle round to reach the destination. All forwards moves are however discouraged by the heuristic since the remaining straight line distance to the goal must first be increased before it can be decreased.

To avoid this problem and also generate smooth paths, we instead choose *strain energy* as the homeostatic variable. Strain is a measure of how much a path bends, meaning that its inverse can be used to define the property of smoothness. The following integral expresses strain energy ($\mathscr{E}$) in terms of curvature κ and path length s [53, p. 455]:

$$\mathscr{E} = \int \kappa^2 ds \tag{4.1}$$

The curvature of an arc is simply the reciprocal of its radius:

$$\kappa = \frac{1}{R} \tag{4.2}$$

For a general curve in the XY-plane described explicitly by the function $y = f(x)$, curvature is expressed as:

$$\kappa = \frac{y''}{[1+(y')^2]^{\frac{3}{2}}} \tag{4.3}$$

where primes indicate derivatives with respect to x.

Putting these together allows the strain of a path to be written as:

$$\mathscr{E} = \int \frac{(y'')^2}{[1+(y')^2]^{\frac{5}{2}}} \tag{4.4}$$

When the equation of the path is unknown, its strain can be approximated using a sequence of arcs. These arcs are small segments of the osculating circles that best approximate the curve within a small interval. The centre points of these circles are determined by the intersection of two normals to the curve at the start and end of the interval. Angular extent is measured as the angle between those normals and their length is the arc's radius. With the radius (R) known, the strain of each arc can simply be written as:

$$\mathscr{E} = \frac{1}{R^2} s = \kappa^2 s \tag{4.5}$$

The strain of the whole n-arc path is approximated by the sum:

$$\mathscr{E} = \sum_{i=0}^{n-1} \kappa_i^2 \, s_i \tag{4.6}$$

If the vehicle moves in arcs, this approximation will in fact produce the exact answer.

The challenge is now to find a goal connecting path with as little strain as possible. Although the robot cannot follow this path directly as this would not allow for compromise, its strain is used to indicate how well-suited a given action is to progressing the goal location task.

4.1.4 Task Classification

Attaining the target configuration has a purpose in and of itself. Since no external motivator is required, the goal location task can be seen to have intrinsic value.

With obstacle avoidance being treated as a separate task, external state is of no concern. Only the robot's own configuration, i.e. its internal state, is of interest during free space travel.

Together, this places the goal location task in the first category of the task taxonomy. Smoothness is a solution property cultivated by a task model that defines urgency in terms of strain.

4.2 Background

Autonomous navigation is a core functionality all mobile robots must provide and is correspondingly well understood. In the early days, mobile agents were predominantly modelled as points without kinematic restrictions and tended to search for shortest paths. Meanwhile smooth paths have garnered a lot of attention, as they can be more easily realised by real robots. Many of the techniques reviewed here were originally developed as mathematical models before being used in computer graphics and computer-aided design [53]. Application to robotics is more recent still. The following outlines different approaches to generating smooth paths.

One of the earlier contributions was made by Dubins [34] in 1957 and published in the American Journal of Mathematics. This very influential paper describes the construction of a minimum length path between two points with given headings. The path is composed of arcs and straight lines that are joined together. Due to its simplicity and low computational complexity, this solution is still used in robotics today (e.g.[82][50]). A drawback is that the path cannot be executed by most vehicles when travelling at speed. This is because of the curvature discontinuities at the join points between circular and linear sections of the curve. Tangents are continuous, but this is not sufficient for vehicles that need to transition gradually from one curvature to another. A bigger problem in the context of task coordination is, however, the fact that the shortest path is achieved by selecting the radius of the circular arcs to be that of the vehicle's MTC. This means the whole path may become invalid if, due to compromise, a wider curve is chosen leaving the remaining path to turn more than the vehicle is able. Finally, the method is designed to optimise path length rather than reduce strain.

Clothoids, also known as Euler or Cornu spirals, are characterised by a linear relationship between path length and curvature. By virtue of this, they are naturally continuous in curvature and have been explored as an avenue for providing realisable paths [79]. Although they do not guarantee least strain energy [53], clothoids can be used to achieve a good degree of smoothness. In [45] complex paths are composed of multiple clothoid sections, whereas [67] makes a more serious attempt as prioritising smoothness over path length reduction. Due to the close relationship between the Cornu spiral and the Fresnel integrals (the former is the result of a parametric plot of the latter), a lot of computation revolves around them. The difficulty of computing these integrals along with the need to set global waypoints detracts somewhat from their use in real-time systems [97].

The Bernstein polynomial functions used in Bézier Curves are much more easily evaluated [24]. The number of control points determine the degree of the curve and with that also its shape. Precision increases with increasing degree, but so does complexity. Placement of the control points to produce smooth curves is an issue addressed in [65].

Splines, which are piecewise polynomials, do a good job of interpolating points and can also provide smooth paths [94] with curvature continuity [114]. Variations on splines and other interpolation techniques such as NURBS [89] are concisely summarised in [97].

Finally, there are optimisation methods for producing not only low but least strain paths [133][81]. However, these involve deeper or more extensive search and are not suited to real-time computation, especially for robots travelling at high speeds. Both contributions cited here use Hermite's interpolation method and are geared towards computer aided geometric design.

4.3 Task State Prediction

For the controller to evaluate how much progress a candidate action makes towards the robot's destination, it must be possible to predict where that action will lead. This is known as the forward kinematics problem [35, p. 29], sometimes also referred to as odometry in the context of autonomous vehicles [112, p. 737]. The equations of motion are of course different for each type of vehicle, but can easily be found for most common designs. This project uses a differential drive robot for which the appropriate equations are derived in [35, p. 41]. Equations for synchronous drive, tricycles, and cars can be found in the same source. Hence, the framework's requirement for a state prediction mechanism is easily satisfied.

4.4 Urgency Heuristics

Unless otherwise indicated, the methods reviewed in the background section are considered to be fairly computationally efficient. Nonetheless, the search space is multi-dimensional with several control points, way points or other parameters having to be found. Essentially, what is meant by "fast" in the context of robotics, is the ability to update the path to the goal with sufficient frequency – typically once every cycle of the control loop. Real-time control is achieved because the generated paths are of high quality with properties that allow them to be followed by the robot with little further processing. In other words, we can afford to run these algorithms because they need only be run once in each time step.

Finding one good path and sticking to it can hardly be described as a recipe for compromise, however. As already established, task coordination requires several alternatives to be rated and compared in each control cycle. Evaluating all candidate actions with such accuracy as provided by cutting-edge methods is unlikely to be feasible. The same problem has been encountered in the areas of computer-aided design and industrial engineering, for example by Bertolazzi and Marco [13], who advocate a more drastic trade-off between smoothness and complexity. Specifically, biarcs (two arcs joined with continuous tangents) are said to be suitable for real-time applications. Biarcs were first used in computer-aided geometric design by Bolton in 1975 [15], having already been used in shipbuilding five years prior to that [60]. In robotics, the virtues of quick two arc approximations seem to be less appreciated as of yet with a recent contribution to collision avoidance [129] being one of few uses.

When connecting two (x, y, ϕ) "knots", i.e. start and end configurations, there is only one degree of freedom left [13]. This section discusses two ways of finding low strain paths among the family of interpolating biarcs created by varying the one free parameter. The first method performs a linear search for the merge point at which the biarc's two component arcs are joined.

In a second version, the turn distribution of the two arcs is set to mimic that of a clothoidal path, thereby eliminating the need for search altogether.

The section concludes with a procedure for converting the strain of the goal connecting biarc path to a suitable urgency value.

4.4.1 Biarc Strain Minimisation

As previously pointed out, it is insufficient to construct a geometric path for the robot to follow. Most vehicles are restricted in their freedom to move and at high speeds even the most agile robots succumb to the laws of physics. Having reinforced this point, we proceed to do just that: Construct a geometric path. However, the robot will not be required to follow it. Instead, the geometric construction is used to calculate the strain of the remaining journey from the robot's current configuration (which may be imaginary) to its destination. An AI controller then selects viable controls which minimise total strain to the best of the vehicle's abilities. Strain is thus used to guide the robot along a path that is as similar as possible to the low strain geometric path.

Here, a goal connecting path is constructed using only two arcs. While this will rarely yield the optimal least strain solution, the method is well-suited to reactive, real-time control, for which it was conceived. The path has continuous tangents, but not continuous curvature. Despite this, it provides a close enough approximation for present purposes.

4.4.1.1 Known Values

A navigation problem is defined by only two given configurations, each of which consists of a position on the (x, y) plane together with an angle that indicates the direction in which the robot is facing. The first is the robot's start (or current) configuration and the second is a known goal configuration. Start and goal can be connected by a straight line, the length of which is easily computed. Unless the robot is already facing in the direction of its destination and the target heading is also the same, the straight line is not a valid solution. A single arc may be used to transition between the start and goal orientations, yet cannot simultaneously achieve the goal position in all but very simple cases. The method described here merges two arcs to do what one alone cannot.

4.4.1.2 Single Arc Connection

The first step in constructing the biarc is to connect start and goal with a single arc that goes through the required total turn. Of course, this arc is not a solution in itself, but it provides a geometric framework for merging the two arcs that form the final path. As can be seen in the diagram below, the headings at the start and goal configurations are at angles α and β to the straight line connecting them. This line will also be the chord of the single arc, which is why it is labelled $chord_0$. In total, the path must go through a turn of $\alpha + \beta$ to reach the desired heading. The single, symmetric arc will generate the required total turn, but inevitably has the same angle at either end. This angle is the average angle $\theta = \frac{\alpha+\beta}{2}$ which results in the tangents shown as dotted blue lines that clearly do not align with the desired headings.

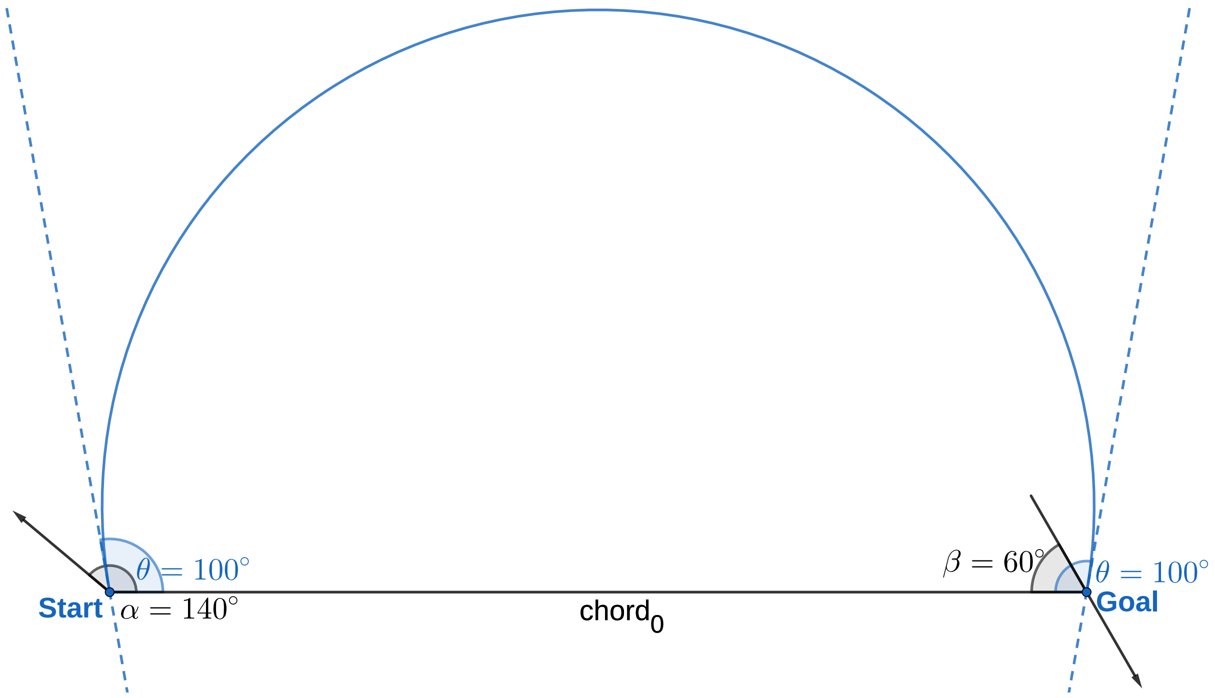


Figure 4.1: A single arc connecting start and goal configurations provides a frame of reference for biarc construction. Headings are shown as black arrows with angles of α and β to the chord. The orientation of the arc's tangents at the start and goal configurations is given by θ, the average of α and β.

4.4.1.3 Merge Point

To satisfy both end angle constraints, the arc is now split into two. Key in splitting the arc is making sure that the first of the two new arcs leaves a problem that can be solved by the second arc. That means the second half of the problem must have equal tangent angles at either end and the tangent at the goal must be in line with the β angle. If this were not the case, we would again be faced with an asymmetric problem that cannot be solved using a single arc, thus failing to reduce the initial problem. The trick is to pick the merge point MP, at which the two individual arcs are joined, on the single arc. Why this works will become apparent as the method is developed.

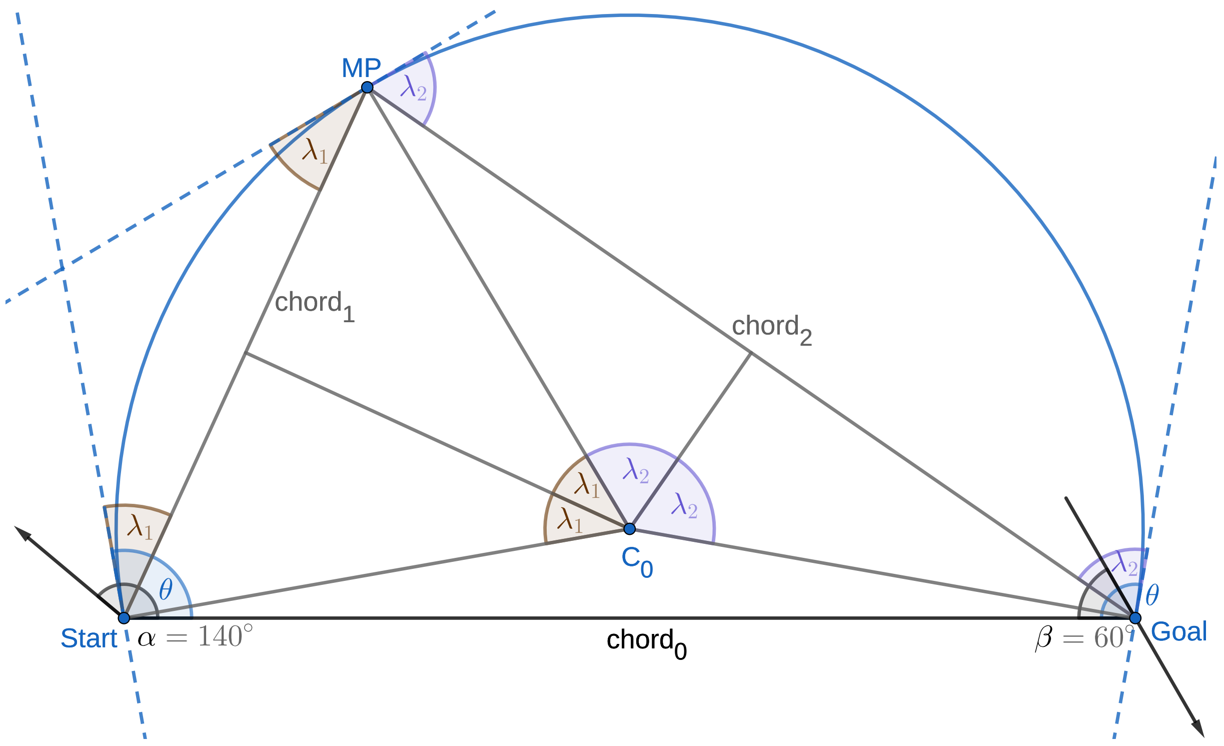


Figure 4.2: Construction of the merge point with one remaining degree of freedom to allow different biarc solutions to be specified by moving the merge point along the single arc

The location of the *MP* can be varied to obtain different two arc solutions. Its position along the single arc is determined by the angle $\angle Start\ \ C_0\ \ MP$ where C_0 is the single arc's centre of curvature. Let $\angle Start\ \ C_0\ \ MP = 2\,\lambda_1$. This is convenient because then λ_1 is the angle between the chord of the first arc (from *Start* to *MP*) and the blue tangent lines at those points. Proceed the same way for the second arc from *MP* to *Goal*, noting that λ_2 is determined by λ_1 because the single arc goes through a total turn of $2\,\theta$.

$$2\,\lambda_1 + 2\,\lambda_2 = 2\,\theta = \alpha + \beta \;\;\Leftrightarrow\;\; \lambda_2 = \frac{\alpha+\beta}{2} - \lambda_1 \tag{4.7}$$

4.4.1.4 Biarc Fitting

Once the merge point has been set, the arcs on either side can be fitted to match the headings at the start and goal configurations. To do so, we add an angle δ to the chord angle λ_1 of the first arc and subtract the same amount from λ_2. This way the total turn remains unchanged, while its distribution does change. If $\delta = 0$, the arcs lie exactly on top of the single arc, but to fit them such that their tangents align with the headings at *Start* and *Goal*, we let $\delta = \alpha - \theta$.

$$\delta = \alpha - \theta = \alpha - \frac{\alpha+\beta}{2} = \frac{\alpha-\beta}{2} \tag{4.8}$$

The first arc then has an angle $\phi_1 = \lambda_1 + \delta$ between $chord_1$ and heading α. The second arc aligns with β using $\phi_2 = \lambda_2 - \delta$. As can be seen from the final construction shown below, the symmetry of the rotation by δ at the merge point ensures that the two arcs have continuous tangents. Furthermore, the total turn is preserved and the remaining turn after the first arc is equally distributed amongst the end angles of the second arc, allowing it to complete the path.

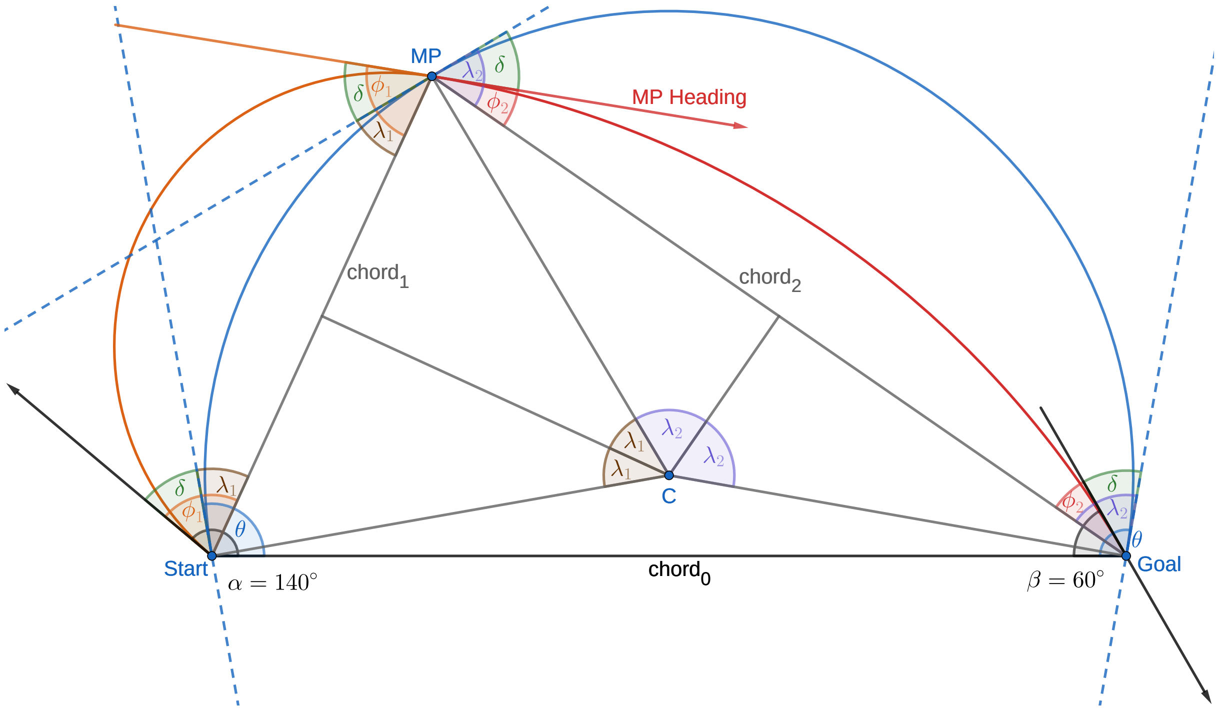


Figure 4.3: Fitting a biarc connecting start and goal configurations via the merge point at which the two component arcs meet with continuous tangents

4.4.1.5 Strain Computation

Using $\kappa = \frac{1}{r}$ and $s = r\,\phi$ (where r is the radius and ϕ the turn of an arc), we can rewrite strain as:

$$\mathscr{E} = \frac{\phi}{r} \tag{4.9}$$

An arc goes through twice the turn of its end angles, meaning that the first and second arcs have respective strains of

$$\mathscr{E}_1 = \frac{2\,\phi_1}{r_1} \qquad \mathscr{E}_2 = \frac{2\,\phi_2}{r_2} \tag{4.10}$$

where only the radii remain to be found and these can be expressed in terms of the chord lengths.

$$r_1 = \frac{chord_1}{2\,\sin\phi_1} \qquad r_2 = \frac{chord_2}{2\,\sin\phi_2} \tag{4.11}$$

The chord lengths are:

$$chord_1 = 2\, r_0\, \sin\lambda_1 \qquad chord_2 = 2\, r_0\, \sin\lambda_2 \tag{4.12}$$

and finally we find the radius of the single arc as:

$$r_0 = \frac{chord_0}{2\, \sin\theta} \tag{4.13}$$

4.4.1.6 Strain Minimisation

The biarc of least strain can now be found via linear search to an arbitrary precision. To do so, the merge point is iteratively moved along the single arc solution by selecting different values for λ_1. If n biarcs are sampled, then the i-th merge point is chosen using:

$$\lambda_1 = \frac{i\,\theta}{n-1} \tag{4.14}$$

The strain of each sample is computed as the sum of the strains of both component arcs. A suitable, smooth biarc path is selected by choosing the sample associated with least total strain energy.

4.4.2 Biarc Clothoid Approximation

While a linear search time is already very good, a direct constant time construction can be achieved by using insights from clothoids. As already mentioned as part of the background, clothoids have a linear relationship between path length and curvature.

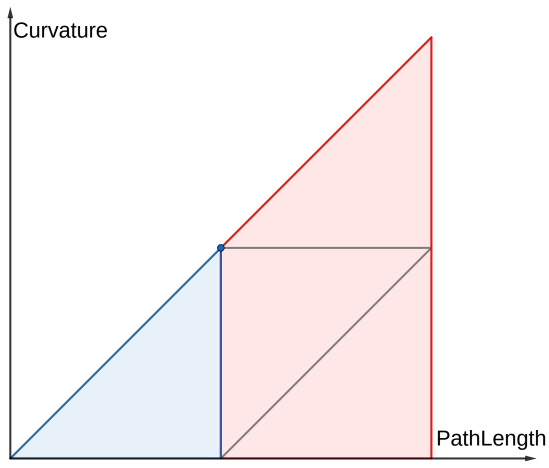


Figure 4.4: Ratio of turn between the first and second halves of a clothoid split equally in terms of path length

In curvature – path length space, the clothoid is a straight line (here of standard gradient 1) with the area under the curve representing turn. The above illustration already shows how the clothoid can be split up into two parts of equal path length (see blue and red areas). By dividing these areas up into equal sized triangles, it is easy to see that the first half of the path goes through a quarter of the overall turn, leaving 75% to be done by the second part. To mimic this turn distribution with a biarc, simply select α and β to be in the same 1:3 ratio. With α fixed by the robot's initial heading and the direction of the straight line to the goal, we can only choose $\beta = \frac{\alpha}{3}$. This skips the whole iterative merge point selection step, allowing direct evaluation of:

$$\delta = \frac{\alpha - \beta}{2} \tag{4.15}$$

The curvatures, path lengths and strain energies of the two arcs are then computed in the same way as above. By approximating a clothoid, we inherit some of its desirable smoothness properties without search. This $O(1)$ solution does come at the price of sacrificing the ability to reach the target heading, seeing as β is determined by the desired ratio. So instead of (x, y, ϕ) goals, only (x, y) points can be chosen for the destination. For a constant time algorithm the generated path is still pretty smooth and represents a good trade-off between complexity and accuracy. Since speed is the most valuable commodity, this is the method of choice for which test are conducted in the following section.

4.4.3 Mapping Controls to Strain

The ability to construct a biarc between any two configurations allows candidate controls to be rated in terms of their prospects of completing the remaining journey smoothly.

Let the robot be at a current configuration (C) with goal configuration (G). To rate an action (a), first use the state prediction mechanism to find the end configuration (E) of the transition its execution results in. For the goal location task, state prediction is assumed to be provided in the form of a forward kinematics function *fk*:

$$E = fk(C, a) \tag{4.16}$$

The path the robot takes from C to E is known (defined by the action) and can be costed for strain. For a differential drive robot that generates arcs, this is trivially done using the already established formulae. Other vehicles may produce different primitive forms of motion, the strain of which will be costed differently. Whichever way obtained, this strain is referred to as the *past* *strain* since the robot is now imagined to be at end configuration E. Generally:

$$pastStrain = strainOfPrimitiveMotion(C, a, E) \tag{4.17}$$

Given that the motion will have the same strain, no matter where it originates, C is in fact an arbitrary input to this function. By extension, so is E which is determined by C and a, leaving the action as the only real parameter:

$$pastStrain = strainOfPrimitiveMotion(a) \tag{4.18}$$

The remaining journey from E to G is approximated using the biarc construction mechanism described. Summing the strain contributions of the two fitted arcs yields the *future strain*.

$$futureStrain = biarcStrain(E, G) = biarcStrain(fk(C, a), G) \tag{4.19}$$

The total strain of the path from C to G via E can then be written as the sum:

$$totalStrain(C, a, G) = strainOfPrimitiveMotion(a) + biarcStrain(fk(C, a), G) \tag{4.20}$$

Using this strain heuristic, the controller can now proceed along the lowest strain path by selecting the action associated with least total strain.

4.4.4 From Strain to Urgency

While the strain heuristic given above is sufficient to guide a controller solving only this one task, an urgency heuristic is required for multi-tasking. Ideal and worst cases are defined here to allow urgency to be expressed in terms of strain. Specifically, the missing parameters to the following equation are sought.

$$goalUrgency = \frac{\mathscr{E}_{sample} - \mathscr{E}_{ideal}}{\mathscr{E}_{worst} - \mathscr{E}_{ideal}} \tag{4.21}$$

Assuming we had a perfect heuristic for the strain of a path between two configurations, the ideal would simply be the value obtained by providing current and goal configurations as the parameters to that function. In lieu of such perfection, the biarc approximation is used.

$$\mathscr{E}_{ideal} = biarcStrain(C, G) \tag{4.22}$$

There is a caveat however: The approximation is not admissible, i.e. the biarc may overestimate the strain of the remaining path. This should not come as a surprise; after all, precision was willingly traded for computational efficiency. Indeed, the trade-off can be made without repercussions as long as no sample can trump the ideal strain. Strictly speaking, admissibility is not even a requirement, only that $\mathscr{E}_{sample} \geq \mathscr{E}_{ideal}$. If this were not the case, the numerator in the above formula for the goal location task's urgency would be negative. Since $\mathscr{E}_{worst} > \mathscr{E}_{ideal}$, urgency would also be negative and thus outwith the valid range of $[0,1]$.

The danger seems remote until one realises that each sample is evaluated using a total of three arcs compared to the two arcs used in the biarc approximation of the ideal. One arc is specified by the sample action and generates the arc from C to E and then the path from E to G is completed with a further two arcs. So in fact, the sample almost inevitably undercuts the ideal due to its superior precision.

An easy remedy is to empirically observe the maximum amount by which the supposed ideal can be outbid and then reduce it accordingly. It was found that strain can be reduced to just over 90% (or by a factor of 0.9) of the initial biarc estimate by using an additional short sample arc. So, the following ideal can no longer be beaten by a sample – although it may still be an overestimate which can be undercut by using more arcs to construct a higher precision path.

$$\mathscr{E}_{ideal} = 0.9\; biarcStrain(C,G) \tag{4.23}$$

The worst case for the goal location task would be never reaching its destination, i.e. continuing along an infinitely long path with infinite strain. In practice, this theoretical case is too abstract to be of interest. A more practical interpretation of the worst scenario that may actually come to pass in the foreseeable future is having to turn around and go in the opposite direction. To quantify this in terms of strain, add a semi-circle arc (i.e. a U-Turn) at the robot's MTC radius to the journey and then evaluate its strain.

$$\mathscr{E}_{worst} = \mathscr{E}_{ideal} + uTurnStrain \tag{4.24}$$

With a bit of cancellation in the denominator, this gives the following expression for the urgency of a control sample:

$$goalUrgency(C,a,G) = \frac{totalStrain(C,a,G) - 0.9\; biarcStrain(C,G)}{uTurnStrain} \tag{4.25}$$

4.4.5 Motivation to Accelerate

The solution developed so far is theoretically sound, but one would be wrong to assume it contains a motivation for the robot to reach its destination. It is easy to presume that setting a goal implies that it should be attained sooner rather than later, but as always, a machine needs to be instructed explicitly. Not only does the robot lack a motivation to accelerate, it is in fact discouraged from doing so. This undesired bias towards slow motion is explained by the higher degree of precision afforded by using many small arcs instead of fewer larger ones. The slower the vehicle moves, the smaller the individual arcs and the more accurately the least strain path is realised. Since the robot *is* motivated to reduce strain, it will edge forwards in a lacklustre fashion.

To encourage a brisker pace, the controller must be rewarded for selecting faster control samples. A slight modification of the formula for the total strain of an action provides the needed incentive:

$$totalStrain(C,a,G) = DF\ \ strainOfPrimitiveMotion(a) + biarcStrain(fk(C,a),G) \tag{4.26}$$

where DF is a *discount factor* in $[0,1]$ which reduces an action's *past strain* contribution to the total. The discount only applies to the transition made by the sample action and not to the future component. So to benefit most from the discount, a larger proportion of the path must be generated by the action and this is achieved by selecting faster speeds. The discount should be as small as possible, to prevent a distortion of the actual strain. If there is no sufficient penalty for a high strain action, the smoothness of the path will degrade. A value of $DF = 0.97$ was found to produce the desired effect without compromising smoothness unnecessarily.

One exception remains, and that is the straight line case in which all strains are zero and the discount has no effect. When a straight line is a valid solution to connecting start and goal configurations, it is also indisputably the best. A line has $\kappa = 0$ and therefore $\mathscr{E}_{ideal} = 0$. Any sample action producing part of that straight line will have a past strain of zero for the same reason. The future journey will be the remainder of the ideal line and likewise have no associated strain. In summary, strain is zero all along that ideal line – including at the robot's start configuration at which it may as well remain, since no gain is to be had by moving.

The problem is evidently the zero curvature factor in the formula for strain. Path length, which is otherwise considered, disappears along with curvature. To make sure this cannot happen, all curvatures are artificially incremented by one. This does indeed solve the issue, but although it appears as though all samples are treated equally this is not the case. Measuring strain in this way can change the ranking of sample actions.

Let there be two arcs with the following lengths, curvatures and strains:

$$\kappa_1 = 1.0 \qquad length_1 = 1.0 \qquad \mathscr{E}_1 = (1)^2 \ 1.0 = 1.0 \tag{4.27}$$

$$\kappa_2 = 2.0 \qquad length_2 = 0.3 \qquad \mathscr{E}_2 = (2)^2 \ 0.3 = 1.2 \tag{4.28}$$

The first arc has a strain of 1 and the second a strain of 1.2, but this ordering changes when curvature is artificially incremented by one:

$$\mathscr{E}_1 = (1+1)^2 \ 1.0 = 4.0 \tag{4.29}$$

$$\mathscr{E}_2 = (2+1)^2 \ 0.3 = 2.7 \tag{4.30}$$

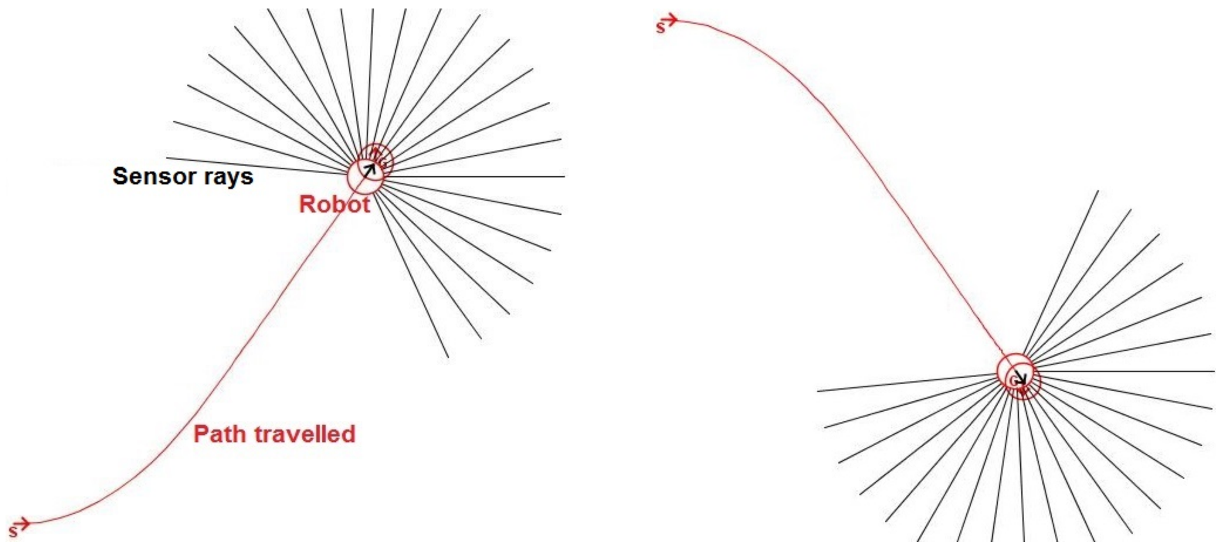
Now the second arc appears to have significantly less strain than the first although the first is smoother. The above is an example of a fairly severe case. In practice, the difference in length between candidate control samples is never as big due to the short time steps. As a result, the adverse affect on ranking is not detrimental. The described fix can be (and is) used to solve the issue of zero strain straight lines, although it somewhat sullies the otherwise clean theory. A more elegant solution that simultaneously removes the need for the discount factor is to add a time-sensitive task as proposed in the future work section.

4.5 Experiments and Results

Experimental results shown in this section attest to the competence of the developed solution for real-time smooth path finding. All screenshots displayed in the following were taken from the purpose-built robot simulator described in Chapter 7. It is important to first make sure that all components of the system are working before turning to the more complex problem of coordination. That is why results presented here pertain only to the goal location task itself, i.e. tests are carried out in isolation without interference from other tasks. For an evaluation of the coordination system as a whole, likewise refer to Chapter 7.

Test cases are set up with goal configurations at different angles in relation to the robot's initial heading. Since only (x,y) goals are selected, the headings at the goal configurations will not influence the path and need not be varied. Specifically, targets are set at intervals of 45° in terms of the initial chord angle, i.e. the angle of the line connecting the start and goal points. This covers the most significant edge cases with which the system may conceivably struggle.

Consider a robot starting at $S : (x = 0, y = 0, \phi = 0)$, i.e. at the origin of an XY-coordinate system facing along the positive X-axis. The first experiment aims for a goal positioned at $G : (x = 1, y = 1)$, which means that the angle of the chord connecting S and G is 45°. A single arc could solve this problem using a quarter circle turn, going through 45° at either end for a total of 90°. Without a goal heading being prescribed, a lower strain path can be found that goes through less turn. The described situation is pictured in figure 4.5a together with the path generated. Figure 4.5b shows the same problem mirrored on the X-axis with a chord angle of −45°. While the path is the exact mirror image as it should be, it is important to test for these cases as wrong assignment of the direction of turn was the most common source of error encountered. Angles going the wrong way around have the opposite sign and can cancel out, leaving the false impression that a straight line is required.

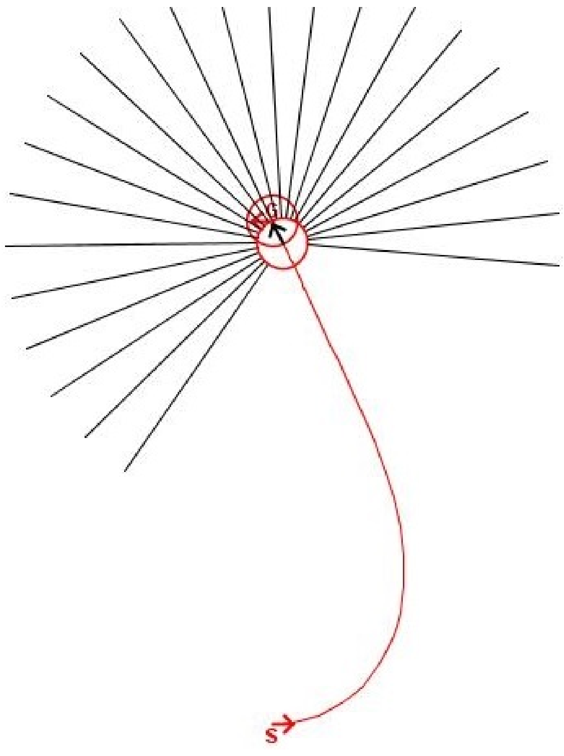


(a) Anticlockwise turn **(b)** Clockwise turn

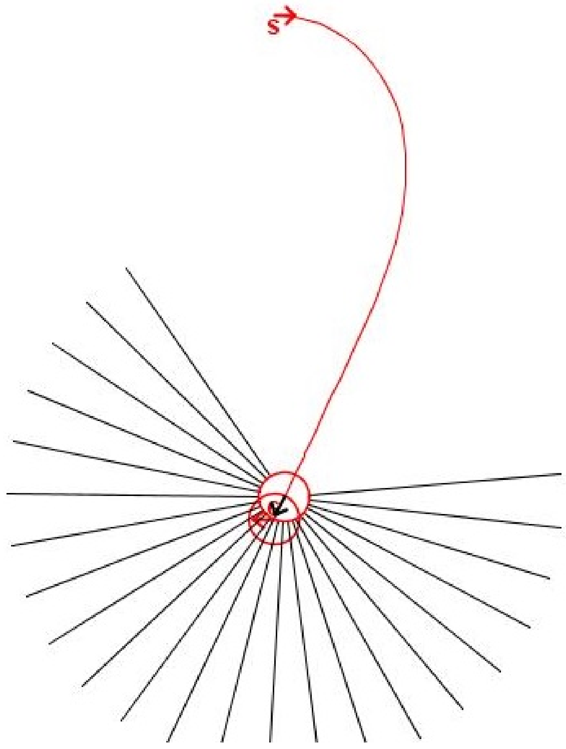
Figure 4.5: Goal connecting paths generated for chord angles of ±45° by minimising biarc strain

Start and goal configurations are marked as S and G respectively with red arrows indicating the heading of the robot (the heading at G can be ignored). A small circle around the goal marks an area of tolerance within which the robot is accepted to have reached its destination. The robot itself is drawn as a larger red circle with a black arrow indicating its current orientation. Black lines radiating out from the vehicle indicate distance sensors. These are not yet needed as the task assumes free space, but will play a key role in obstacle navigation (see Chapter 5).

The same experiment is repeated for a more severe chord angle, this time of 90°. A semi-circle could solve this problem, but again a path with less turn is preferred.



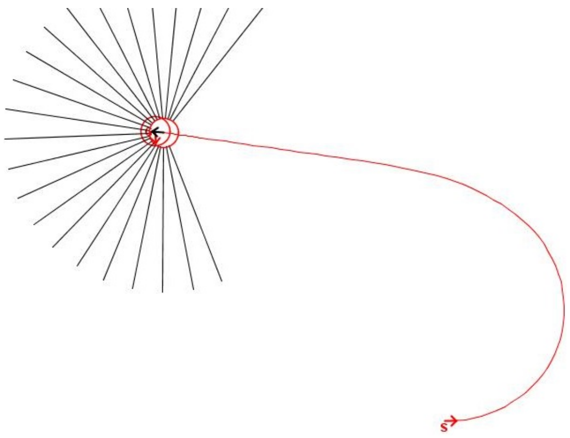
(a) Anticlockwise turn



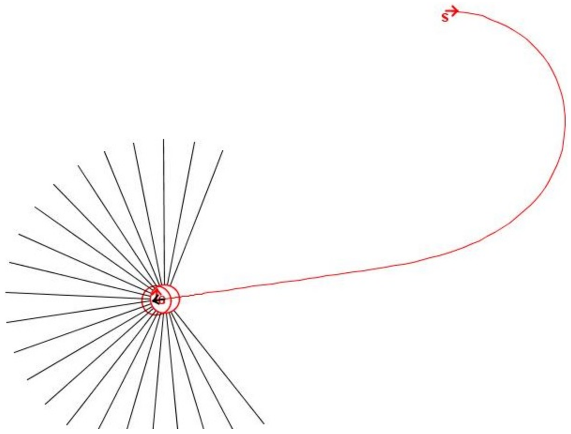
(b) Clockwise turn

Figure 4.6: Goal connecting paths generated for chord angles of ±90° by minimising biarc strain

With a chord angle of over 90° in magnitude, the goal is located behind the robot. This case is of particular interest because the desired path initially leads away from the goal. A distance based heuristic would struggle in this case, but minimising strain results in a smooth path without a drastic turn to greedily reduce goal distance.



(a) Anticlockwise turn



(b) Clockwise turn

Figure 4.7: Goal connecting paths generated for chord angles of ±135° by minimising biarc strain

A straight line is ironically one of the most difficult paths to produce when working with strain energy. There simply is no incentive to move towards a goal when the heuristic function is already at its minimum value of zero right where the robot currently is. Because the remaining journey is not associated with any strain, the situation is (at least numerically) indistinguishable from one where the robot has already reached its destination. The following test confirms that the workaround for this problem does indeed have the desired effect.

Also shown is the pathological case where the goal is exactly behind the robot with a chord angle of 180°. Due to collinearity of S, G and the robot's orientation, the problem looks similar to the straight line case. Indeed, if the robot were allowed to reverse it should do so but here we assume forward only motion. Due to perfect symmetry clockwise and anticlockwise routes have the same strain. This may make it difficult to decide which direction to turn in. If other tasks were present, their preferences could tip the balance towards one direction or the other. In the absence of external influences, the decision is made simply based on floating point precision which will favour one side by a minuscule amount of strain. Oscillation between left and right turns need not be feared however, since symmetry is broken after the very first step is made.

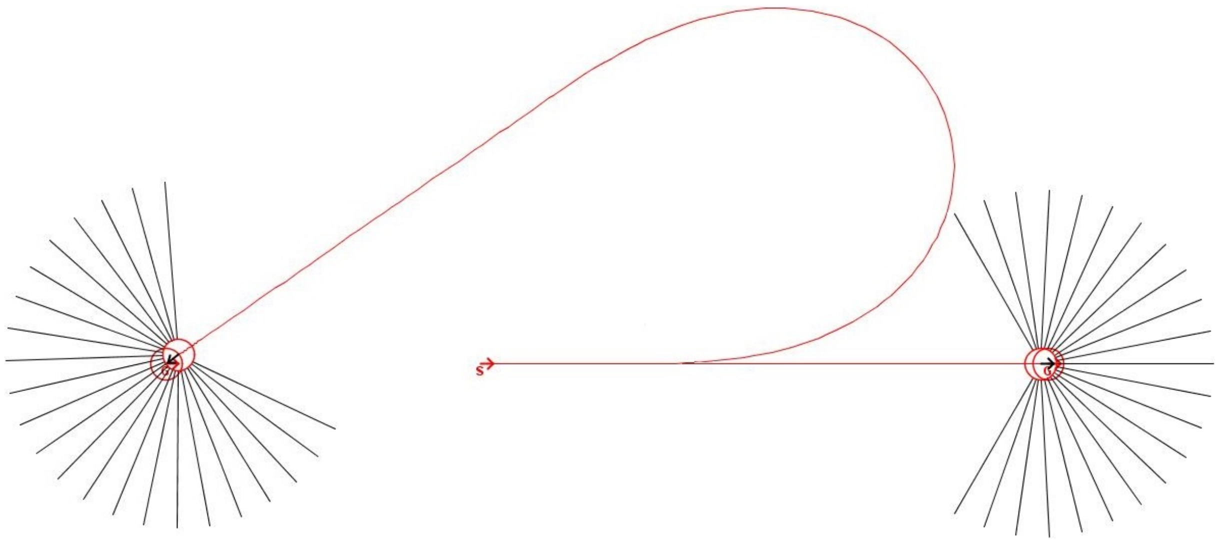


Figure 4.8: Goal connecting paths generated for chord angles of 0° (straight line) and 180° by minimising biarc strain

So far the initial orientation of the robot has not been varied. To make sure that all angles are rotated correctly with respect to the robot's orientation, the above experiments are repeated for different initial headings. The following screenshots show the successful generation of paths for all rotated problems.

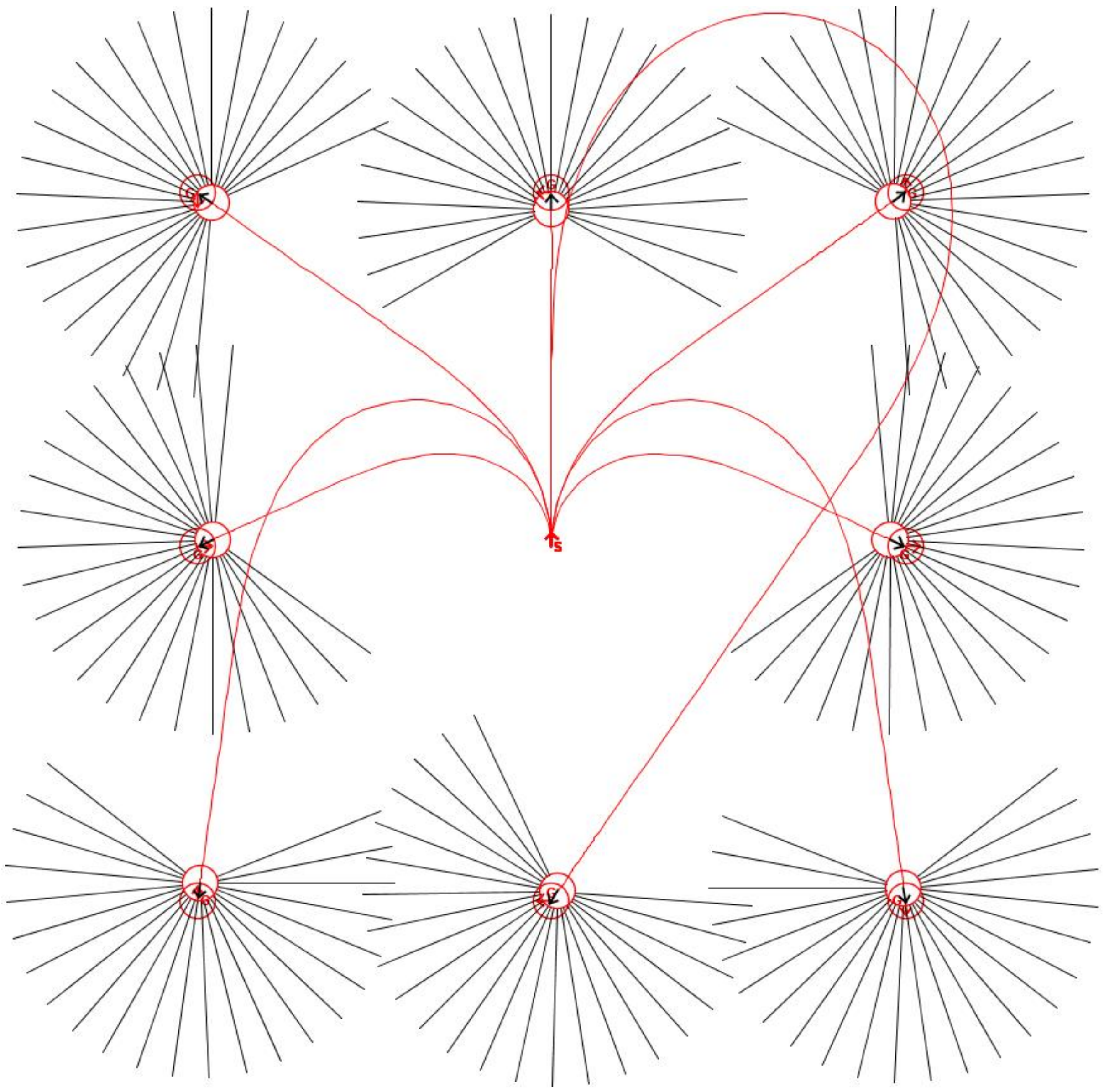


Figure 4.9: Collection of test cases with an initial robot heading of 90° and a variety of chord angles. Solutions consistent with those from previous experiments demonstrate correct rotation in line with the robot's orientation.

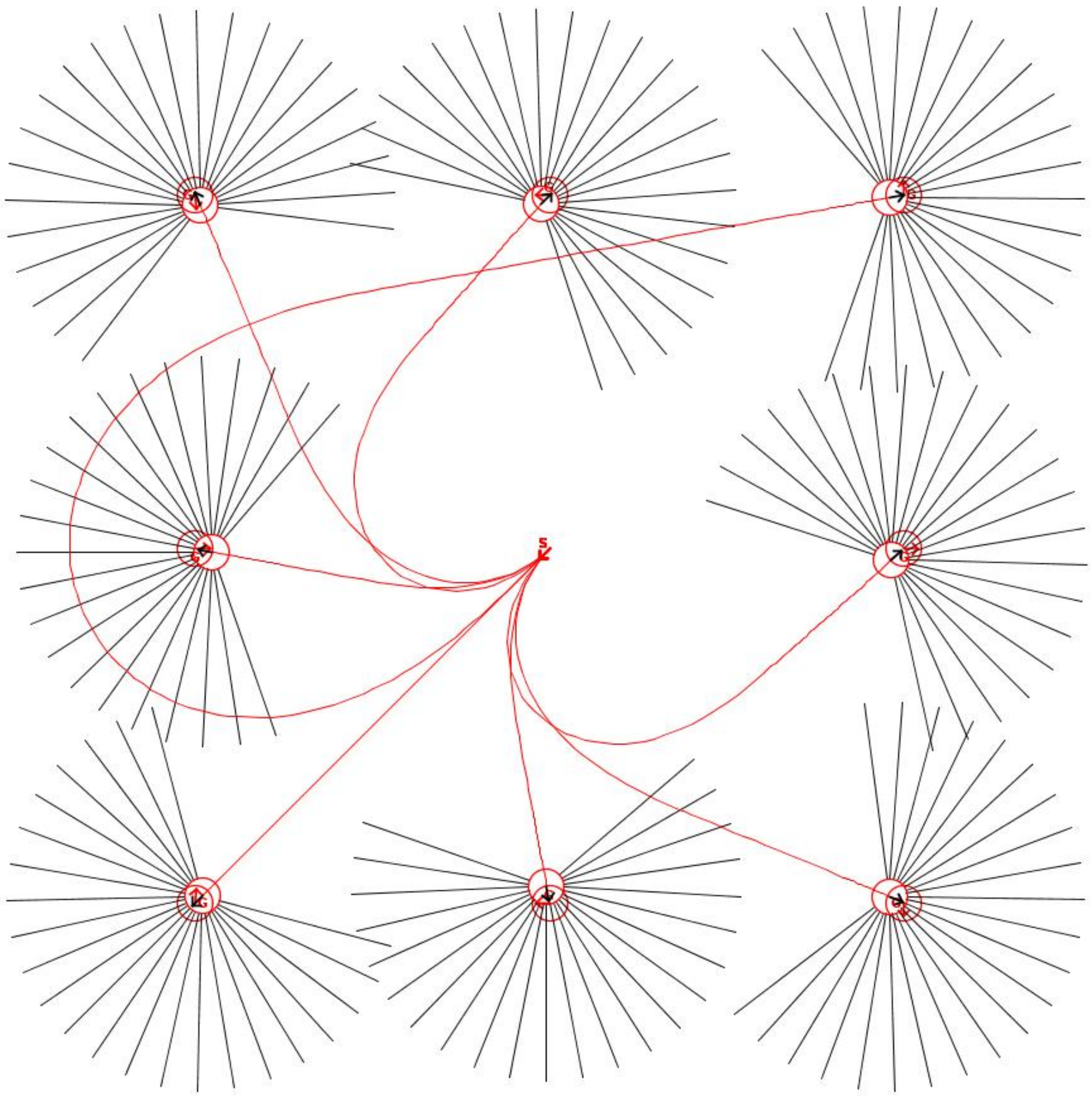


Figure 4.10: Collection of test cases with an initial robot heading of $-135°$ and a variety of chord angles. Solutions consistent with those from previous experiments demonstrate correct rotation in line with the robot's orientation.

As can be seen, the path shapes are invariant under rotation. Experiments show success across the board with the ability to connect any (x, y, ϕ) configuration to any (x, y) goal.

4.6 Summary and Evaluation

Solutions for finding least strain energy curves already exist, but are better suited to computer-aided design and graphics where time constraints are laxer. A number of smooth path generation techniques that produce low, but not least strain were reviewed. These trade precision for speed to different degrees and have been applied to robotics problems with great success. For rating and comparing the large number of alternative actions that must be considered in task coordination, the search space is still dauntingly large however.

This chapter proposed an even more drastic trade-off between smoothness and speed. A low strain path is roughly approximated using only two circular arcs. This is insufficient to provide support for reaching a goal heading, but enough to connect an (x, y, ϕ) start configuration to any (x, y) goal. Most importantly, the arcs can be fitted and costed for strain in $O(1)$ time, which makes up for their lack in precision – at least for present purposes. Such low computational complexity could only be achieved by foregoing curvature continuity, which is justifiable as the path will not be followed directly. Instead the arcs are discarded as soon as they have played their part in computing a strain based urgency heuristic to guide the robot in selecting actionable controls. Making a fair comparison to existing techniques is difficult due to this disparity in their underlying premise.

The presented solution conforms to the task coordination framework. In line with the second hypothesis, it provides a state prediction mechanism and an urgency heuristic. The goal location task fits category one of the proposed taxonomy of tasks and is the first example given in support of the first hypothesis. With tests showing successful navigation in free space, the task can be declared a suitable demonstrator for the coordination system.

CHAPTER FIVE

OBSTACLE NAVIGATION TASK

Having obtained a working solution to the goal locational task we now consider another common problem, namely that of navigating obstacles. Two separate heuristics are designed to deal with avoiding collisions and circumnavigating obstructions in the robot's path. These heuristics and other key aspects relating to this task are developed over the course of six sections laid out as in the previous chapter.

The first section reflects on the intuitive expectations one might have of obstacle navigation before progressing to a more precise specification. In creating a model to satisfy the set objectives, the need for a second task is discovered. The conceptual task of obstacle navigation is accordingly split into two category four tasks that together produce the desired behaviour. A review of related literature examines a variety of existing approaches and ways in which these may be adapted to meet the requirements of the framework and coordination mechanism. State prediction concerns itself with obstacle detection and the measurement of distance between the robot and obstacles in its vicinity. A geometric model of the environment allows distances to be estimated after simulating the application of candidate controls. Urgency heuristics are developed for both aforementioned taxonomy tasks. The obstacle avoidance heuristic builds on classical methods mentioned in the literature review. Circumnavigation uses a novel technique combining results from the goal location task with existing concepts. Testing assesses the ability to steadily circle a diverse range of obstacles at a safe distance. As with other tasks, the tests are performed in isolation to rule out interference from other sources. A summary of key points concludes this chapter together with an evaluation of the presented solution. This final section also assesses compatibility with the framework and the suitability of obstacle navigation to act as a demonstrator for task coordination.

5.1 Task Specification

5.1.1 Task Description

Obstacle navigation is such a prevalent task it should require little explanation, yet ironically it has been so well studied that distinction between its many variations is meanwhile called for. While intuitively the task is for the robot to deal with all encountered obstructions using any means necessary, a given solution will be more specialised. Some will only deal with static obstacles whereas others can avoid moving obstacles by making predictions about their future positions. Reliance on a priori knowledge of the environment (as opposed to sensor information) is a further distinguishing factor. Sometimes even the type of obstacle matters, with a number of planners not being able to escape concave shapes. Finally, the robot may react to obstacles in different ways. Some obstacle avoidance techniques, merely seek to prevent collision by turning the vehicle and retreating back into free space. A more proactive approach is to circumnavigate obstacles by following their outline until progress has been made. This would for instance be appropriate when traversing a labyrinth as it corresponds to the hand-on-the-wall rule.

The latter approach is pursued in this chapter, which defines obstacle navigation as the task of circumnavigating obstacles by following their outline at a safe distance. In this way sustainable progress can be made even in the presence of concave obstacles. Following a contour around obstacles rather than clinging to their edges leaves room for error and more importantly compromise. Furthermore, the safety margin can be varied to allow for robots of varying, finite dimensions. The task must rely solely on sensor information but does not consider moving obstacles.

5.1.2 Objectives

The aim of this chapter is not to improve on the many existing solutions to obstacle navigation. Instead, it seeks to adapt already existing approaches to fit within the task coordination framework and serve as a demonstrator for SMT. Adaptation is required in two main areas.

Firstly, there appears to be no existing technique for obstacle navigation in complete isolation of other tasks. This is unsurprising as such a task would be useless outside of a task coordination framework in which the impetus for the robot to move comes from other tasks. Consequently, available solutions invariably include the drive to reach a goal location as well. Since the framework prescribes strict separation of the responsibilities of all tasks being coordinated, the goal oriented aspect needs to be divorced from the rest of those mechanisms for them to qualify. The difficulty of segregating these aspects depends on how closely they are linked in theory.

Secondly, the adapted demonstrator task must provide some leeway for task coordination to be possible. A too rigid policy for following a specific route around obstacles would be detrimental to the process of finding the required compromises.

Another objective is to develop a mechanism to allow the single-step planner to succeed in situations that would otherwise require the foresight it cannot provide. Using a short-sighted obstacle navigation mechanism, the robot can quite literally paint itself into a corner. The problem is exacerbated by the requirement to generate realisable paths adhering to a realistic minimum turning circle. While the robot may currently be satisfied that it is at a safe distance to the nearest obstacle, it may already be destined to crash due to its failure to anticipate that even the most severe turn it is capable of will lead to collision a few steps down the line. To pre-empt this issue, an incentive is to be created for the robot to turn towards and align itself with the target obstacle contour in a timely fashion. Early turning prevents situations such as the robot reaching the contour facing the obstacle with too little space to perform the necessary right-angle turn to either side. Even once the contour has been successfully joined, a similar problem persists. Irregular obstacle shapes and corresponding changes in the direction of the contour which follows its outline may demand abrupt turns that must likewise be commenced before the vehicle's MTC is breached. In summary, the desired mechanism should facilitate smooth contour joining and following.

Lastly, a successful adaptation of the task to comply with the requirements of the framework serves as further evidence to support the first and second hypotheses.

5.1.3 Task Model

Inciting the robot to perform the task as described requires guidance by heuristics defined in terms of the appropriate homeostatic variables. This section discusses which of the obtainable state variables best reflect an action's ability to further the task's agenda.

Collision prevention is the most fundamental feature of obstacle navigation and also the easiest to provide. Urgency may simply be based on obstacle distance, or more specifically the gap between the robot's perimeter and the closest obstacle edge to it. This distance is suitable as a homeostatic variable but must first be derived. It is neither part of the robot's configuration, nor a property of the environment but calculated from a combination of both. State prediction is responsible for inferring the required information by interpreting readings from the available sensors. The only constant that needs to be supplied is the target contour distance, which is chosen with the vehicle's dimensions in mind. Its value simultaneously defines the desired offset from the obstacle and the bounds for the homeostatic variable.

Regulating obstacle distance is necessary but not sufficient to fulfil all stated objectives. Figure 5.1 illustrates what would happen, as opposed to what should happen, when the robot is guided by obstacle proximity alone.

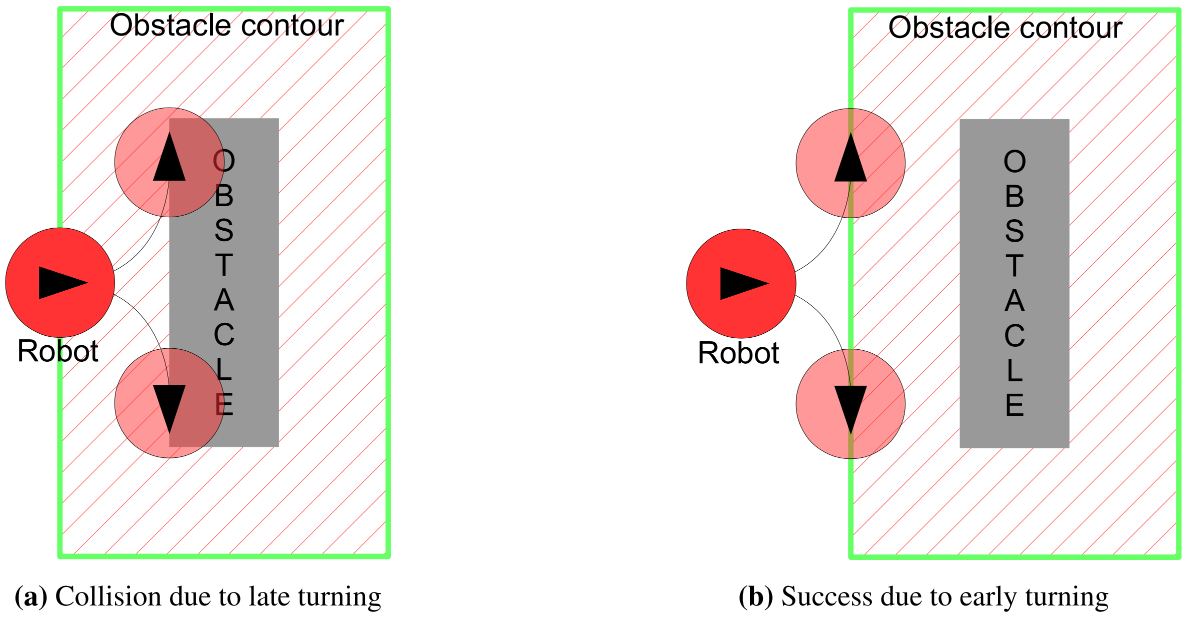

Obstacle contour
OBSTACLE
Robot
(a) Collision due to late turning
Obstacle contour
OBSTACLE
Robot
(b) Success due to early turning

Figure 5.1: Late turning (informed by obstacle proximity) versus early turning (using path smoothing)

The figure shows the contour as a green line following the edge of a grey obstacle at a safe distance. Ideally, the centre point of the circular robot should follow this line. In the diagram on the left the vehicle's current position (solid red) is on the contour exactly as it should be, but its heading (black arrow) is not aligned with it. The predicament the robot is in can already be seen although its single-step planner is oblivious to the fact that collision is imminent as indicated by the possible future positions shown in transparent red. To ensure there is enough manoeuvring space to align with the contour, the turn must be commenced before entering the red-striped danger zone. Since foresight is limited and the distance-based penalty for encroaching on the obstacle only comes into effect once the contour has been crossed, the robot cannot be expected to turn before reaching it. Herein lies the necessity for a smooth contour joining and following mechanism, which will also allow the fourth hypothesis to be upheld. Producing the desired behaviour shown in the diagram on the right will evidently require an additional homeostatic variable.

Seeing as proximity alone is not able to induce early turning, the question is what can. An obvious first choice is to encourage the robot to align with the contour by reducing the angular difference between their orientations. As with contour distance, this homeostatic variable is also derived from parts of the robot's configuration and information about the obstacle – in this case the course of the contour. Conveniently, the latter can be deduced from the obstacle locations already detected, so no additional state prediction is required.

The caveat is the lack of an incentive to turn gradually. If all paths generating the correct amount of turn were equally valid, turning may be postponed without apparent consequences until the robot again comes up against its MTC. A possible remedy is to consider the rate of turn rather than its total, but it is unclear how to define the worst case (i.e. the homeostatic bounds) for turning rate. Even the ideal, average rate can only be calculated with knowledge of which point on the contour the robot should merge with and how long the path to that merge point is. At this stage, it emerges that in fact what is needed is simply a smooth path, for which a strain based solution has already been presented. Applying that solution requires a goal location to be set, meaning that a way of determining suitable contour merge points needs to be devised. Inspiration for how to do so is sought in related literature reviewed as part of this task's background.

There are now two homeostatic variables on which to base the urgency heuristic for obstacle navigation. Contour distance and the strain of a contour joining path do not however share the same denominator. This makes it difficult to use both to parametrise the same heuristic function. Fortunately, the task coordination framework was designed precisely for this situation in which seemingly unrelatable objectives need to be weighed against each other. To take advantage of this, the single conceptual task of obstacle navigation is reformulated in terms of two taxonomy tasks: collision prevention and contour following. Urgency heuristics for each are developed in the eponymous section later in this chapter.

5.1.4 Task Classification

The only reason for approaching an obstacle is if another task benefits sufficiently to justify the risk. Lacking any sort of internal drive, obstacle navigation must have extrinsic value. The same applies to the two taxonomy tasks into which it was divided – although it is easier to see this is the case for the collision avoidance task. Contour following may initially appear to be pursuing its own goal by aiming for a specific merge point. While this is true, it should not be forgotten that the need for doing so only ever arises due to other tasks. Furthermore, the merge point need never be reached. It serves only to encourage the robot to converge on and follow the contour.

As discussed, there are alternatives for the homeostatic variables. Irrespective of which is chosen, the situations to be avoided will always be defined in terms of the robot's position relative to that of obstacles in the environment. Both taxonomy tasks depend on the position of the contour and by extension that of the obstacle, meaning that external state is involved.

Collision prevention and contour following can therefore be classified as category four tasks.

5.2 Background

This section roughly sifts through the plethora of contributions to obstacle navigation in search of solutions that can be adapted to comply with the task coordination system. An outright solution cannot be hoped for due to the already mentioned unique requirement for the complete segregation of task responsibilities. Instead we hope to extricate the relevant parts of existing mechanisms which, all too often, interweave goal seeking with obstacle navigation. Solutions with irresolvable codependencies and/or other shortcomings can be excluded from further consideration. Examples of these cases serve to explain why the same is true for all approaches sharing the undesirable trait.

Khatib's potential fields approach [66] was already reviewed in the context of task coordination. Here, we take another look at it from the perspective of obstacle avoidance for which it was originally intended. Choset et al. [25, p. 77] aptly summarise this technique using an analogy to a charged particle:

> The potential function approach directs a robot as if it were a particle moving in a gradient vector field. Gradients can be intuitively viewed as forces acting on a positively charged particle robot which is attracted to the negatively charged goal. Obstacles also have a positive charge which forms a repulsive force directing the robot away from obstacles. The combination of repulsive and attractive forces hopefully directs the robot from the start location to the goal location while avoiding obstacles.

As can be seen from this description, potential fields make no exception when it comes to entwining goal seeking with obstacle navigation. What sets the approach aside from many others is that it is easy to separate the goal and obstacle components. The gradient vector field that the robot descends on is composed of the superposed (i.e. additively combined) individual vector fields for goal attraction and obstacle repulsion. Using the repulsive obstacle field on its own provides the type of behaviour required for the collision prevention task. The only mandatory adjustment is to replace the particle model of the robot with one able to take the robot's size into account. Indeed, the urgency heuristic developed for collision prevention in the next section is based on the principle of distance based repulsion described here.

In terms of the contour joining task, Khatib's solution has little to offer. With some adaptations, the robot can be made to observe the desired contour distance but the technique cannot induce smooth or early turning. Seeing as the robot gets stuck in concave shapes due to the well-known local minimum problem, it is a far cry from following a contour around the obstacle.

Bug algorithms [57] [58] do not suffer from the local minimum problem and are "amongst the earliest and simplest sensor-based planners with provable guarantees" [25, p. 17]. The robot moves in a straight line towards the goal location until an obstacle is detected by one of its sensors. Upon contact, the particle robot latches onto the obstacle edge like a bug, giving the method its name. Completeness guarantees stem from a set of rules that only allow the robot to relinquish the obstacle under certain conditions. Specifically, sustainable progress has to have been made such that the possibility of a loop, by which the robot returns to a previous location, can be ruled out. Given that there are only a finite number of obstacles and each is visited at most once, the robot must eventually arrive at its destination. There are many versions of bug algorithms with different conditions for leaving the obstacle. For example, [64] improves on the earlier versions cited above by allowing earlier leaving without loss of guarantees.

A remaining problem is that the robot is still modelled as a particle. With the planner assuming it will fit through any gap, collision is inevitable, e.g. when a real robot of finite width tries to move through a narrow passage. The problem may, however, be overcome using obstacle dilation [35, p. 179]. This technique artificially enlarges obstacles by creating a virtual boundary around them. Although the robot is still considered to be a point it will now cling to the virtual boundary, thus not colliding with the real obstacle. Obstacle dilation requires knowledge of the positions and shapes of all obstacles in order to enlarge them, meaning it cannot be used in unknown environments.

Even if a global map of the environment were provided, the traditional bug algorithms named so far produce only geometric paths. That is to say, they assume an omnidirectional vehicle without restrictions on speed, acceleration or minimum turning circle. Since a real robot cannot follow such a path, this technique does not deliver the desired degree of realism.

A more pragmatic approach is taken in Potbug [126] which addresses a number of practical issues. As its name suggests, it is a crossover of potential fields and bug algorithms, which allows it to provide many of the features we require while preserving guarantees. Most importantly, it selects subgoals on a contour at a safe distance around the obstacle. Aiming for these bypasses the local minimum problem in Khatib's approach and simultaneously allows the robot's width to be taken into account without needing a global map. Furthermore, realisable paths are generated in real-time due to the efficiency of potential fields. Goal reaching is quite deeply ingrained in the technique, but the idea of setting subgoals can be used and further improved. The urgency heuristics section explains how the principles used here can be adapted for contour joining.

More recent solutions such as [42] and [122] focus on improving path smoothness but rely on prior knowledge of the environment and are inextricably linked to goal reaching.

5.3 Task State Prediction

A prerequisite for applying any obstacle navigation technique is the ability to localise the robot in relation to features in its environment. Specifically, state prediction for this task involves estimating how far the robot will be away from obstacles after making an imaginary move. The previous chapter already explains how forward kinematics can be used to predict the position of the robot. This leaves only the challenge of inferring the position of nearby obstacles using the robot's sensors. Having gathered all relevant information, obstacle distance can be calculated for the robot's current configuration as well as estimated for future positions reached after applying given controls. Of course, predictions are only valid for samples within the robot's sensor range as a global map is neither provided nor constructed.

5.3.1 Obstacle Detection

A variety of sensors can be used for obstacle detection. They can be grouped into contact sensors, proximity sensors and visual sensors.

Contact sensors (mechanical bumper switches) are the only type not appropriate for this project. This is because they are activated upon physical contact – at which point the obstacle navigation task has already failed according to its present definition.

Proximity sensors include radar, laser range finders, infrared and sonar sensors. All of these have different advantages, disadvantages and areas of application. For example, infrared sensors suffer from interference through sunlight, but such details are not of great concern here. The following assumes the robot is equipped with sonar sensors but is applicable as long as a reasonably reliable source of information can be provided.

Visual sensors, i.e. cameras can offer a lot more than just depth of field. The computational cost of image processing is a disadvantage however, especially as the additional data is not required for present purposes.

Whichever way obtained, the recorded distance values need to be transformed into knowledge about the location of obstacles. The point at which a sensor ray hits an obstacle can be computed from the position of the sensor, the distance it measured and simple trigonometry. Obstacle points thus detected are relative to the robot's frame of reference. Optionally, these points can be expressed in terms of a global coordinate system using a translation by the robot's position and rotation by its orientation. Global coordinates are a requirement for building up a map of accumulated information. When, as in this case, the robot strives not to return to previously visited areas of the environment, there is however little value in doing so.

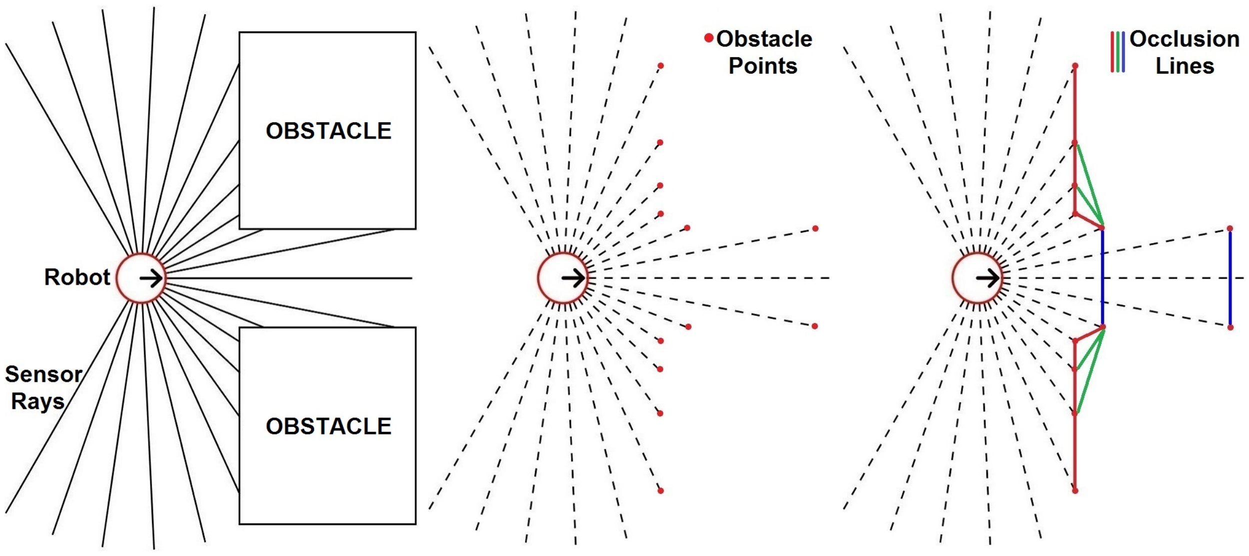
5.3.2 Gap Occlusion

Detected obstacle points can be used without any further processing, as indeed they are in many methods such as Khatib's potential fields [66]. A drawback of this approach is that the AI controller will interpret the gaps between detected obstacle points as free space. While this could be the case, the perceived gap may just be a blank in our knowledge due to insufficient sensor coverage or an invalid reading. Even if there really is free space between adjacent obstacles, it may not be wide enough for the robot to pass through.

Sealing off such gaps by connecting obstacle points that are closer together than the vehicle is wide can prevent collisions caused by erroneous assumptions of free space. This type of occlusion also serves to encourage smoother paths by ensuring the robot does not turn towards every crevice in the obstacles it engages.

One method of filling in the gaps between known points is to use a feature extraction technique such as the "Hough Transform" [8]. Using this, it is possible to approximate the actual shape of objects such as circles, lines and parabolas from a fairly sparse set of edge points.

Linear interpolation of close together points is a crude but more efficient alternative to feature extraction and has been found to be sufficient. The implemented approach is illustrated below using the example of a robot approaching a gap between two obstacles.



(a) Workspace view **(b)** Detected obstacle points **(c)** Occlusion lines

Figure 5.2: Representation and interpretation of distance sensor readings showing the occlusion of gaps too small for the robot to navigate. A bird's-eye view of a robot facing two obstacles with full information is shown in (a). The same scenario is displayed from the robot's perspective in (b) with detected obstacle points marked as red dots at the ends of the black sensor rays. Occlusion lines are sketched in (c).

The algorithm used for deciding occlusion computes the distances between all detected obstacle points, closing any gaps narrower than a pre-set occlusion distance. For scalability with the robot's size, this distance is specified as a multiple of the vehicle's width. The red lines block gaps that are definitely too narrow to navigate, while the blue lines indicate a potential passageway that may or may not be occluded depending on what is considered safe. In the figure above, the occlusion distance is set to twice the robot's width, which just closes the gap but leaves a little recess into which the robot may dip. The actual implementation uses a width-multiple of three which would result in that alcove being smoothed over entirely.

Lines drawn in green are evidently superfluous as they are concealed by the red ones which are closer to the robot. They are created because it is not sufficient to only consider joining obstacle points detected by adjacent sensor rays. The ray passing straight through the gap would otherwise prevent it being occluded by the blue lines shown. Deciding connectivity based on Euclidean distance solves this problem but produces the superfluous lines as a by-product. Determining which lines are hidden from the robot's view may however take more time than is saved by removing them from consideration.

5.3.3 Distance Measurement

Locating obstacles is merely a means to the end of satisfying the framework's requirement for a state prediction mechanism. For obstacle navigation, two mappings, which can together associate control samples with obstacle distance, need to be provided. The vehicle's forward kinematics are used to map any given control vector to the end point of the motion generated by its execution. That end point (or indeed any coordinates) can then be mapped to the Euclidean distance between it and the nearest known obstacle in its vicinity. Once the obstacles have been detected, the distance is trivially found by iterating over all obstacle points and selecting the one closest to the coordinates being sampled. The procedure is the same when working with occlusion lines, only that here the perpendicular distance is computed between a point and a line rather than between two points.

It is important to note that using occlusion lines does not guarantee more accurate measurements. In the ideal case, the obstacle edge is indeed straight (at least between the two known points linearly interpolated) and an exact distance is obtained. When the obstacle is convex, the obtained result will be an overestimate of the actual distance but more precise than using only known obstacle points. Concave obstacles will lead to an underestimate that can be further off the mark than point to point distance. Even in this worst case, occlusion lines are still superior. While accuracy is important, their main virtue lies in removing noise by smoothing over irregularities in the obstacle's shape – not to mention blocking gaps the robot would otherwise get stuck in.

5.4 Urgency Heuristics

Having successfully defined an appropriate state prediction mechanism, we proceed to mapping the obtained state information to an indicator of task success as per the second hypothesis. As such, this section develops urgency heuristics for both obstacle avoidance and contour joining tasks. It also explains in more detail why two separate tasks are required, or more specifically why contour joining cannot fully replace collision prevention.

5.4.1 Collision Prevention

Collision prevention is a task that depends purely on the distance between the robot and the nearest obstacle. For the purposes of distance measurement and collision detection, we assume a circular vehicle of radius r. This can be done without loss of generality as a bounding circle may be substituted for any shape of robot. Obstacle distance must be measured from the centre of that circle to the closest obstacle point or occlusion line. State prediction will provide the distance to the robot's coordinates which are assumed to coincide with the centre point of the bounding circle. Should this not be the case, state prediction must be invoked using the centre of the circle in place of the robot's coordinates. Let the resulting distance be:

$$d_{sample} = distance(circleCentre, closestObstaclePoint) \tag{5.1}$$

With the ability to predict the distance for any sample position, this leaves only the question of how to define the best and worst cases for this task. The worst case is easily defined as the robot crashing into an obstacle. This need not to happen at speed. The task fails as soon as the robot's body comes into contact with the obstacle edge. According to our model, that is the case when the robot's centre point is at distance of r from the obstacle. The worst case distance is thus:

$$d_{worst} = r \tag{5.2}$$

The ideal case is when the robot is in free space, but since this is unlikely to be the case for long, we settle for keeping obstacles at a safe distance. As with gap occlusion, distances should be expressed in terms of the robot's size for scalability. Potbug follows a contour set at a distance of three times the robot's radius, which corresponds to leaving one robot width between the obstacle and the robot's outer body [126, p. 7]. To account for likely compromises, a slightly larger safety margin is appropriate. For this reason contour distance (CD), which is also the task's ideal obstacle distance, was set to four times the robot's radius.

$$d_{ideal} = CD = 4r \tag{5.3}$$

Now equipped with the ideal, worst and sampled distances, urgency for obstacle avoidance can be expressed using the equation suggested in Chapter 3. What this formula yields is effectively the percentage of the buffer zone that the robot has penetrated.

$$proximityUrgency = \frac{d_{sample} - d_{ideal}}{d_{worst} - d_{ideal}} \tag{5.4}$$

Inserting the known constants and simplifying yields:

$$proximityUrgency = \frac{CD - d_{sample}}{CD - r} = \frac{4r - d_{sample}}{3r} \tag{5.5}$$

The function is defined for domain $r \leq d_{sample} \leq CD$ and has the required range of $[0,1]$. Edge cases are $d_{sample} < r$ in which case $proximityUrgency = 1$ (task failure) and $d_{sample} > CD$ where $proximityUrgency = 0$ (task fully satisfied).

5.4.2 Contour Joining

Strain has been identified as a suitable homeostatic variable for parametrising an urgency heuristic capable of encouraging smooth contour joining. The task model developed in a previous section already plays with the idea of enlisting the help of the mechanism underlying the goal location task. Doing so would provide an elegant solution, but requires target locations to be set at strategic locations along the contour where the robot's path can be merged with the course it takes. Here, the remaining pieces of the puzzle are assembled to bring the outlined technique to full fruition. This involves setting merge targets in both directions along the contour and defining *merging urgency* to fairly represent the choice between these alternatives. In addition to developing the method itself, we also discuss when it should be applied, i.e. under which conditions the robot should or should not follow a contour around an obstacle.

5.4.2.1 Setting Contour Merge Targets

Merge targets are moving subgoals positioned on the contour of an obstacle with a heading pointing along it. If well placed, they can guide the robot towards and into alignment with the contour without running the risk of reaching the vehicle's MTC. The attractive force they exert may, however, lead the robot astray in some cases. One such occasion is when the robot is drawn into engagement with an object that was not in fact in its way to begin with. A solution to this problem will be presented later. First, we discuss why using only a single merge target can create an almost arbitrary, yet strong and potentially damaging bias towards a certain direction. The following depicts this scenario.

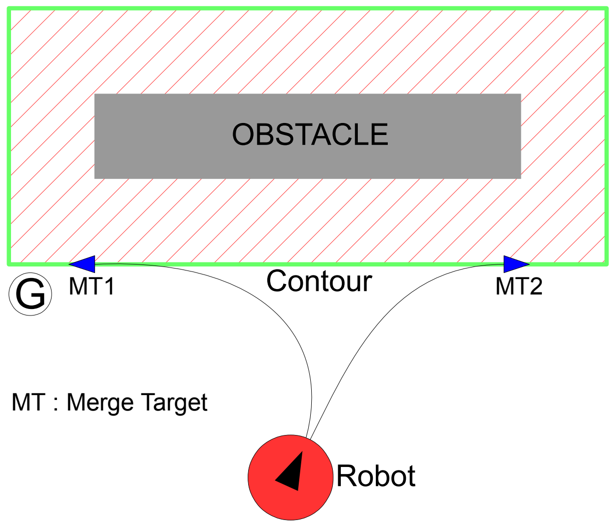


Figure 5.3: Illustration used to explain why fixation on a single merge target can result in needlessly poor decision making. If only one of the merge targets (drawn as blue arrows) is selected, a powerful bias is created towards the direction marginally preferred by the obstacle navigation task. Better founded preferences put forth by other tasks may be overruled due to the false impression that turning away from the single merge target will result in failure.

Two candidate merge targets are shown as blue arrows, although only one of them will be selected. If selection is based on strain as suggested, the right merge target will be chosen and the left discarded. No matter how merging urgency is now defined, it must surely reflect an action's prospects of reaching the selected subgoal. With a left turn representing the polar opposite of the desired direction, the corresponding action will come close to, if not define, the worst case. Any competent coordination system will then make it near impossible for the robot to move left towards supposed task failure. Consequently other tasks' preferences will be dwarfed and largely ignored. For example, to reach the goal location (G) marked in the diagram, the robot would have to go all the way around the obstacle. This is as lamentable as it is unnecessary. The strain of the left arc is only slightly greater than that of the right and represents a very real alternative – especially if other tasks favour that direction. As it stands though, even heading straight for the obstacle will seem more appealing.

The key is to consider both merge targets in the urgency heuristic. Before going into the details of how to cost the alternatives fairly, we first need to find the merge targets to be used. The following presents an algorithm for the geometric construction of two merge targets along the contour of a detected obstacle. An illustration of all relevant points supplements the description.

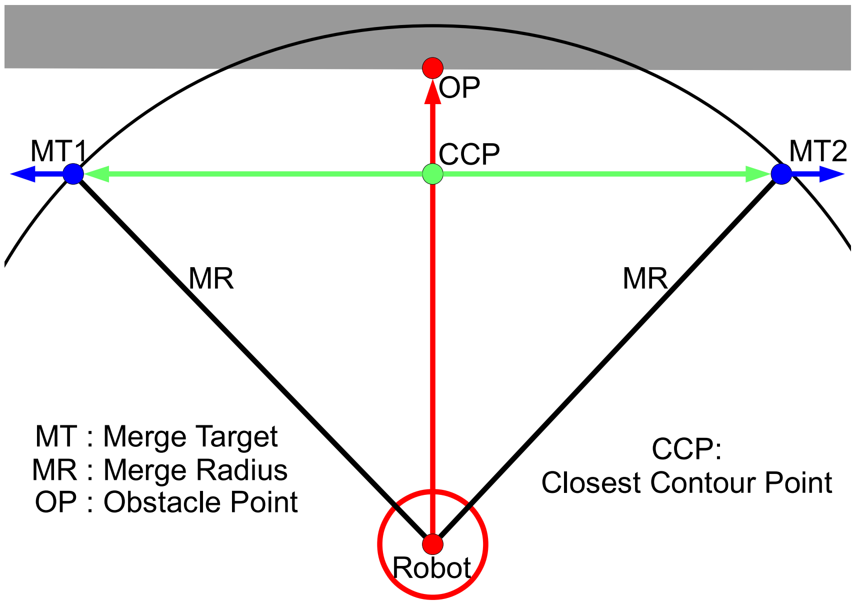


Figure 5.4: Geometric construction of merge targets based on the robot's position in relation to the closest detected obstacle point

1. Use the described state prediction mechanism to obtain the coordinates of the robot's centre point (R) and the obstacle point (OP) closest to it.

2. Let $\vec{o}$ be the obstacle vector from R to OP (depicted as a red arrow).

3. To the end of $\vec{o}$, add a contour vector $\vec{c}$ going in the opposite direction with length CD.

4. The end point of $\vec{c}$ is the closest contour point (CCP) in relation to the robot's position R. Note that this point can lie on either side of the robot depending on whether or not it has already penetrated the contour.

5. Let $\vec{l}$ and $\vec{r}$ be orthogonal to $\vec{o}$ and pointing to the left and right of the CCP respectively (see green arrows).

6. Scale $\vec{l}$ and $\vec{r}$ until they reach the intersection with the merge radius (MR). The required vector length of $\sqrt{MR^2 - d(R,CCP)^2}$ is obtained from the right-angled triangle $\angle CCP\ R\ MT_1$.

7. The two merge targets (shown in blue) are defined by the ends of the scaled vectors $\vec{l}$ and $\vec{r}$, the directions of which also specify the target headings.

The only constant remaining to be chosen is the merge radius. Like the contour distance previously defined, the merge radius should also be expressed as a multiple of the robot's body radius. With nothing to go by, an appropriate value was found through experimentation, which suggests that merge targets should be chosen approximately four robot lengths ahead of the vehicle. Again taking the robot's size into account, we set $MR = 9r$.

It is acknowledged that the described geometric construction algorithm only produces merge targets that fall exactly on the contour when the obstacle is straight-edged. For curved obstacles, the merge targets may be closer to, or further from, the obstacle's edge than they should ideally be. More accurate coordinates could be found through search, but the effort has proven to be unwarranted. Precision is not paramount because the robot's step size is small and the merge targets are continuously updated while travelling along the contour. Because the selected target need never be attained, it suffices as long as it provides a reasonable basis for local decision making. As is evidenced by the experiments conducted in the following section, the desired behaviour can be achieved as a result of chasing a moving goal that remains a fixed distance ahead of the robot.

5.4.2.2 Merging Urgency

With a mechanism for generating merge targets now at our disposal, we can define merging urgency in terms of the strains of the paths leading to them. In fact, it is not even necessary to get to urgency via strain. The urgency function developed in the previous chapter can be applied directly with the merge targets set as the goal locations. This saves having to specify new ideal and worst cases, since the returned values are already in the interval $[0,1]$. Moreover, path smoothness properties are automatically inherited, meaning that urgencies will not only indicate how to join the contour but how to do so gradually.

For a given control sample the obtained urgencies are:

$$urgency1 = goalUrgency(MP_1, sample) \tag{5.6}$$

$$urgency2 = goalUrgency(MP_2, sample) \tag{5.7}$$

The problem, of course, is that there is a separate urgency for each of the merge targets when actually a single overall heuristic is desired.

A simple remedy is to combine both goal location urgencies into merging urgency by averaging.

$$mergingUrgency = \frac{goalUrgency(MP_1, sample) + goalUrgency(MP_2, sample)}{2} \tag{5.8}$$

Unfortunately, the resulting cost metric struggles to distinguish clearly between different control samples. This is because the urgency for going to the left merge target increases as the robot turns right, while the urgency for reaching the right target decreases. The result is that the opposing preferences effectively cancel each other out (more or less depending on the robot's angle to the obstacle).

A better approach is to cost a sample with regard to how well it does at joining the contour at *either* merge target. This more accurately represents the choice the robot has. We assume that a left turning sample will continue moving left and a right turn will in fact proceed to the right merge target. Merging urgency is then the minimum of the two goal urgencies:

$$mergingUrgency = min(goalUrgency(MP_1, sample), goalUrgency(MP_2, sample)) \tag{5.9}$$

The urgency thus defined can be visualised as a conic shape which the robot is discouraged from entering.

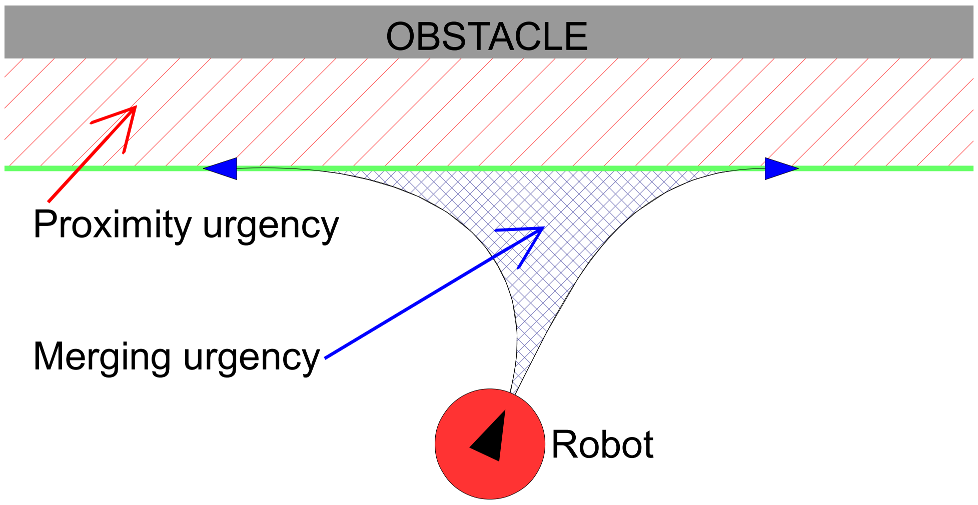


Figure 5.5: Visualisation of proximity and merging urgencies as areas in workspace

The above shows both proximity urgency (red striped area) and the newly introduced merging urgency (blue hatching). To avoid entering the blue area, the robot will turn smoothly towards one of the merge targets. In the absence of other tasks it will choose the merge target associated with least strain, but is also ready to strike a sensible compromise in the face of conflict.

It is interesting to observe that contour joining defines an undesirable region of state space just like obstacle avoidance. This helps understand why both are category four tasks despite one of them actually building on the goal location task, which is a prime example of category one.

5.4.2.3 Engaging

Returning to a previously mentioned problem, consider the prospect of a merge target attracting a robot towards an object that is not an obstacle to it. Unlike proximity urgency that only acts to repel a robot already close to the obstacle, a merge target exerts an active pull. With a fairly large merge radius ($MR = 9r$), it is not unlikely for a merge target to be created along the contour of a fairly distant object. Under normal circumstances the robot may simply pass by the object, but due to the merge target it will be drawn into its orbit and onto a detour. As a preventative measure, obstacle engagement is restricted to situations in which the robot's path is in fact obstructed. Two conditions must be met for an object to be deemed an obstacle.

Firstly, only obstacle points within a cone in front of the robot will be considered in the decision to engage. In other words, the robot must be driving towards the object for it to be deemed an obstacle. Centring the cone on the robot's heading with a width of around 80° to 90° has proven effective. Relevant obstacles are engaged while those in the peripheral vision are safely ignored.

Secondly, an engage radius (ER) is introduced. Obstacle points detected outside of a circle around the robot are likewise not considered as requiring action. This leaves the robot some time to change its course before committing to following the obstacle. The engage radius must be larger than the contour distance ($CD = 4r$) and smaller than the merge radius ($MR = 9r$). If it were smaller than the contour distance, obstacles would be engaged too late for smooth joining to be possible. On the other end of the spectrum, an engage radius larger than the merge radius would mean this condition poses no restriction as all merge targets would be in range. Between these extremes there is quite a lot of flexibility. A value of $ER = 7r$ works well.

Unless both of these conditions are met, no merge targets will be generated and consequently there will be no merging urgency. The robot is then able to continue on its path in perceived free space without being led astray. It should be noted though that the cone is only used to decide whether or not to enter the *engaged mode*. Samples outside of the cone are not discarded, meaning that all obstacle points detected are still considered in merge point construction once the robot has engaged.

5.4.2.4 Disengaging

Having formally defined a state of engagement, there must also be a way of disengaging so as not to chase merge targets around the same obstacle in perpetuity. Even if the urgencies of other tasks may eventually overcome the pull of the obstacle, the contour would likely be followed for much longer than it should be. To avoid forcing other tasks into making needless compromises, the situations in which the contour can be left need to be identified. While leaving too late certainly impacts other tasks, leaving too early may result in the robot falling back into a concave out of which it cannot escape. Fortunately, BUG algorithms have already addressed this problem by defining conditions under which the robot can and should disengage. The following three conditions were adapted from those (especially [64]), but may vary between different members of the BUG family.

All of the following conditions must hold for the robot to disengage:

1. The robot is facing away from the obstacle.
2. Moving forwards in the direction the robot is facing reduces the distance to the goal that the obstacle was obstructing.
3. The robot is currently closer to that goal than when it engaged the obstacle.

Sticking to the obstacle edge and these (or similar) conditions, BUG algorithms can guarantee to make lasting progress. With only a finite number of obstacles between the robot and its goal, completeness is assured. In the context of task coordination these guarantees are, however, reduced to a best efforts policy. Interference from other tasks is unpredictable, making it impossible to rule out a forced departure from the prescribed path.

Realistic restrictions on acceleration and deceleration rate may also cause a diversion from the contour. For example, the robot may have accelerated along a straight edge. When the obstacle finally curves round allowing the robot to get to the other side of the obstruction, the vehicle is moving too fast to make the required turn. It overshoots while trying to decelerate but finds itself in free space facing away from the obstacle which is now out of sight. Even if not all conditions are met, the robot must disengage in this situations as there are no merge targets to aim for. See figure 5.11 for a depiction of the describe scenario.

5.4.3 The Need for Two Tasks

The experiments shown in the following section may give the false impression that the obstacle avoidance task is effectively subsumed or replaced by contour joining. This illusion is created due to the latter always being able to pursue its favoured actions in the absence of other tasks, i.e. under the circumstances in which the tests are conducted. In this situation, contour joining is indeed sufficient to achieve and maintain the desired, safe contour distance. As soon as other tasks force a compromise action, success can no longer be assured. Figure 5.6 illustrates a conflict of interest that would cause collision were the obstacle avoidance task not explicitly represented.

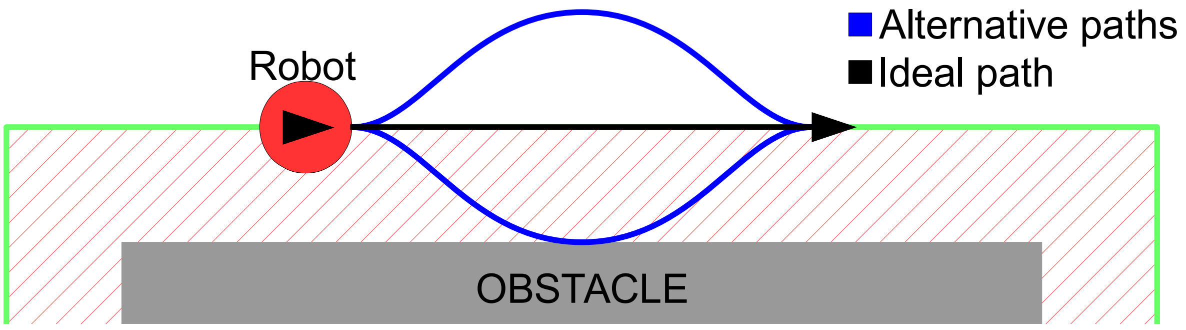


Figure 5.6: Compromise paths (blue) with equal merit in terms of contour joining despite the lower trajectory leading to collision without intervention by the obstacle avoidance task which is able to disambiguate

In this example, the robot is already on the contour and would elect to continue along it by following the black line shown. The blue curves represent identical, but mirrored alternatives that may be advocated for by other tasks. Due to symmetry, both curves must have the same strain and consequently also produce identical goal location urgencies. Evidently though, the lower path leads to collision, demonstrating that the contour joining task, despite appearances, has no concept of obstacle avoidance and would be insufficient on its own.

5.5 Experiments and Results

This section documents the experiments carried out to confirm the validity of the solutions to both collision prevention and contour joining. Tests were run using the robot simulator described in Chapter 7, from which the screenshots shown here were taken. All results are for the tasks in isolation so as to rule out interference from other sources.

5.5.1 Collision Prevention

To begin with we examine how the collision prevention mechanism can solve the aforementioned problem, i.e. prevent the contour joining task striking compromises leading to a collision it is unable to foresee.

Consider the experiment in figure 5.7 where the robot starts facing along a wall which it is dangerously close to. To re-establish a safe contour distance, the vehicle immediately starts turning left. The realistic acceleration model precludes an instant left MTC turn, although this would be ideal. Instead the curvature of the path increases gradually as the robot accelerates. The collision prevention task is satisfied as soon as the robot reaches the contour, marked here with a green dot.

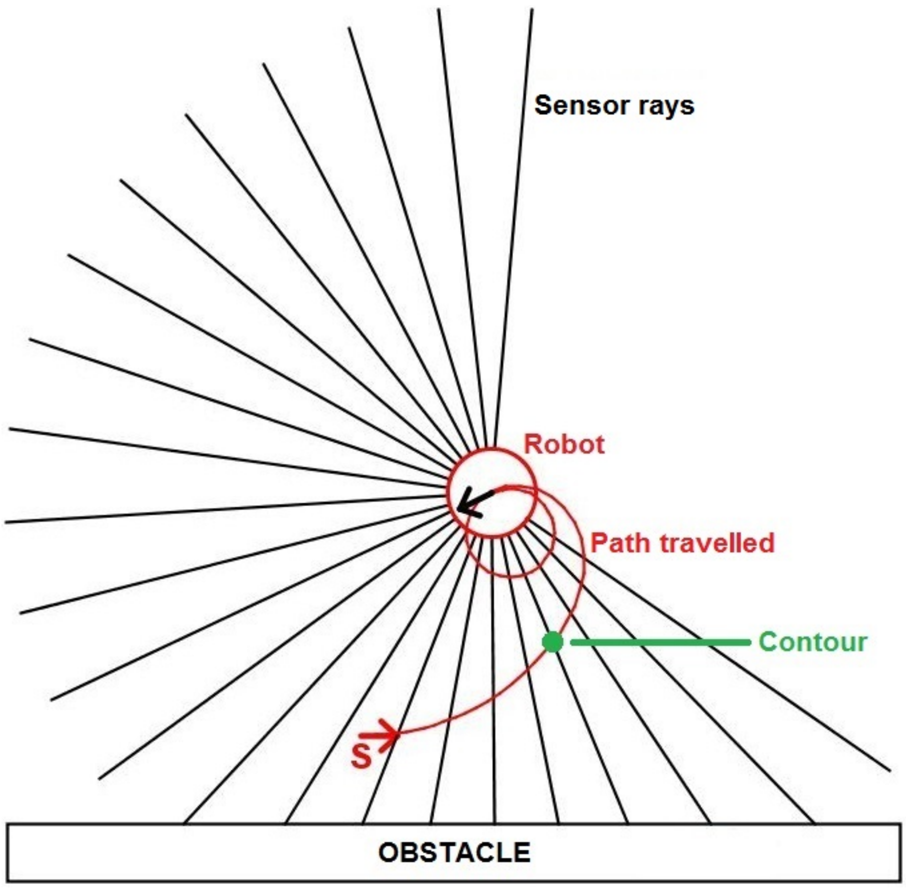


Figure 5.7: Re-establishing a safe obstacle distance by turning away from a dangerously close obstacle. After reaching the contour (green dot) behaviour is undefined due to the absence of unsolved tasks. In this case the robot decelerates while continuing to turn left.

Once the contour has been reached, there are no more active tasks to influence the robot, leaving it unguided and its behaviour undefined. One may expect it to stop immediately when there are no more tasks to solve but even if there were a motivation for stopping, it would be illegal to do so. The robot cannot stop abruptly because it has to decelerate as gradually as it accelerates. Instead the robot goes into a left MTC spiral and slowly decelerates because, in the absence of any active heuristics, the controller defaults to selecting the slowest legal left turn. While the resulting path may be unexpected, this is by no means an error but simply an implementation specific artefact. Were there still other active tasks, their goals would be pursued.

The above experiment is not repeated for different types of obstacle as the behaviour is always exactly the same.

5.5.2 Individual Obstacles

Unlike collision prevention, contour joining results in different paths for different obstacles. First, consider navigating individual obstacles of different shapes and sizes.

The following shows successful circumnavigation of a regular square and an irregular polygon. It is interesting to note that the path around the square is a rotated square. This is explained by the controller trying to follow the contour exactly but lagging behind due to the acceleration limit that introduces a type of phase shift.

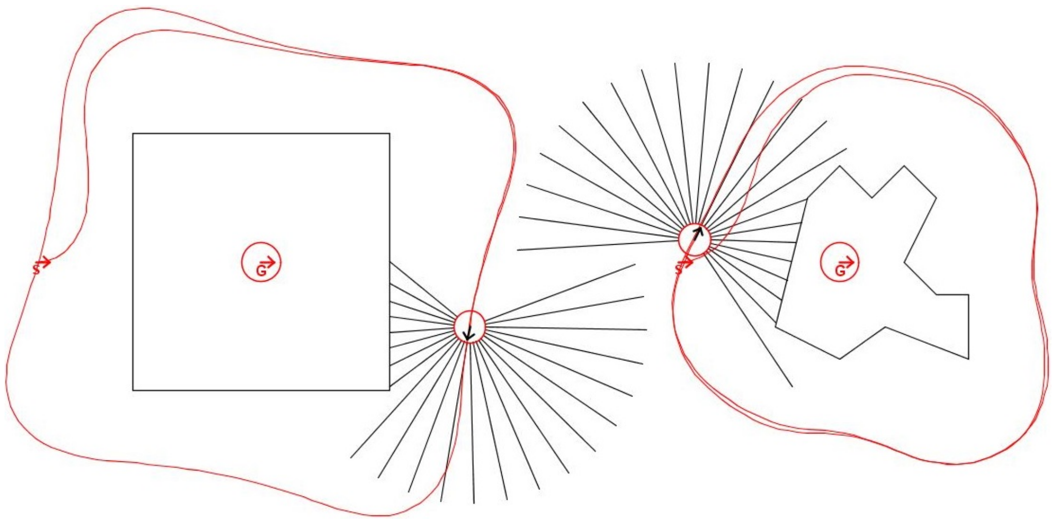


Figure 5.8: Smooth contour joining and following for polygonal obstacles

The effects of obstacle occlusion are demonstrated next. Trying to navigate the star shape on the right would normally result in a very jagged path due to high variations in the readings from the distance sensors. Gap occlusion smooths out these irregularities and facilitates the generation of a path almost as smooth as that around the circle shown on the left for comparison.

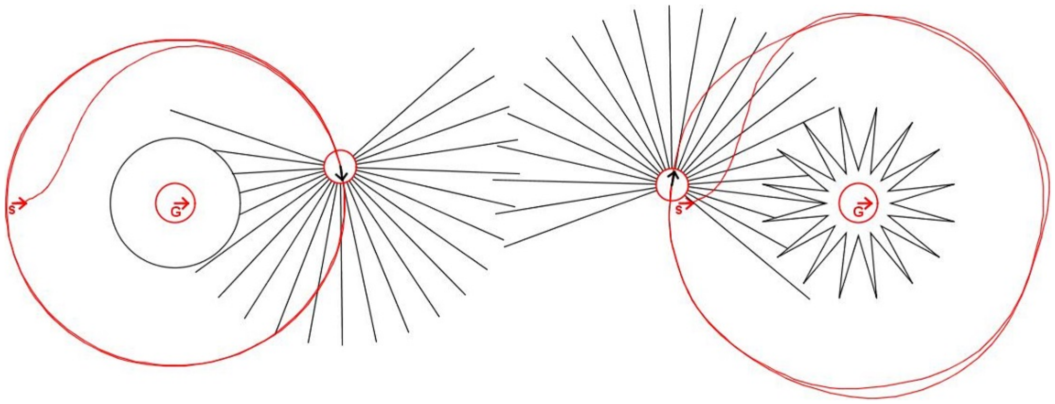


Figure 5.9: Smooth contour joining and following for smooth and jagged circular obstacles

Navigation of the following concave is made possible by the BUG-like features of the developed solution. A standard potential fields based technique would get stuck in a local minimum, but by aiming for merge targets moving along the contour, the obstacle can be successfully rounded.

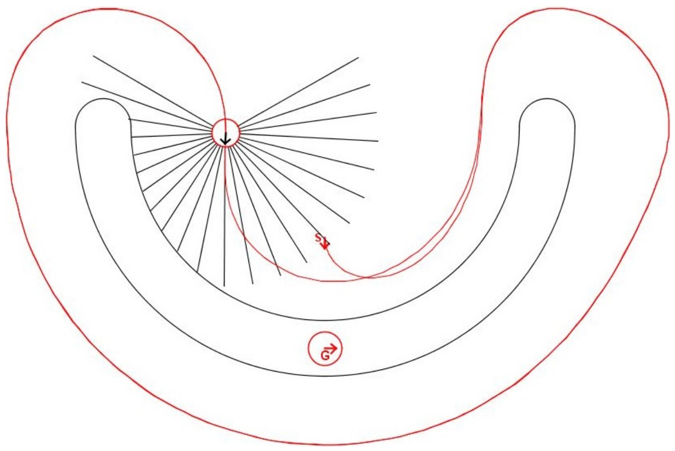


Figure 5.10: Smooth contour joining and following for a concave obstacle

Next is an example of forced disengagement, an eventuality already mentioned after the discussion of the situations in which the robot should disengage. At first the robot has no trouble going around the triangle, but by the second lap it has accelerated too much to manage the turn at the acute point of the triangle. While still decelerating, the robot overshoots the obstacle and loses sight of it. Suddenly finding itself in free space, there is no task left to solve and the controller decelerates going round in a left spiral for reasons already explained. If another task were still active it could now be pursued without interference, so the overshoot is not a problem. Even if the obstacle were still in the way, it would simply be re-engaged as needed. Theoretically, overshoot could occur every time the obstacle is engaged. This pathological case is as unlikely as it is that the robot will re-engage at exactly the same point and with exactly the same speed each time. Nevertheless, this is one of the reasons for which completeness cannot be guaranteed.

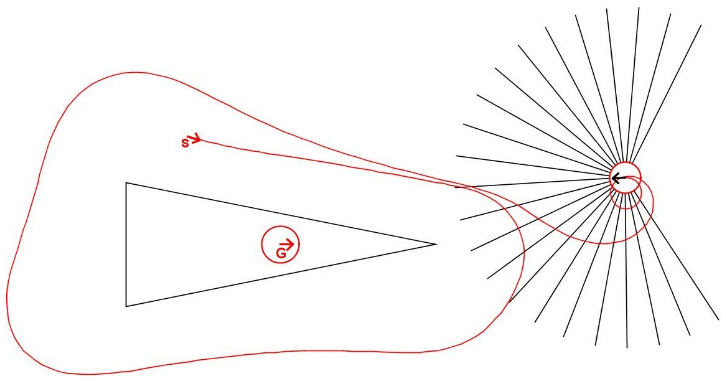


Figure 5.11: Smooth contour joining and following for an acute triangle

The obstacle the system struggles most with is an extreme V-shape in which it is easy to get stuck. Although this is a concave, that is not what causes the issue. The root of the problem lies in the algorithm for setting merge targets. As the robot enters the V-shape it is closer to one of the V-lines than the other. The side it is closer to defines the contour which the robot will follow into the narrow (see figure 5.12). By the time the opposite V-line is closer and the merge target switches to the other side, it is already too late to initiate a U-turn to escape the funnel. The problem can be alleviated by setting a larger maximum occlusion distance to close up the narrow part of the V-shape. Relaxing the acceleration limit to allow the robot to perform tighter turns to get back out of the cul-de-sac also helps. What is really desired though, is a more sophisticated mechanism capable of recognising the need to switch to the other side of the V-shape sooner. For present purposes the suggested remedies are sufficient, as evidenced by the successful run shown on the right. Nevertheless, improved merge targets represent a more elegant solution.

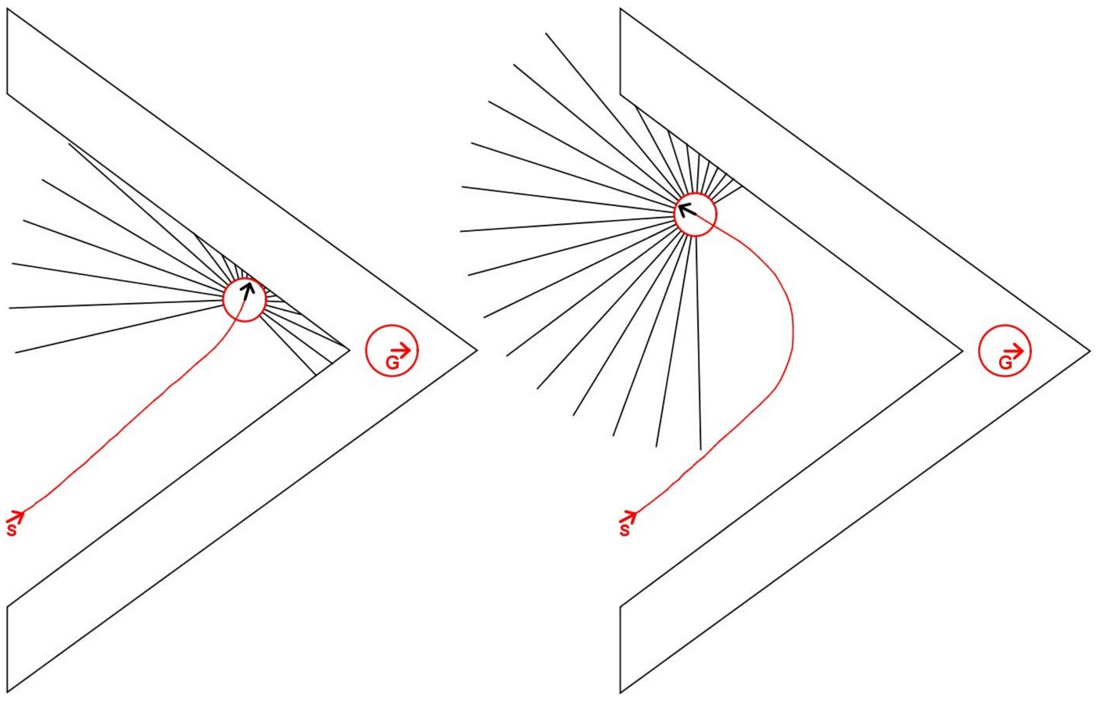
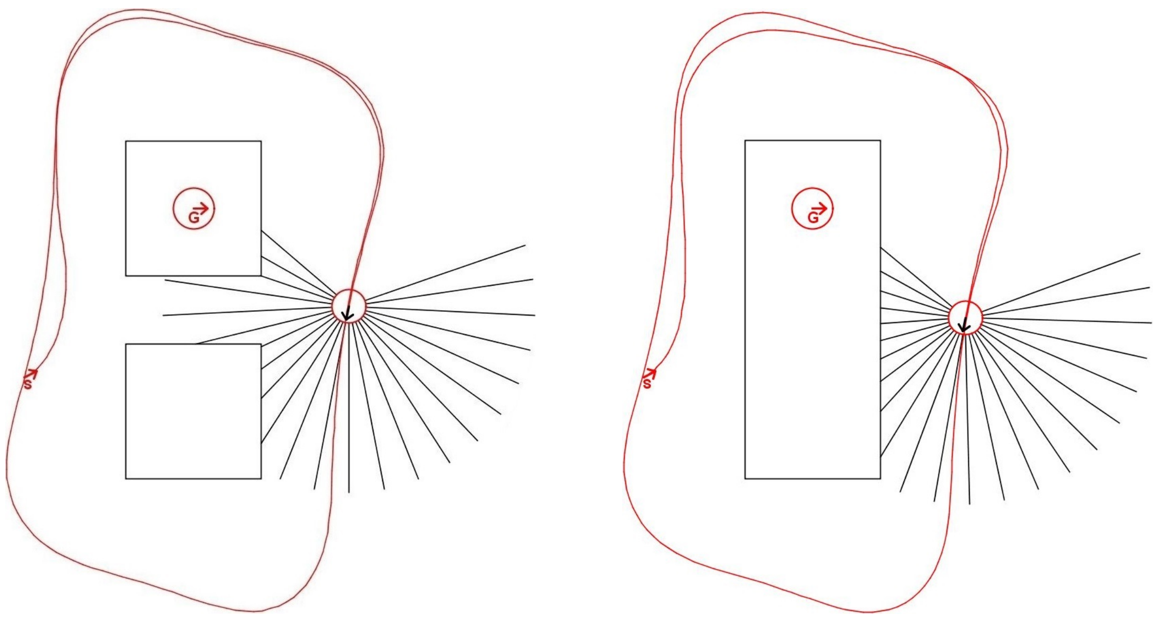


Figure 5.12: Smooth contour joining and following for a V-shaped obstacle demonstrating how a short occlusion distance can lead to collisions that can be prevented using a more appropriate setting

5.5.3 Gaps Between Obstacles

The ability to differentiate between passable gaps and dangerously narrow ones is tested here.

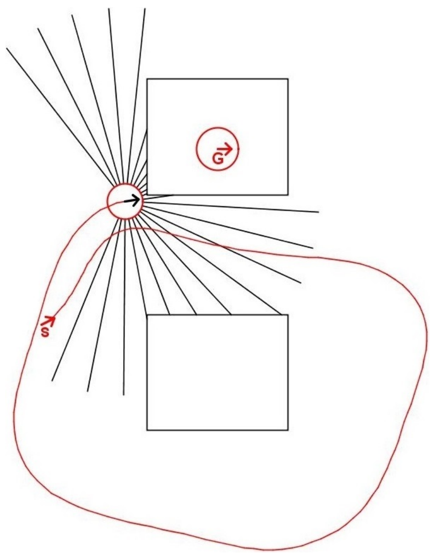
Figure 5.13a is an example of a gap that is theoretically passable but too narrow for passage to be deemed safe. The robot correctly ignores the gap, behaving as through it were a solid rectangle. Upon close comparison with the actual rectangle (figure 5.13b), an almost imperceptible difference can be noticed when looking closely at the top left curve of the generated path. This difference can be ascribed to the fact that the gap is not occluded at all times. For occlusion to work, the gap must be straddled by two detected obstacle points. At times when only one of the two squares is in sensor range, the gap will momentarily be open.



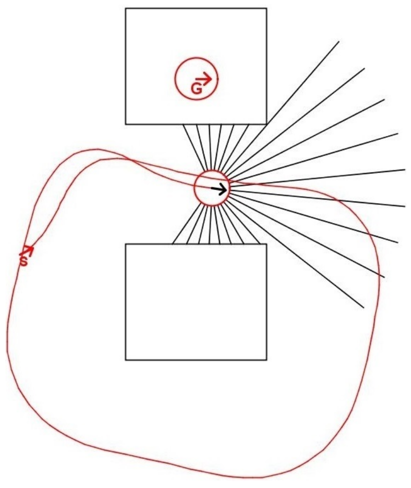
(a) Occlusion of a small gap **(b)** Solid object for comparison

Figure 5.13: Successful occlusion of a narrow gap with behaviour almost identical to that produced when following a comparable solid obstacle

Now consider the same scenario with a wider gap (see figure 5.14). The screenshot on the left shows the robot seizing the opportunity to pass through a gap that was recognised as being open. After completing one circuit of the obstacle the robot returns to the open gap. Again it decides to go through but this time it fails and collides with the upper obstacle. The reason is a very restrictive acceleration limit that was deliberately set for demonstration purposes. On first encounter with the gap the robot is still travelling slowly, having just accelerated from a standing start. Due to the relatively low speed, the vehicle's dynamic MTC is still sufficiently small to pass through the gap. The second time round, the robot has had time to accelerate to a speed that no longer permits executing the tight turn required. Since the contour joining task is unaware of obstacles posing a threat, the controller makes best efforts to turn and ultimately crashes. The experiment is repeated with a more reasonable acceleration limit which leads to success as can be seen in the image on the right. One may also observe that the ability to change speeds more rapidly allows for a path that more closely follows the contour.



(a) Open gap, too restrictive acceleration limit



(b) Open gap, appropriate acceleration limit

Figure 5.14: Navigation of an open gap showing the effect of different acceleration limits

5.6 Summary and Evaluation

This chapter provides a solution to obstacle navigation. To satisfy the objectives of collision prevention and smooth contour joining, two corresponding taxonomy tasks are developed. A state prediction mechanism is provided for both and a separate urgency heuristic is derived for each.

Obstacle avoidance depends on the distance to the closest obstacle edge detected by the robot's sensors. Smooth contour joining is based on strain and reuses the solution to the goal location task in the computation of its urgency. Merge points to either side of the robot are considered, allowing for fair and unbiased compromise with other tasks when it comes to deciding on a direction in which to follow the contour.

Experiments see both tasks fulfil their intended responsibilities. Obstacle occlusion allows a wide variety of irregularly shaped obstacles to be navigated smoothly.

The first and second hypotheses are substantiated by the successful definition of and solution to this classical robotics problem in a manner compatible with the task coordination framework. The fourth hypothesis is upheld by eliminating the need for significant foresight in joining the obstacle contour.

In light of this, both obstacle avoidance and contour joining tasks are deemed fit to act as demonstrators for task coordination.

CHAPTER SIX

BALL BALANCING TASK

This chapter introduces ball balancing as the third and final task. The structure mimics that of the two previous chapters and comprises the same six sections.

The task specification gives a definition of the task to be solved along with concrete objectives. It also discusses how ball balancing can be classified in terms of the task taxonomy and which principles are most suited to informing a solution. Literature on the approach to ball balancing taken here is sparse but related work includes some similar problems such as pole balancing, which was the inspiration for creating this task. State prediction is the main section in this chapter. A custom physics model is developed to simulate and predict the behaviour of a ball in a moving spherical bowl. The same formulae used for state prediction are also relevant to the following section, which derives an energy based heuristic function quantifying the urgency with which ball balancing needs to be tended to. Experiments for this task in isolation focus on becalming an initially perturbed ball. Success is demonstrated by controlling the robot in such a way as to counterbalance the forces acting on the ball. The final section summarises the main features of this task and evaluates its suitability to task coordination.

6.1 Task Specification

6.1.1 Task Description

The ball balancing task is easily described as that of keeping a ball within a shallow bowl mounted atop the robot. When the robot accelerates, forces act on the ball causing it to move away from its resting position at the bottom of the bowl. The latter is modelled as a spherical cap that is open towards the top. This allows the position of the ball within the bowl to be defined in terms of spherical coordinates (r, θ, φ). Since the radius of the bowl stays fixed, only θ and φ change as the ball moves. The zenith angle, θ, indicates the height of the ball, while the azimuth angle, φ, describes its geographic position in the horizontal plane. The task of keeping the ball from falling out of the bowl can then be described as keeping its zenith angle below that of the rim of the bowl. Difficulty can be adjusted by raising or lowering the height of the rim.

Keeping the ball balanced or re-balancing an already perturbed ball requires control over the robot's acceleration. The ball is subjected to the types of forces that people experience in a car where acceleration, breaking and sudden changes in direction can be felt. While these forces are responsible for agitating the ball, they can also be used to becalm it. To do so the robot should select controls that result in forces that oppose the motion of the ball and rob it of its energy. Conflict arises due to other tasks also demanding control over speed and direction of the vehicle.

6.1.2 Objectives

The main objective of this chapter is to create a visual and intuitive task for demonstrating the developed coordination system. Ball balancing is the embodiment of homeostatic stability and an indicator of coordination success that goes beyond the traditional goal and obstacle problems.

In addition to its value as a demonstrator, the developed balancing abilities also find practical application. The example of a waiter balancing a tray was used in the introduction to explain the need for SMT. As with the ball, balance is kept by controlling the forces of acceleration. The same principle can be applied to ensure comfort of ride in a self-driving car or, for that matter, the stability of the robot itself. For instance, a humanoid could easily lose balance when not coordinating its movements. Ball balancing provides an abstract representation of all these possible applications and demonstrates the concept rather than a specific incarnation of the issue.

Finally, the solution developed in this chapter contributes towards verifying the first and second hypotheses. Balancing is an example of a meaningful task that fits the task taxonomy, thereby encouraging faith in the first hypothesis. The second hypothesis is addressed by providing a working physics engine for ball movement prediction and a corresponding urgency heuristic.

6.1.3 Task Model

As in the preceding chapters, the task needs to be modelled in terms the machine can understand. Again this is achieved by formulating the objective in terms of variables that can be placed under homeostatic control. While the conceptual task is readily understood, it is not immediately clear which factors the urgency heuristic should depend on. Various options including existing configuration variables as well as derived values are considered.

As a human performing this task, one would keep the following factors in mind:

- The distance of the ball from the centre of the bowl
- The maximum height the ball reaches when moving in the bowl
- The angles between which the ball swings (θ)

All of these are essentially the same in the sense that they describe the ball's location, albeit using different variables. Since the ball's theta angle is part of the robot's state already, it would make sense to use it. It does not require any values to be derived or converted from spherical to Cartesian coordinates.

There is, however, a problem with all purely positional approaches: The speed of the ball is not considered and as such a ball speeding through the bottom of the bowl would be associated with zero urgency. Unlike a ball resting at the bottom, which really is the ideal case, a fast moving ball will soon pass through the centre and climb up the opposite side of the bowl. If the ball is moving fast enough this may even cause the task to fail, meaning that an urgency of zero is totally inappropriate.

Having recognised the significance of the ball's speed, one may decide to base urgency on it. However, speed alone is also insufficient. As the ball reaches the apex of its motion, its speed will decrease to zero before rolling down in the other direction. Consequently, a ball that stops just short of the rim of the bowl before changing direction will be assigned zero urgency. A small nudge in the wrong direction could cause immediate failure, but, even if that does not happen, the ball will gather a significant amount of speed when rolling down.

Since neither position nor speed are sufficient on their own, urgency will have to take both factors into consideration. The problem is that these indicators are different measures that cannot simply be added. At this point one may consider decomposing the balance task into two taxonomy tasks, but fortunately there is a more elegant solution. Converting the ball's height into potential energy and its speed into kinetic energy gives two comparable values in joules. These can now be added to yield a single objective value. As long as the energy of the ball stays below a certain threshold, it cannot fall out of the bowl.

6.1.4 Task Classification

Ball balancing can easily be identified as a task with extrinsic value. In the absence of other tasks, the robot would not have to move and the ball would never be agitated. Even if the ball were initially in motion, it would lose energy due to friction and eventually come to a standstill without the robot taking any action.

Depending on whether the task is concerned with internal or external state, it would be classified as category three or category four respectively. Interestingly, this distinction depends on the interpretation of the task. In the literal sense, the ball is not part of the robot as it is not firmly attached to it. This would mean the task is interacting with the external environment and therefore in category four. Here, the ball is seen as an indicator of the robot's balance however. This is an internal parameter, meaning the task is placed in category three. For practical purposes this distinction is of little consequence. The only difference it makes in the implementation is whether the ball's state is part of the robot's configuration or the environment.

6.2 Background

When it comes to ball balancing in the present sense, there is very little background to speak of. Some approaches to balancing other objects exist but are not applicable here. The problem is that the physical properties of different objects vary significantly and require different solutions. For this reason, there is no close correspondence in the literature to the approach taken here, which was specifically designed as a demonstrator task for SMT. Nevertheless, the following gives some background on related work, including pole balancing, which was the inspiration for this task.

Searches for keywords such as *robot ball balancing* turn up results pertaining almost exclusively to a particular class of robots called "Ballbots". Ballbots are omnidirectional vehicles propelled by a single spherical wheel which is also their only contact with the ground (see figure 6.1). Lauwers et al. describe the development of "Ballbot" in [73] and [74].

Since these projects are concerned with balancing the robot on a ball and not the other way around, they have little in common with the task presented in this chapter. However, Ballbot does show how acceleration can affect balance and demonstrates that it can be restored and maintained through controlled motion.

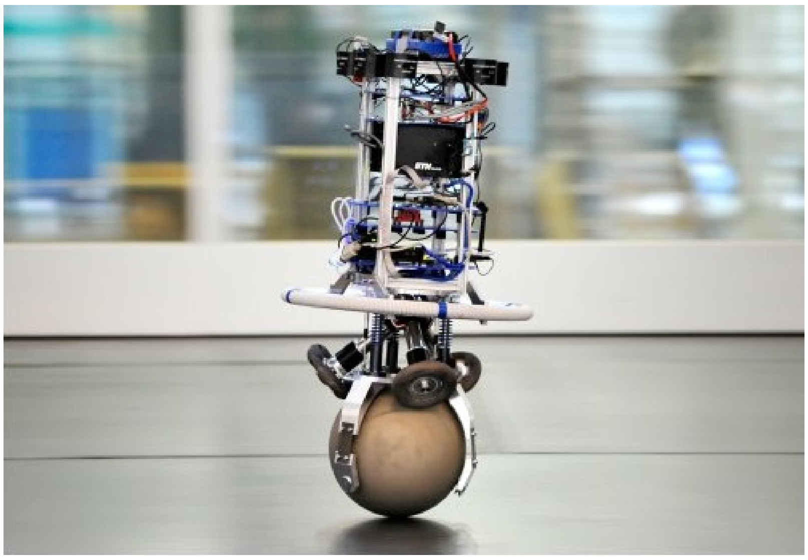


Figure 6.1: Ballbot "Rezero" balances on a sphere which is at the same time its only means of propulsion (image taken from [52])

The Stewart Platform [115] is another example of a machine capable of achieving balance through controlled motion. This parallel manipulator uses six linear actuators to support and control a platform on which objects may be stabilised.

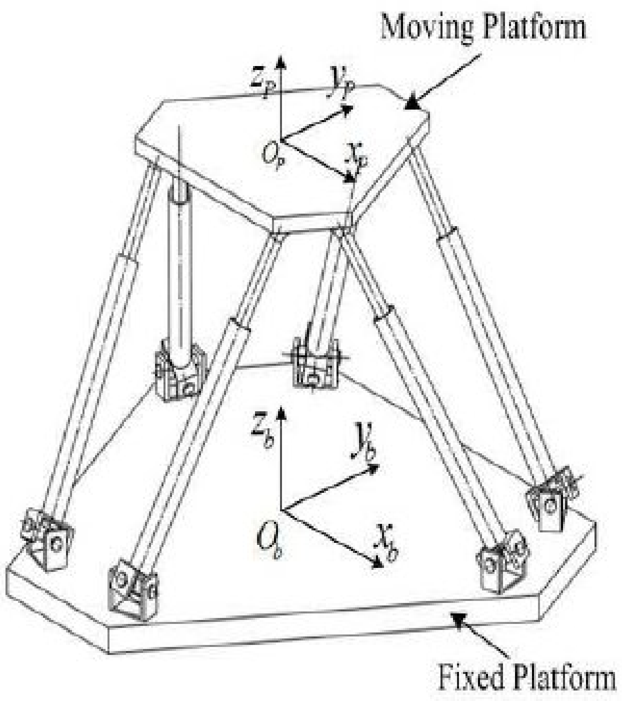


Figure 6.2: The Stewart Platform is able to stabilised and rebalance objects on top of it using six linear actuators to compensate for perturbation and arrest unwanted movement (image taken from [72])

A video by “Full Motion Dynamics” [39] shows how a ball can be balanced on a Stewart Platform. While the mechanism is proficient at maintaining balance, it would be impractical to install such a platform on most robots. In any case, it circumvents the problem rather than solving it. The goal here is not to use additional hardware to allow the robot to ignore the balancing issue, but to show that the robot can itself take care of balancing while also performing other tasks.

Taipalus [119, p. 1] describes a balancing problem involving liquids: “Consider the scenario where a human or a robot is moving a glass filled with a liquid substance. The human would intuitively avoid spillage depending on the environment, urgency of the movement or value of the liquid, while the robot would have difficulty even noticing if it spills something”. However, the problem is not further addressed.

Pole balancing is the closest counterpart to the task developed throughout this chapter. It involves an agent, traditionally a small cart, moving back and forth in order to keep a pole hinged to its top in as upright a position as possible [22]. This is also known as the pole-cart, stick balancing or inverted pendulum problem. Given the similarities to ball balancing, one may even consider using this task in its stead. One reason for which this may seem appealing is the prospect of reusing existing solutions to the problem. For example, Hougen et al. [54] propose a mechanism for training a neural network to control the cart. Provided with an input vector encoding the current state of the system, the trained network outputs a single torque value, the sign of which determines the direction of motion. The problem with this design, and generally all approaches that produce controls for direct execution, is that no provision is made for assessing the quality of alternative actions. As already established in Chapter 3, this is a basic requirement for compromise formation and the reason for which such techniques cannot be used directly. The system could be modified to output a rating in place of controls, but there are further limitations inherent to the task of pole balancing that cannot be overcome. Its high volatility demands almost undivided attention and dominates the robot’s behaviour by forcing it to change direction rapidly in order to prevent the sensitive pole from toppling. This leaves little leeway for compromising with other meaningful tasks. In any case, pole balancing can only be paired with tasks operating in the same one dimensional space in which it operates.

Ball balancing was conceived as a more flexible and versatile alternative with wider application beyond cart-like robots limited to moving horizontally in one dimension. While the ball is volatile enough to pose a significant challenge for an AI system to control, its behaviour is not too erratic for task coordination to be feasible. Since the task described does not appear to have been solved, a physics model for simulating the behaviour of the ball is developed from first principles. The following section relies on formulae that can be found in the mechanics section of most physics books (e.g. “Fundamentals of Physics” [48]).

6.3 Task State Prediction

This section develops a physics model for predicting the next position of the ball after the execution of a given control vector. The following discusses the forces acting on the ball and how they can be quantified using known values.

6.3.1 Known Information About the Robot's Position

Information about the robot's movement is required to work out the direction and magnitude of the translational acceleration acting on the robot and the ball it is carrying.

The robot's current position is obtained from its sensors as a vector in the XY-plane on which it is travelling. Let $\vec{P}(t)$ denote this position with t representing the current time step.

Keeping a record of the coordinates the robot passes through allows previous positions to be looked up. The robot's position in the preceding time step $(t-1)$ is given as $\vec{P}(t-1)$.

After applying a given set of controls, the robot's new position will be

$$\vec{P}(t+1) = fk(\vec{P}(t), controls, TSL) \tag{6.1}$$

where fk represents the forward kinematics function for the vehicle and TSL is the length of a time step (in seconds) i.e. the length of time for which the controls are applied.

6.3.2 Known Information About the Ball's Position

The development of a state prediction mechanism for the ball requires a model for representing both its position and motion. As previously mentioned, spherical coordinates are well suited to specifying the ball's position within its spherical cap. Angular velocities are used to represent motion.

Current information about the ball's state at time t, is available from the robot's sensors which yield

$$currentBallPosition = (r_\theta, \theta, \varphi) \tag{6.2}$$

where r_θ is the known radius of the spherical bowl, θ is the zenith angle and φ the azimuth angle.

At any given time, the ball is at a particular height on a plane parallel to the XY-plane. This horizontal plane has a circular intersection with the sphere. If the θ coordinate of the ball were fixed and only φ varied, the ball would travel along such a circle, the radius of which is given by:

$$r_\varphi = r_\theta \ \sin\theta \tag{6.3}$$

The following illustration uses such horizontal circles to represent the sphere.

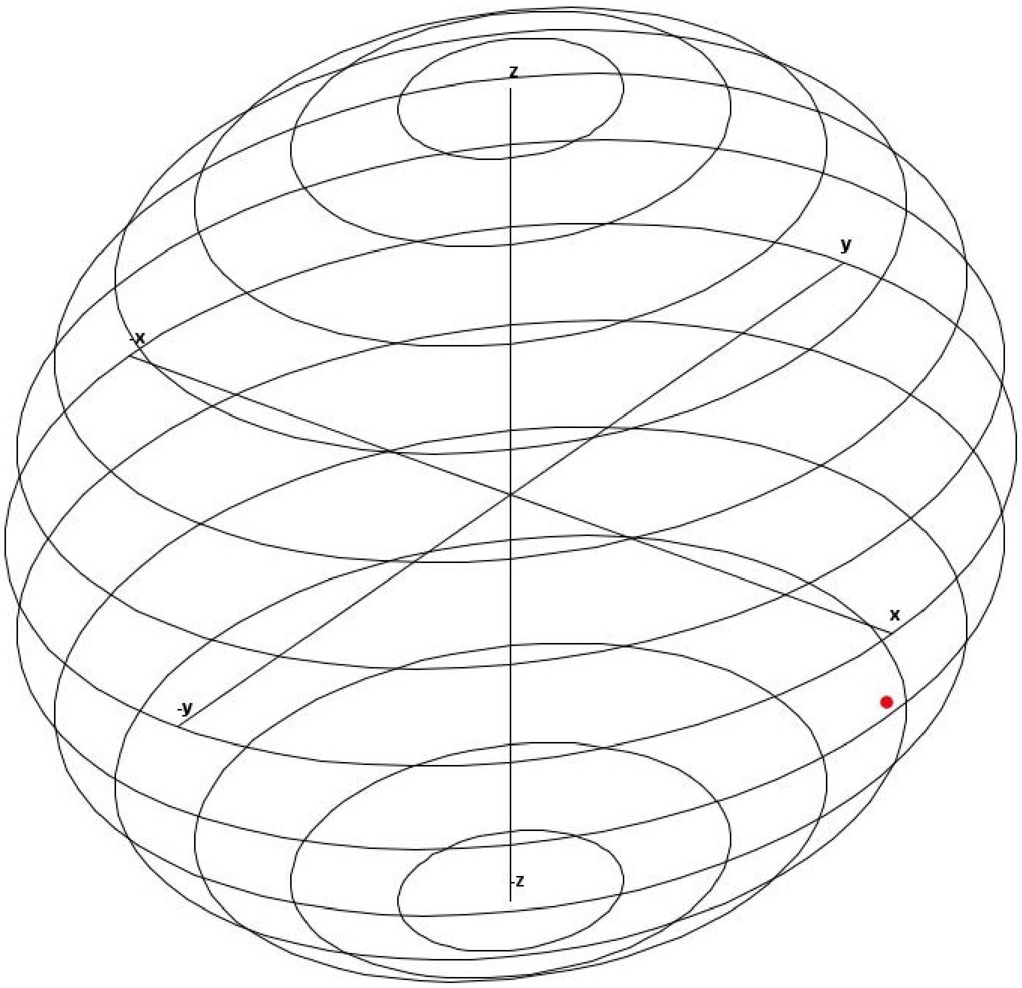


Figure 6.3: The ball moves within a spherical bowl, represented here using horizontal circles. The red dot indicates the position of the ball which can be expressed in terms of both Cartesian and Spherical coordinates.

As can be seen, the origin of the Cartesian coordinate system is at the centre of the sphere with unit vectors:

$$\hat{\mathbf{i}} = \begin{bmatrix}1\\0\\0\end{bmatrix}, \hat{\mathbf{j}} = \begin{bmatrix}0\\1\\0\end{bmatrix}, \hat{\mathbf{k}} = \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{6.4}$$

Finally, the ball's current angular speeds are known as $\omega_\theta(t)$ and $\omega_\varphi(t)$. These are stored between each time step, but could also be computed as the positional differences divided by the time step length TSL.

The task is now to predict the ball's spherical coordinates at time t+1 using the available information. To do so, consider the forces that act on the ball when a given set of controls is executed.

6.3.3 Acceleration due to Robot Translation

The first force to be modelled is that caused by acceleration or deceleration of the robot. Acceleration can be felt as the force pressing a driver into their seat when making a fast start. The seatbelt is needed to quell the opposite force caused by sudden breaking.

Acceleration is the change in speed over time, so we first need to ascertain the robot's speeds.

$$\vec{v}_{Robot}(t) = \frac{1}{TSL}(\vec{P}(t) - \vec{P}(t-1)) \qquad \vec{v}_{Robot}(t+1) = \frac{1}{TSL}(\vec{P}(t+1) - \vec{P}(t)) \tag{6.5}$$

Using these, the acceleration can be calculated as:

$$\vec{a}_{Robot} = \frac{1}{TSL}(\vec{v}_{Robot}(t+1) - \vec{v}_{Robot}(t)) \tag{6.6}$$

The ball's acceleration due to the robot's translation in (x, y) is shown here as a vector in 3D Cartesian space. Note that the force on the ball is equal and opposite to the direction of the robot's acceleration as indicated by negation.

$$\vec{a}_t = \begin{bmatrix} -\vec{a}_{Robot}.x \\ -\vec{a}_{Robot}.y \\ 0 \end{bmatrix} \tag{6.7}$$

6.3.4 Acceleration Due to Gravity

Gravitational acceleration is easily represented as a pull towards the negative Z-axis using Earth's well known constant $g \approx 9.81 \ m/s^2$.

$$\vec{a}_g = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{6.8}$$

6.3.5 Centrifugal Acceleration

Finally, when travelling horizontally along a circle in φ, the ball experiences a radial acceleration known as the centrifugal force. This fictitious force really expresses the absence of centripetal force, which always points towards the centre of rotation and is given by $F_c = \frac{mv^2}{r}$ where v is velocity and r is the radius of the circle. The centrifugal force is modelled as having the same magnitude but pointing in the opposite direction, i.e. outwards from the centre of rotation. To find its magnitude, the ball's (known) speed in φ, which is given in radians per second, first needs to be converted to a speed in meters per second using $v = \omega \; r$, where ω is angular velocity. We can now write:

$$F_c = \frac{mv^2}{r} = m \; r \; \omega^2 \tag{6.9}$$

Inserting our known angular velocity, $\omega_\varphi(t)$ and the radius r_φ yields the magnitude of our force.

$$|a_c| = m \; r_\varphi \; \omega_\varphi(t)^2 \tag{6.10}$$

To give direction to this force, consider how the cylindrical unit vectors are related to the Cartesian unit vectors.

$$\begin{bmatrix} \hat{\mathbf{r}} \\ \hat{\varphi} \\ \hat{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{i}} \\ \hat{\mathbf{j}} \\ \hat{\mathbf{k}} \end{bmatrix} \tag{6.11}$$

For a small enough period of time (in which the height z does not change), the ball can be seen as orbiting a cylinder. The outward direction of the centrifugal force is given by the radial unit vector $\hat{\mathbf{r}}$. Since this vector is of length 1 by definition, we can scale it to the desired magnitude by multiplication.

$$\vec{a}_c = |a_c| \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} \tag{6.12}$$

We now sum all force vectors to get a_{ball}, the total acceleration the ball is subjected to.

$$\vec{a}_{ball} = \vec{a}_t + \vec{a}_g + \vec{a}_c \tag{6.13}$$

6.3.6 Effective Acceleration Component

The ball cannot move freely in 3D Cartesian space however, so we are only interested in the component of this force that points along the surface of the bowl. At any given time, the surface of the bowl can be approximated as the sphere's tangent plane at the ball's location. The aim is now to project the Cartesian acceleration vector onto this plane to get the effective acceleration vector $\vec{a}_{eff}$. To do so we need the normal vector of the plane, which is conveniently given by the spherical unit vector $\hat{\mathbf{r}}$ as seen below.

$$\begin{bmatrix} \hat{\mathbf{r}} \\ \hat{\theta} \\ \hat{\varphi} \end{bmatrix} = \begin{bmatrix} \sin\theta\cos\varphi & \sin\theta\sin\varphi & \cos\theta \\ \cos\theta\cos\varphi & \cos\theta\sin\varphi & -\sin\theta \\ -\sin\varphi & \cos\varphi & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{i}} \\ \hat{\mathbf{j}} \\ \hat{\mathbf{k}} \end{bmatrix} \tag{6.14}$$

In other words, the plane's normal vector is:

$$\hat{\mathbf{n}} = \begin{bmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{bmatrix} \tag{6.15}$$

Using this normal vector, the acceleration orthogonal to the plane of motion can be expressed as the projection $\vec{a}_{orth} = (\hat{\mathbf{n}} \cdot \vec{a}_{ball})\hat{\mathbf{n}}$ and subtracting this orthogonal component from the original acceleration vector yields the effective acceleration in the ball's momentary plane of motion.

$$\vec{a}_{eff} = \vec{a}_{ball} - \vec{a}_{orth} \tag{6.16}$$

6.3.7 Linear Velocity to Angular Velocity

So far, we have been operating in 3D Cartesian coordinates. A translation to spherical coordinates is required in order to obtain the accelerations in terms of θ and φ. This is done by projecting the effective acceleration vector onto the spherical basis vectors. The dot product directly yields the length of these projections because, as above, one of the vectors has unit length.

$$a_\theta = \vec{a}_{eff} \cdot \hat{\theta} \quad m/s \qquad a_\varphi = \vec{a}_{eff} \cdot \hat{\varphi} \quad m/s \tag{6.17}$$

The unit of these accelerations is still meters per second and needs to be converted to angular velocities. In so doing, it is essential to bear in mind that the radii of the movements in θ and φ are different. They have already been defined as r_θ and r_φ respectively.

$$a_\theta = \frac{\vec{a}_{eff} \cdot \hat{\theta}}{r_\theta} \quad rad/s \qquad a_\varphi = \frac{\vec{a}_{eff} \cdot \hat{\varphi}}{r_\varphi} \quad rad/s \tag{6.18}$$

6.3.8 Damping and Friction

So far, we have looked at factors that contribute to the ball's acceleration, but, over time, the ball will also lose energy. The biggest factor to consider is friction, which is modelled here in a simplified manner by use of a damping factor. Friction has no effect when the ball is half way up the sphere on either side ($\theta = \frac{\pi}{2}$ or $\theta = \frac{3\pi}{2}$) because it is in free fall at these points. As the ball descends, friction increases until it reaches its maximum effect at the bottom of the bowl. More precisely, the effectiveness of damping due to friction varies according to negative $\cos\theta$.

The amount of friction would, in reality, depend on a number of factors such as the material of the ball and the bowl. Instead of defining these, a damping factor is used by which speeds are multiplied. Let this factor be *dampingPercentagePerMs* and define the percentage of the angular velocity that is lost due to friction every millisecond. The effective damping factor, considering the ball's height, is then given as:

$$effectiveDampingPerMs = -\cos\theta \;\; dampingPercentagePerMs \tag{6.19}$$

This now specifies how much energy is lost, but it is more useful to define a damping factor that specifies how much energy remains.

$$dampingFactorPerMs = 1 - effectiveDampingPerMs = 1 + \cos\theta \;\; dampingPercentagePerMs \tag{6.20}$$

Finally, this factor is applied every millisecond, meaning that in our time step length (given in seconds) damping is applied 1000 TSL times, giving a total damping factor of:

$$dampingFactor = dampingFactorPerMs^{1000\,TSL} \tag{6.21}$$

Through experimentation it was found that reducing velocities by 0.1% every millisecond led to the best results, but the damping percentage can be adjusted as desired.

6.3.9 Predicted Theta Position

It is now possible to calculate the ball's new velocity as:

$$\omega_\theta(t+1) = dampingFactor \;\; (\omega_\theta(t) + a_\theta \;\; TSL) \tag{6.22}$$

Applying this speed for the length of the time step yields the next θ coordinate.

$$\theta_{Next} = \theta + (\omega_\theta(t+1) \;\; TSL) \tag{6.23}$$

6.3.10 Conservation of Angular Momentum

It may be tempting to calculate φ_{Next} analogously, but it is not quite that simple. Due to the change in θ, the radius of the circle along which the ball travels in φ has now changed. If, for instance, the new radius were smaller but the angular velocity remained the same, then the ball would travel a shorter distance in the same time. This goes against one of the most fundamental laws of physics: Energy cannot disappear. Energy conservation is achieved by compensating for the change in radius by multiplying the new angular momentum by the quotient of the old and new radii. We call this factor the angular momentum conservation factor (AMCF).

$$AMCF = \frac{r_\theta \ \sin\theta}{r_\theta \ \sin\theta_{Next}} \tag{6.24}$$

6.3.11 Predicted Phi Position

Using the $AMCF$, the new angular velocity in φ can be expressed.

$$\omega_\theta(t+1) = dampingFactor \ AMCF(\omega_\varphi(t) + a_\varphi \ TSL) \tag{6.25}$$

As the last step, compute the next φ coordinate.

$$\varphi_{Next} = \varphi + (\omega_\varphi(t+1) \ TSL) \tag{6.26}$$

The predicted next coordinates of the ball are:

$$predictedBallPosition = (r_\theta, \theta_{Next}, \varphi_{Next}) \tag{6.27}$$

6.3.12 Additional Factors

Two factors have yet to be accounted for in the presented physics model. The first is rotational inertia, which essentially describes how much effort is required to change the rotational velocity of an object. In the case of the ball rolling in the bowl, there is not much effort involved and inertia can be neglected without causing any significant changes to the ball's behaviour. All it would do is reduce rotational speeds by a small amount.

Secondly, the ball's radius has not been considered, i.e. it has been modelled as a particle. If the ball had a significant radius, its centre of mass would be closer to the centre point of the sphere and that would mean its radius of rotation would be shorter than r_θ. Again this would have very little impact – unless the ball were really quite large.

6.4 Urgency Heuristics

For assessing the urgency of the ball balancing task, a heuristic function of the robot's configuration is defined. The function can be called with any sample configuration to be evaluated. As mentioned earlier in this chapter, urgency will be based on the ball's total energy. The energy consists of potential and kinetic components, which can be calculated using the information stored in the sample configuration.

The potential energy is computed using the well known formula

$$E_{pot} = m \; g \; h \tag{6.28}$$

where m is mass, g the gravitational acceleration, and h the height of the ball above the bottom of the spherical bowl. Mass will be known, but can be disregarded as it is a constant factor and will cancel out at a later stage. With g also known, this leaves only the height to be computed from the spherical coordinates of the ball.

$$h = r_\theta + r_\theta \; \cos\theta \tag{6.29}$$

The radius r_θ is that of the sphere.

Kinetic energy is given by the expression:

$$E_{kin} = \frac{1}{2} m V^2 \tag{6.30}$$

where m is mass and V is velocity.

Seeing as the ball has a velocity in θ and in φ, there will be two contributions. Before plugging these values into the energy equation, the angular velocities (stored as part of the configuration) have to be converted to linear velocities.

$$v_\theta = r_\theta \; \omega_\theta \qquad v_\varphi = r_\varphi \; \omega_\varphi \tag{6.31}$$

The sum of both kinetic energies can then be written as follows (mass can be omitted, i.e. set to one, for the same reason as with potential energy):

$$E_{kin} = \frac{1}{2} m (v_\theta^2 + v_\varphi^2) \tag{6.32}$$

The total energy of the sampled configuration is simply the sum.

$$E_{sample} = E_{pot} + E_{kin} \tag{6.33}$$

In order to score this on the urgency scale, the ideal and worst cases need to be defined. The worst case is task failure, where the ball falls out of the bowl. To modulate the difficulty of the balance task, the hight of the ball's rim can be varied. It is specified in terms of an angular displacement in θ from the bottom of the bowl. For example, with a maximum angle of $\frac{\pi}{4}$, the ball is allowed to oscillate between $\pi - \frac{\pi}{4}$ and $\pi + \frac{\pi}{4}$. The maximum height above the bottom of the bowl is then

$$maxHeight = r_\theta + r_\theta \ \cos(\pi - maxAngle) \tag{6.34}$$

The energy of the ball at the rim of the bowl is its potential energy at the maximum height. Reaching or exceeding this energy means failing the task.

$$E_{worst} = m \ g \ maxHeight \tag{6.35}$$

The ideal case is easily defined as the ball resting at the bottom of the bowl where it has neither potential, nor kinetic energy.

$$E_{ideal} = 0 \tag{6.36}$$

Since the ball's energy may change fairly rapidly, these ideal and worst cases can be achieved within a foreseeable amount of time. If this were not the case, it would be preferable to set them to values that can in fact be attained within a few seconds. A simple policy may be to use a maximum amount by which the sample's potential can vary (e.g. $E_{sample} \pm 10\%$).

Whichever way the ideal and worst cases are set, the balance task's urgency at the sample configuration is computed using the formula discussed in the coordination chapter. At this stage the mass of the ball cancels out.

$$urgency = \frac{E_{sample} - E_{ideal}}{E_{worst} - E_{ideal}} \tag{6.37}$$

6.5 Experiments and Results

6.5.1 Physics Simulator

The correctness of the physics engine was continuously verified throughout its development and refinement using a specially created simulator, of which the following is a screenshot. On the left, the sphere is shown as a 3D projection. A top-down view of the ball can be seen at the bottom right and above it is the control panel of the simulator.

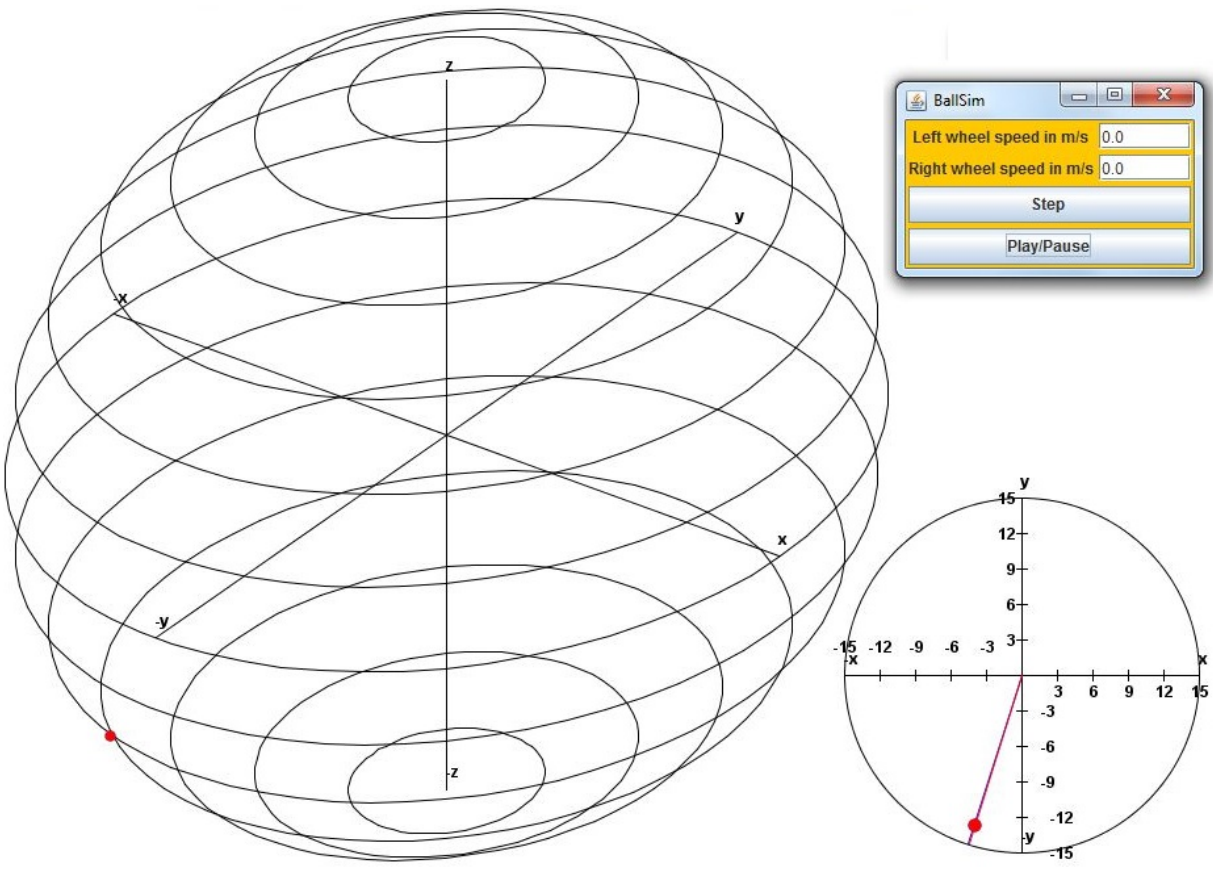


Figure 6.4: Screenshot of the ball-physics simulator showing a 3D projection of the sphere along with a top-down view of the ball and the application's control panel

Initially, the ball was given a non-zero height (e.g. $\theta = 100°$) with no initial speeds. When the simulation starts, the ball can be seen rolling down the side of the bowl as potential energy is converted to kinetic energy. The ball passes straight through the bottom of the bowl where it reaches maximum speed and then begins its ascent on the opposite side, demonstrating how kinetic energy is converted back to potential energy. Due to friction (modelled using the damping factor as explained earlier), the ball loses energy over time, meaning that it cannot regain its starting height on the opposite side of the bowl. As the simulation continues, the ball oscillates back and forth like a pendulum with gradually decreasing period, until it comes to rest at the bottom of the bowl with all energy depleted.

As a next step, the ball was given an initial speed in φ. For example, $v_\varphi = \pi$ means the ball goes half way around the bowl in one second or full circle in two seconds. Of course the ball also drops in θ as it circles in φ. The resulting behaviour can best be described as elliptical oscillation about the bottom of the sphere.

Finally, the simulator allows the user to set the robot's wheel speeds. This feature allows examination of the ball's response to acceleration and was used to verify that the robot can influence the energy of the ball. Once the ball was moving the way one may expect and the model was deemed accurate, a robot controller was implemented to reduce the energy of the ball.

6.5.2 Stationary Robot

To establish a benchmark for evaluating the experiments to come, first consider the simple case of a stationary robot. Like in the physics simulator, the ball is given an initial height. Its starting position can best be seen in the top-down view of the sphere. Here, a ball with coordinates $(r,\theta,\varphi) = (15cm, 115°, 0°)$ is displayed.

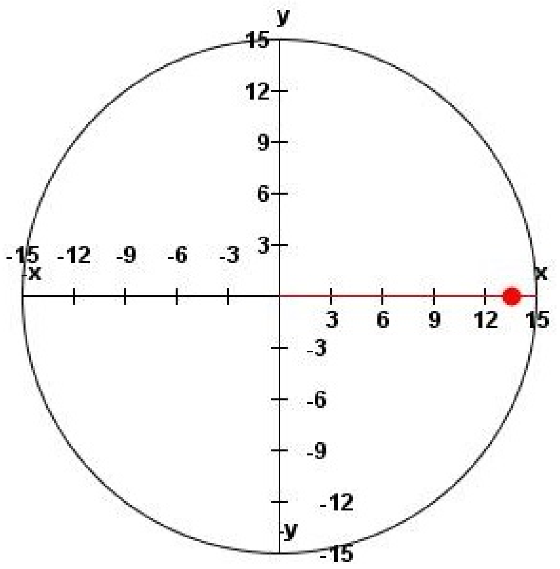


Figure 6.5: Top-down view of a ball with an initial displacement from the bottom of the bowl

Without intervention by the robot, the ball's energy graph looks as follows. The vertical axis shows the ball's total energy (sum of potential and kinetic energies) in joules and the horizontal axis shows time in terms of the number of discrete time steps.

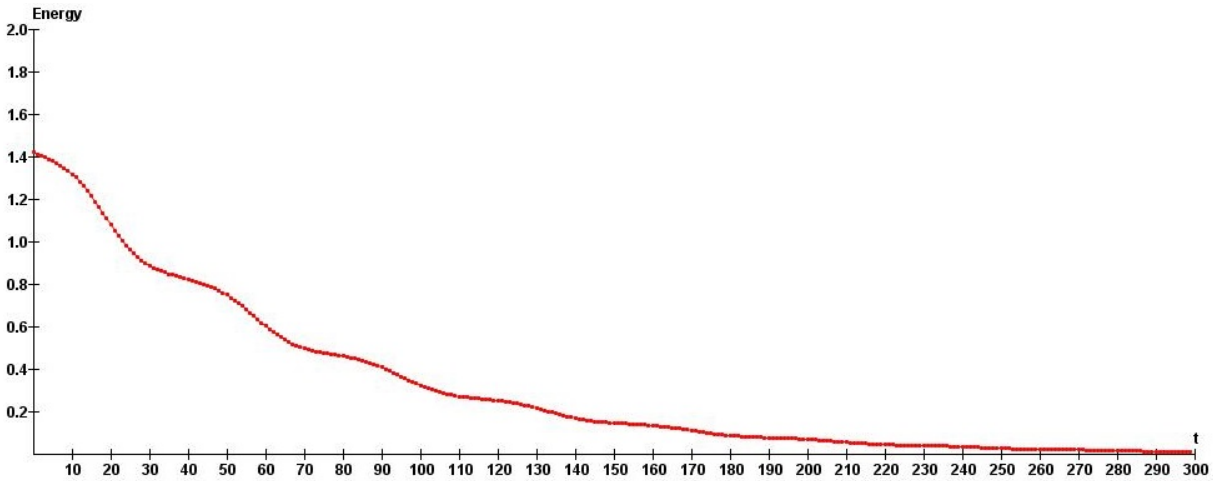


Figure 6.6: The ball's initial potential energy diminishes over time without robot intervention. Positional variations in friction explain the small wave-like ripples in the energy graph (see Damping and Friction).

6.5.3 Straight Line Acceleration

We now allow the robot's controller to intervene and observe how effectively it can reduce the ball's energy by accelerating. It will do so by driving forwards (its heading is along the positive x-axis) in a straight line, i.e. in line with the ball's motion, because this way its acceleration has maximum effect on the ball. The robot could choose to turn as well, but acceleration orthogonal to the ball's direction of motion has no damping effect on its energy and is therefore of no benefit.

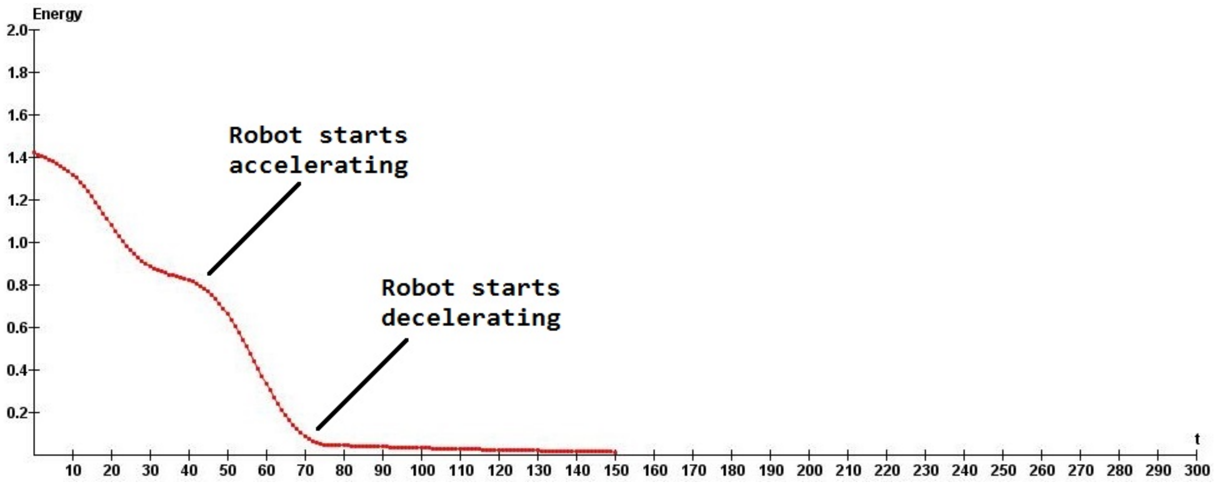


Figure 6.7: Graph showing a ball's potential energy being actively depleted by accelerating the robot in a straight line to counteract the forces moving the ball once released from its initial height ($\theta = 115°$)

The graph shows that the robot is able to bring the ball to a standstill in 150 time steps, which is only half of the time it takes to come to rest naturally. This is despite the fact that the robot has to wait for 40 time steps before it can start accelerating because it takes that long for the ball to swing to the opposite side of the bowl. As can be seen from figure 6.8, the ball does not regain its starting height due to losing some of its energy to friction while rolling from one side of the bowl to the other.

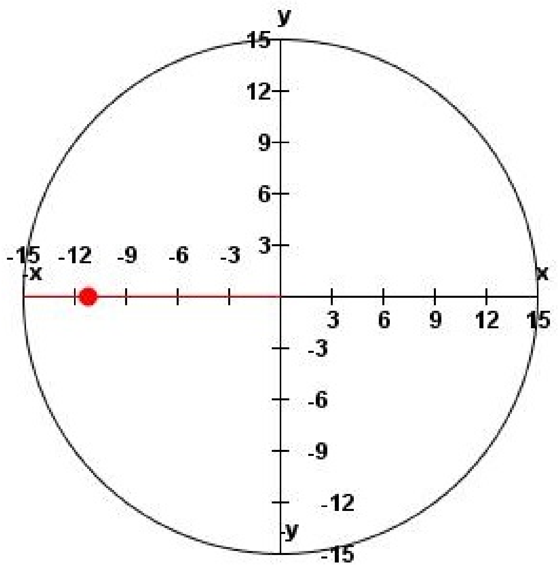


Figure 6.8: Top-down view of the ball having reached the opposite side of the bowl with a maximum displacement of just under 12 cm (down from 13.5 cm due to losing some energy to friction)

Accelerating before that time would increase the energy of the ball because driving forwards accelerates the ball backwards (in direction of the negative x-axis), exacerbating the problem. If the robot were allowed to drive backwards, which it is not, it could already intervene by doing so. This experiment demonstrates that the AI is able to discern that not moving at all can sometimes be the best policy. Since the robot does not move until t=40, the energy graph exactly matches the previous one up to this point.

Between t=40 and t=70, the ball rolls down the slope of the bowl from the direction of the negative x-axis towards the bottom of the bowl. In this time the robot accelerates as much as it physically can to maximally decrease energy, which is evident in the steep drop in the graph.

At t=70 the ball reaches the bottom of the bowl with only very little energy remaining. Nevertheless, this residual energy carries the ball through the centre of the bowl. In response, the robot then slowly decelerates to help roll the ball back into the centre. The deceleration phase takes fairly long as it needs to be done slowly. An abrupt decrease in speed would give the ball enough energy to roll back up the slope it just rolled down.

6.5.4 Acceleration Along a Curve

To further increase the complexity of the problem, let the ball start at an angle to the heading of the vehicle. We set up the experiment with the same starting height, but set $\varphi = +45°$ as shown.

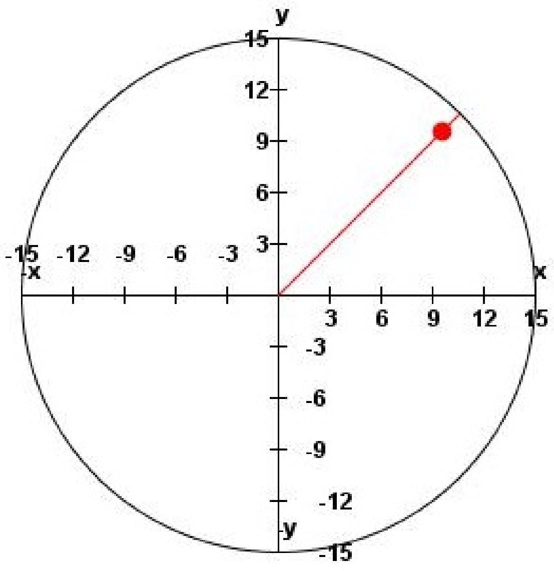


Figure 6.9: Initial ball displacement to the left of the robot's heading ($\theta = 115°, \varphi = 45°$)

As before, the robot is facing in the direction of the positive x-axis. Since the vehicle is not modelled as a point, it cannot immediately drive towards the ball. Using the arcs it is restricted to generate, the most effective path is a curve of gradually increasing curvature as displayed below.

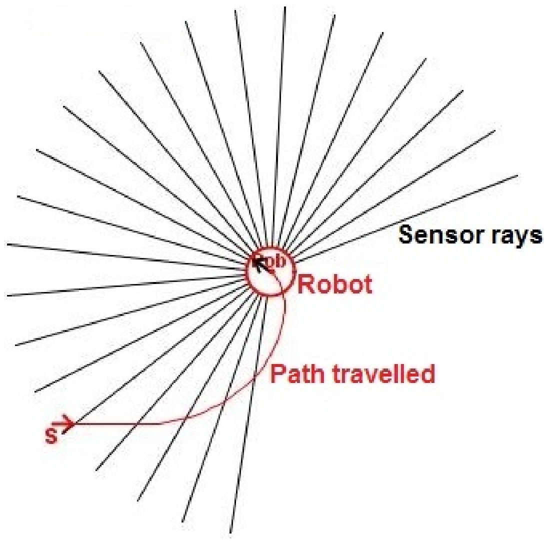


Figure 6.10: Left-curving path generated in workspace to becalm a ball released from an initial position of ($\theta = 115°, \varphi = 45°$)

Despite not being well aligned to solve this problem, the experiment shows success at driving into the ball to reduce energy as much as possible. Again this is only possible past t=40, but after that the energy can be seen to rapidly decline as the curved path is generated.

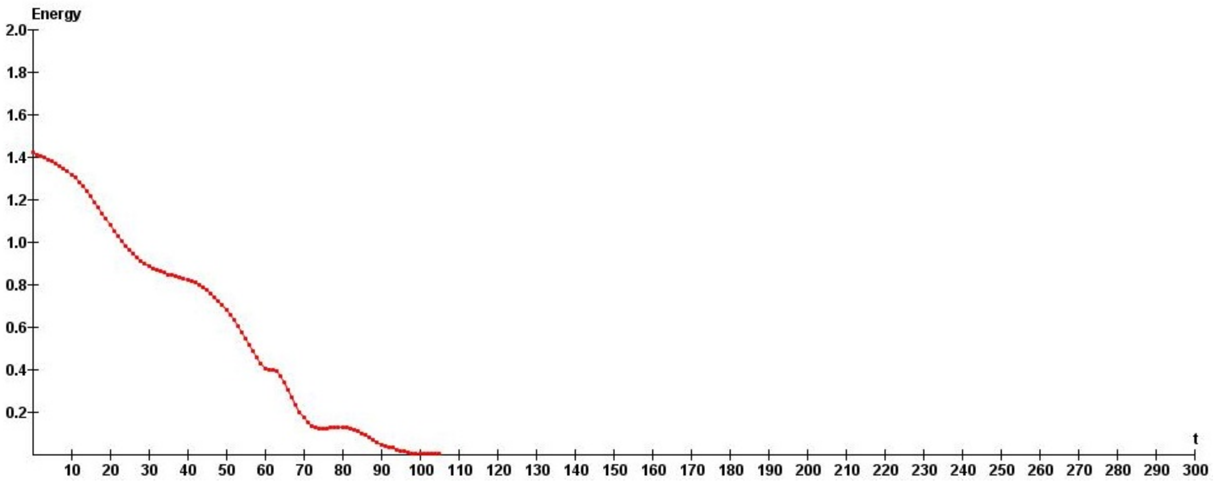


Figure 6.11: Graph showing a ball's potential energy being actively depleted by accelerating the robot along a curved path to counteract the forces moving the ball once released from its initial coordinates $(r, \theta, \varphi) = (15cm, 115°, 45°)$

The same experiment is repeated with the ball's initial position mirrored on the x-axis. This is to confirm that the resulting path curves in the opposite direction, as one would expect.

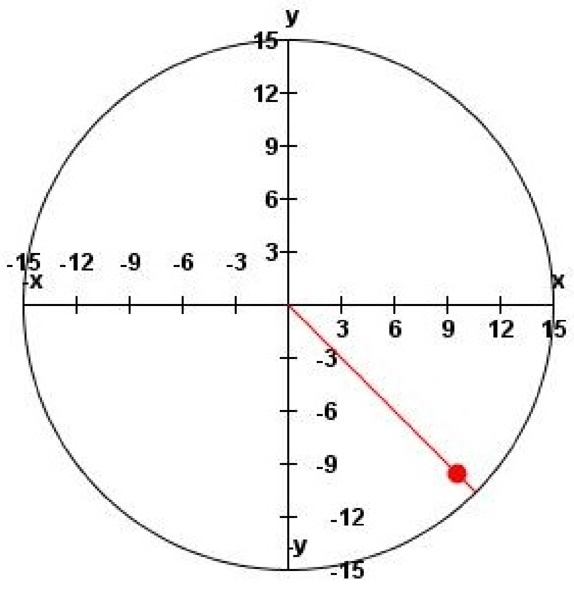


Figure 6.12: Initial ball displacement to the right of the robot's heading $(\theta = 115°, \varphi = -45°)$

Indeed, the robot produces a path of the exact same shape except that it is mirrored on the x-axis. This means the robot is able to orient itself in the direction the ball oscillates, where its acceleration has maximum effect.

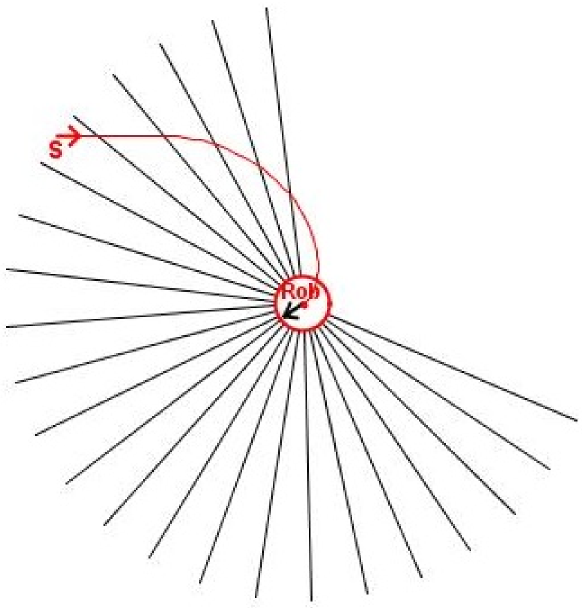


Figure 6.13: Right-curving path generated in workspace to becalm a ball released from an initial position of ($\theta = 115°, \varphi = -45°$)

The energy profile is identical to the one previously shown – as indeed it should be.

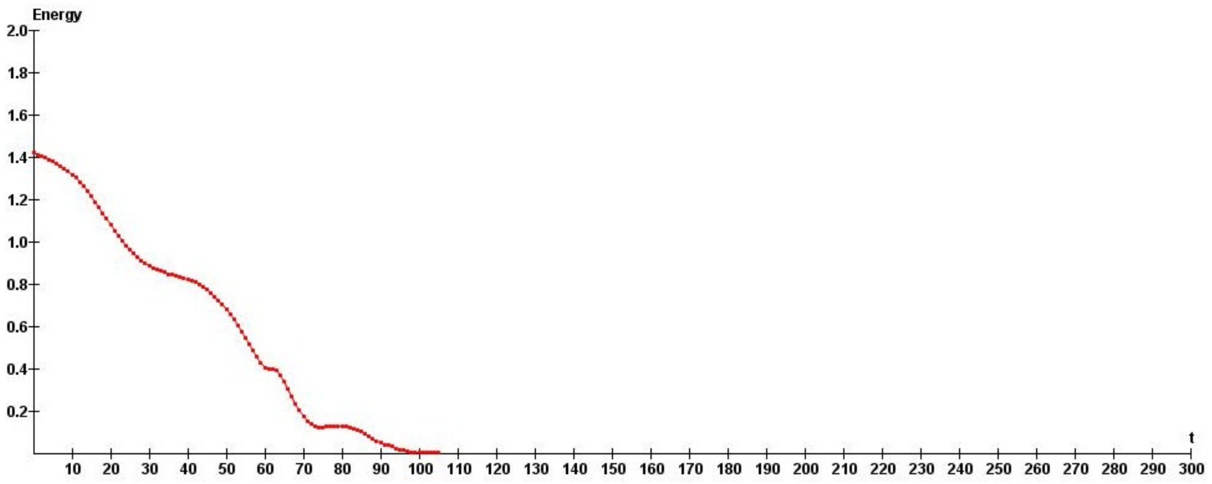


Figure 6.14: Graph showing a ball's potential energy being actively depleted by accelerating the robot along a curved path to counteract the forces moving the ball once released from its initial coordinates $(r, \theta, \varphi) = (15cm, 115°, -45°)$

Interestingly, the robot can stop the ball even more quickly when it starts at an angle in φ, despite this being a more difficult problem. The reason behind this is the short-sightedness of the one-step planner being used. Guided only by the desire to reduce the ball's energy as much as possible in the very next time step, the robot greedily accelerates in the straight line case. This strategy almost completely stops the ball by t = 70, which is indeed faster than the 100 time steps needed in the case where the ball starts at an angle. The AI has not, however, considered that a deceleration will be required if the ball cannot be stopped before it passes through the lowest point of the bowl. By the time this happens, it is already too late to correct the error. The robot is forced to decelerate slowly so as not to excite the ball. When driving in an arc, it accelerates less quickly which means it takes 100 steps to stop the ball, but it need not spend as long decelerating.

Finally, it should be noted that the robot will not necessarily stop when the ball reaches the bottom. Its best course of action is to maintain whichever speed it is at when the ball reaches zero energy because any change in speed will cause an acceleration that will reintroduce energy into the system.

6.6 Summary and Evaluation

This chapter documents the development of a complex physics model for simulating ball motion in a spherical bowl under the influence of translational, gravitational and centrifugal acceleration. State prediction also takes friction and the conservation of angular momentum into account.

An urgency heuristic is defined in terms of the potential and kinetic energies of the ball. Together, these two mechanisms provide the functionality required by the coordination mechanism and show that the first and second hypotheses are not unreasonable.

Experiments demonstrate a robot controller successfully bringing the ball to a stop from a state of unrest by reducing its energy. The results also indicate a disadvantage with single-step planners, which is of interest in the context of the fourth hypothesis. Since the short-sightedness does not lead to failure, but only a non-ideal solution being selected, this is not conclusive evidence that task coordination cannot work with a single-step planner. As long as it is simply a trade-off between computational complexity and accuracy, this is a compromise that likely needs to be made to allow for real-time planning.

The solution to the task presented here is independent of the homeostatic mechanism. It may be a useful addition to other systems and could also be used in conjunction with different task coordination mechanisms, including a multi-step planner. The task fulfils all the requirements laid out and is deemed suitable for coordination purposes.

Chapter Seven

Simulations and Results

Equipped with solutions to the developed demonstrator tasks, this chapter proceeds to test the proposed task coordination system. The first section describes the robot simulator that was developed for this purpose. Next, the design of the AI control system used in all the following tests, as well as the ones already presented in the task chapters, is explained. The bulk of the chapter is made up of the experiments themselves. Results are obtained for coordinating each pair of tasks, as well as all of them simultaneously. Findings are analysed and interpreted in a final section.

7.1 Robot Simulator

Before discussing the experiments conducted in the evaluation of the task coordination system, this section introduces the simulator with which results were obtained.

7.1.1 Available Simulators

Initially, Webots [27] was trialled as a testbed for the developed theory. At the time, this was a commercial robot simulator sold by Cyberbotics. One of the main difficulties encountered with Webots had to do with its physics engine, a fork of the Open Dynamics Engine (ODE) [104]. While very popular, the engine did have a number of known issues [33]. Approximations were used to make up for a lack of computational efficiency, leading to a loss in precision. This adversely affected the ball balancing task, as the forces acting on the ball were often too insignificant to register and when they did, the ball could suddenly lurch forwards. Meanwhile, Webots has become an open source project and may have improved but the change came too late for this PhD.

The Robot Operating System (ROS) [113] could have been a viable alternative, had it not been built on the same physics engine as Webots and inherited the problems already observed. Since ROS is an open source project, it may have been possible to develop a custom physics plug-in or patch the issue. However, such a fix would have had to be actively maintained to ensure continued compatibility with the simulator. Backward compatibility may have been dropped with any update, leading to a loss of functionality.

7.1.2 Design of a Custom Simulator

To ensure all required features could be provided and ball physics calculated with the necessary accuracy, an own simulator was built in Java. This endeavour did not involve as much additional work as one may expect. Task state prediction mechanisms had to be developed for all demonstrator tasks and the robot simulator is essentially an amalgamation of these. So as not to overcomplicate matters, it is designed for a simple 2D environment and, for now, only supports differential drive robots.

The following diagram displays the simulator's main components and indicates how they interact. A brief explanation of each component follows.

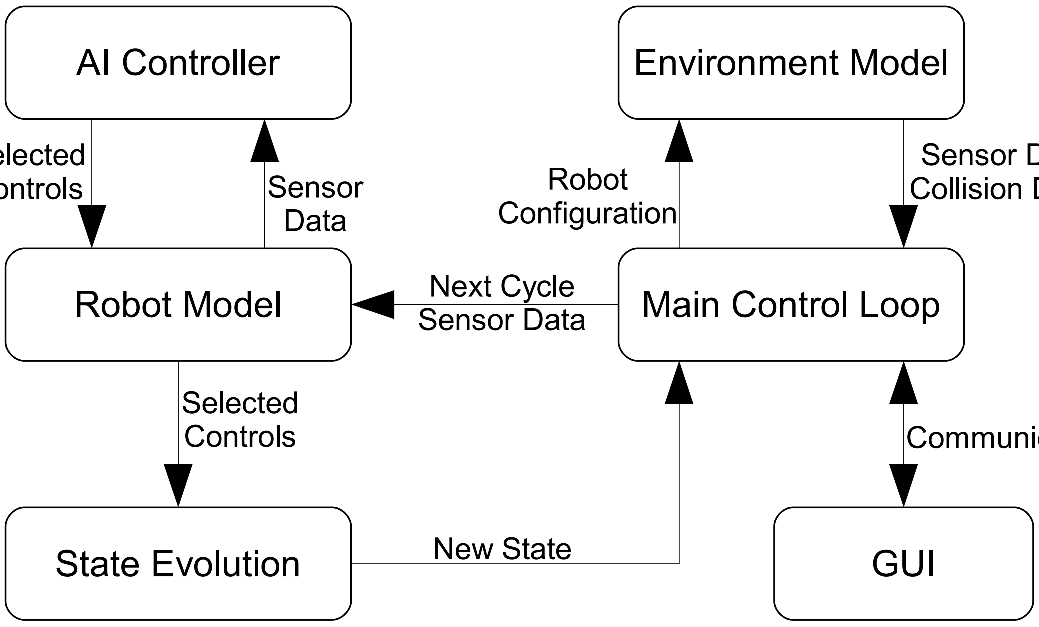


Figure 7.1: Design of the robot simulator showing all major components and their interactions

7.1.3 Graphical User Interface

The simulator's graphical user interface consists of two windows, a control panel and a view of the simulated environment.

Robots and obstacles are viewed top down as seen in the following screenshot of the main application. The buttons at the top can be used to play, pause or step through the simulation.

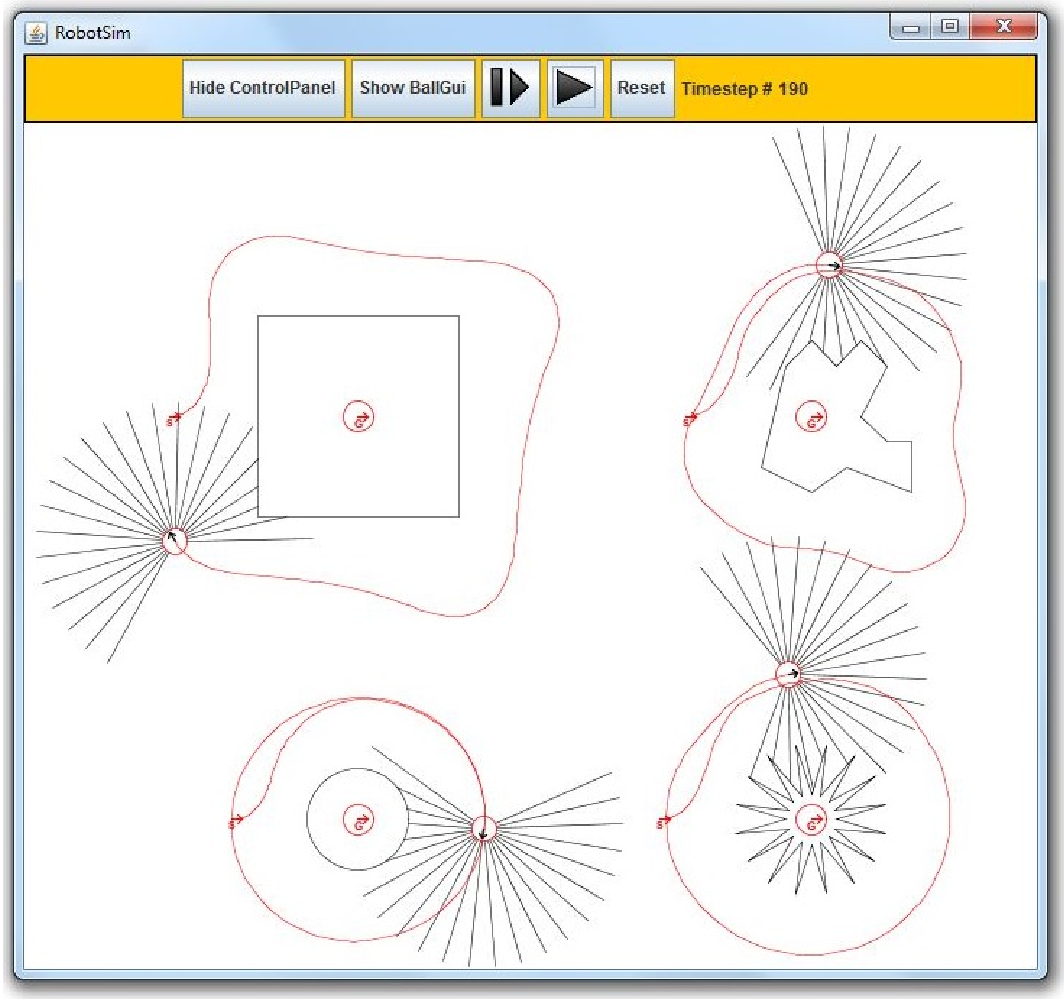
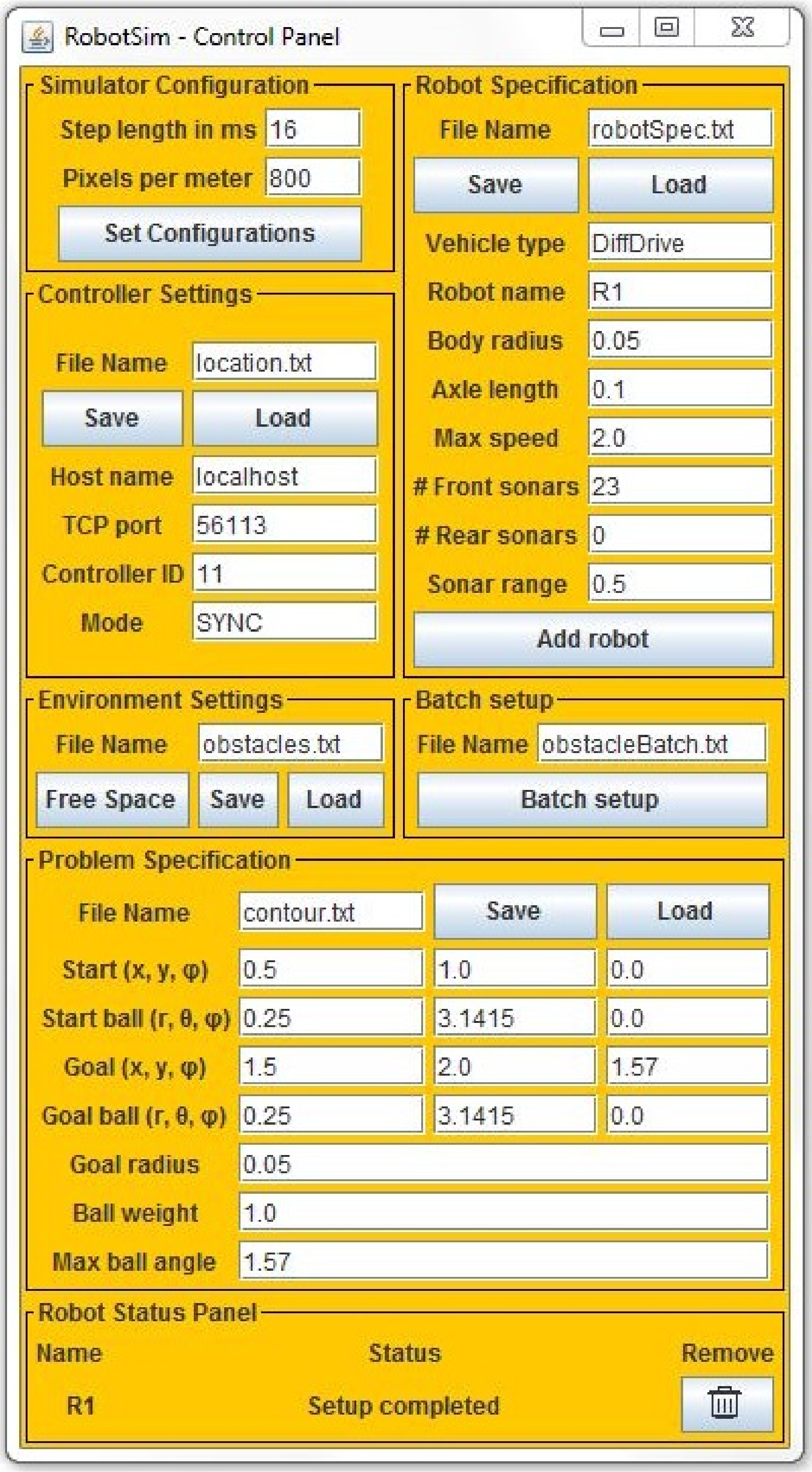


Figure 7.2: User interface of the robot simulator used in the evaluation of the proposed homeostatic task coordination mechanism

The control panel is used to add robots to the simulation. A wide array of settings may be specified, including the robot's size, number of sensors, start and goal configurations etc. Different obstacle courses can be loaded from file and a variety of controllers can be selected.


RobotSim - Control Panel

Simulator Configuration

Step length in ms	16
Pixels per meter	800

Set Configurations

Controller Settings

File Name	location.txt
Host name	localhost
TCP port	56113
Controller ID	11
Mode	SYNC

Save Load

Robot Specification

File Name	robotSpec.txt
Vehicle type	DiffDrive
Robot name	R1
Body radius	0.05
Axle length	0.1
Max speed	2.0
# Front sonars	23
# Rear sonars	0
Sonar range	0.5

Save Load Add robot

Environment Settings

File Name: obstacles.txt

Free Space Save Load

Batch setup

File Name: obstacleBatch.txt

Batch setup

Problem Specification

File Name	contour.txt	Save	Load
Start (x, y, φ)	0.5	1.0	0.0
Start ball (r, θ, φ)	0.25	3.1415	0.0
Goal (x, y, φ)	1.5	2.0	1.57
Goal ball (r, θ, φ)	0.25	3.1415	0.0
Goal radius	0.05		
Ball weight	1.0		
Max ball angle	1.57		

Robot Status Panel

Name	Status	Remove
R1	Setup completed	🗑

Figure 7.3: The control panel is used to add and remove robots to/from the simulation. It allows properties of the robot to be changed, different controllers to be selected and task parameters to be set.

7.1.4 Main Control Loop

Once the user has set up a problem by specifying its parameters in the control panel, the simulation can be started. Doing so spawns a thread to run the main control loop.

In each iteration of that loop, which marks the passing of one discrete time step, the system evaluates the sensors of all active robots. It does so with the help of the *Environment Model*, which, among other things, works out the distances that should be registered by a robot's sonar sensors. The recorded sensor data is sent to the *Robot Model* and passed on to the *AI Controller*, which has until the start of the following time step to decide the next action. While the next controls are being computed, the ones selected at the end of the previous step are executed. This means the controller is always acting on sensor information that was recorded in the previous time step. Not implementing this delay would be unrealistic as it would imply that controls are computed instantly. In reality, the robot is moving while it is processing. It cannot spend the whole time step computing and then teleport to the end of the transition it decided on.

For the response time to be good, the time step length must be as short as possible. Along with many other parameters it can be set using the simulator's control panel. The default duration is only 16 milliseconds to ensure the robot is responsive and the physics simulation accurate.

7.1.5 Robot Model

A separate model is created for each robot in the simulation and contains all the information pertaining to it. This includes fixed parameters such as the robot's size and number of sensors, as well as changing variables making up its configuration. The problem the robot is tasked to solve is represented in memory by its start and goal configurations.

In addition to storing data, the model, which can be thought of as a virtual robot, also communicates with its *AI controller*. It does so by sending it the raw sensor information recorded by its sensors. In return it receives the controls selected for execution in the next time step. Execution is simulated by passing the controls to the *State Evolution* component.

The only type of robot currently supported is a differential drive vehicle [35, p. 18]. Although in future the developed control system is to be tested using a variety of vehicle architectures, this type is well-suited for initial experimentation. Differential drive describes propulsion by two independent wheels which are positioned on either side of the chassis. The vehicle is controlled by changing the speed with which each wheel turns. When both wheels rotate at the same rate the robot travels straight ahead in the direction it is facing. Curves, specifically circular arcs, are generated by using unequal wheel speeds. The robot will turn in the direction of the slower wheel. Straight lines and arcs are thus its primitive motions. More complex paths, such as ellipsoidal ones, cannot be generated directly but are instead approximated using a sequence of primitive motions.

7.1.6 AI Controller

The *AI controller* is not part of the robot simulator, but a stand-alone component that communicates with it. This design decision was made in preparation for the transition to working with physical robots (see Future Work).

The controller is an implementation of the task coordination framework, or, more specifically, its homeostatic variant. The details of the proposed action selection mechanism for SMT can be found in Chapter 3 and will not be repeated here.

7.1.7 State Evolution

State evolution is the component in which most of the actual simulation takes place. It receives selected controls and works out how their execution will affect both the robot's configuration and the state of the external environment.

Fortunately, the state prediction mechanisms already developed for each task perform the same functionality and can be reused (see sections 4.3, 5.3 and 6.3). This allowed the robot simulator to be built with very little overhead.

When the new state has been determined, it is passed to the *Main Control Loop*, where the states of all robots are collected and used in the following controller cycle.

7.1.8 Environment Model

The simulated world is a 2D workspace with obstacles. Obstacle courses can be loaded from files containing a description of the locations and sizes of circular, polygonal and C-shaped obstacles.

When the *Main Control Loop* receives new robot state information, it uses the environment model to perform collision tests and to assign new sensor data to the robot model. If collisions or other indicators of failure are detected, the control loop is notified and an appropriate error message is shown in the graphical user interface. The user is likewise informed upon successful completion of all tasks.

7.2 Controller Implementation

To show that the developed theory has practical application, a controller needs to be able to select useful controls by minimising the provided heuristics. Individual tasks are solved by selecting the controls associated with least urgency. When multiple tasks are to be coordinated, their homeostatic mortality index is used instead.

For a differential drive robot, search space is two dimensional with only left and right wheel speeds needing to be found. Not yet suffering from the curse of dimensionality with two degrees of freedom, a sampling approach is tractable. For higher dimensions, a technique such as Benjamin's "Interval Programming" (IvP) [9] method may be more appropriate.

Sampling takes place in the first quadrant of control space, i.e. only positive values for the wheel speeds are considered. A differential drive robot can usually reverse by selecting negative speeds, but here we restrict the vehicle to forwards-only motion. This serves several purposes. Firstly, not all vehicles can reverse (e.g. aeroplanes) so a general solution should not rely on this feature. Secondly, slowing down to change direction and then accelerating again is typically inefficient and can result in motion that is not at all smooth. Finally, this restriction allows certain features of the system to be demonstrated. For example, the strain-based heuristic allows the goal location task to first move away from its destination before turning round to approach it.

Search is even further restricted to model (semi-)realistic acceleration and deceleration limits. Although the entire space can be sampled, most of those samples could not be executed by a real robot. Changing speeds takes time and the time steps are small, so each wheel can only be accelerated or decelerated by a small amount in each interval. This is accounted for by limiting search to a small square centred around the robot's current wheel speeds. The valid region is illustrated in figure 7.4 as a blue frame.

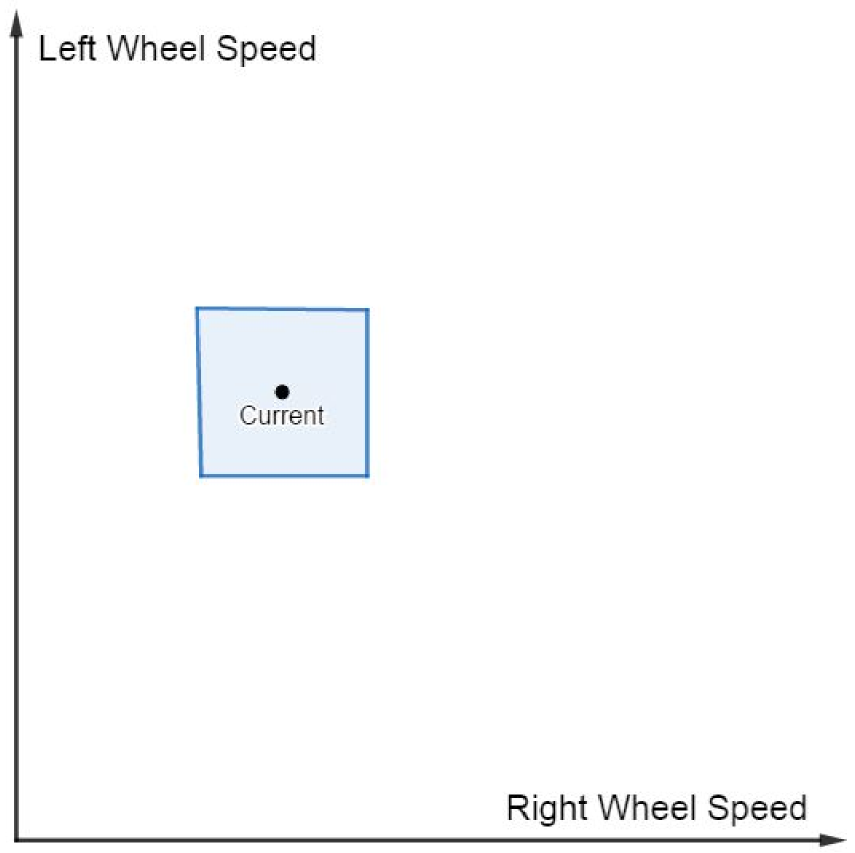


Figure 7.4: Control space for a differential drive robot within which a blue frame bounds the space of realisable controls to be considered in deciding the next action

It is assumed that the robot can instantly change speeds within this region, which of course is not entirely accurate. The simulation step lengths are however so small that the approximation seem reasonable. Depending on the power of the motors being simulated, the size of the speed frame can be varied.

Samples are distributed evenly throughout the marked area in a grid. A 16 by 16 grid would for instance generate 256 control samples. Each sample represents a candidate action which is costed in terms of urgency or using the HMI. The controls with least cost are selected for execution. For best results, the controller should evaluate as many samples as it can in its allotted time.

7.3 Experiments and Results

This section presents a series of experiments designed to empirically evaluate the developed homeostatic task coordination mechanism.

The first step towards ensuring the correctness of the system is to assert that it does not intervene unnecessarily. When only one task is being "coordinated", the ideal controls for that task should be executed without modification, i.e. as though the coordination mechanisms were not active at all and the task were in full control of the robot. This property was successfully confirmed by running the coordination mechanisms with each task to be coordinated. Consequently, the results do not deviate from those already documented in the task chapters above and will not be repeated here. The results presented in this chapter focus on cases where there are at least two tasks active at the same time such that conflict occurs and coordination is required.

7.3.1 Goal location with ball balancing

Simultaneous coordination experiments begin with the combination of goal seeking and ball balancing. In this constellation, one might expect the robot to rapidly accelerate towards its destination, while making only the required concessions in terms of speed and direction to keep the ball in its bowl. The reality does not live up to these expectations, which are borne of the human proclivity to inject a sense of time and urgency into most any task. As already discussed in Chapter 4, the goal location task is motivated primarily by the reduction of strain and cares less about the time of arrival. The little motivation for haste that was artificially instilled by use of the *discount factor* for past strain is quickly annulled by the balancing task's aversion to acceleration.

Without any notion of a deadline, coordination is simple. The robot can follow its desired path to the goal as long as it moves so slowly that the ball is not perturbed. From the controller's point of view, it is finding the perfect solution as both urgencies can be kept close to their ideals and the HMI is correspondingly low. So although the resulting behaviour may not correspond to initial expectations, the system is in fact doing exactly what it was constructed to do.

Any plans to hasten proceedings by increasing the goal location task's discount on past strain will backfire. While doing so does increase the gap between the urgency of fast and slow control samples, it also lowers overall urgency. In comparison, the balance task's urgency is now larger than it was before, which lends even more weight to its preference for low speeds. It is then unsurprising that the results obtained vary little from those presented in the task coordination chapter. An overview of generated paths is shown below. The biggest difference is that these similar paths take a lot longer to execute.

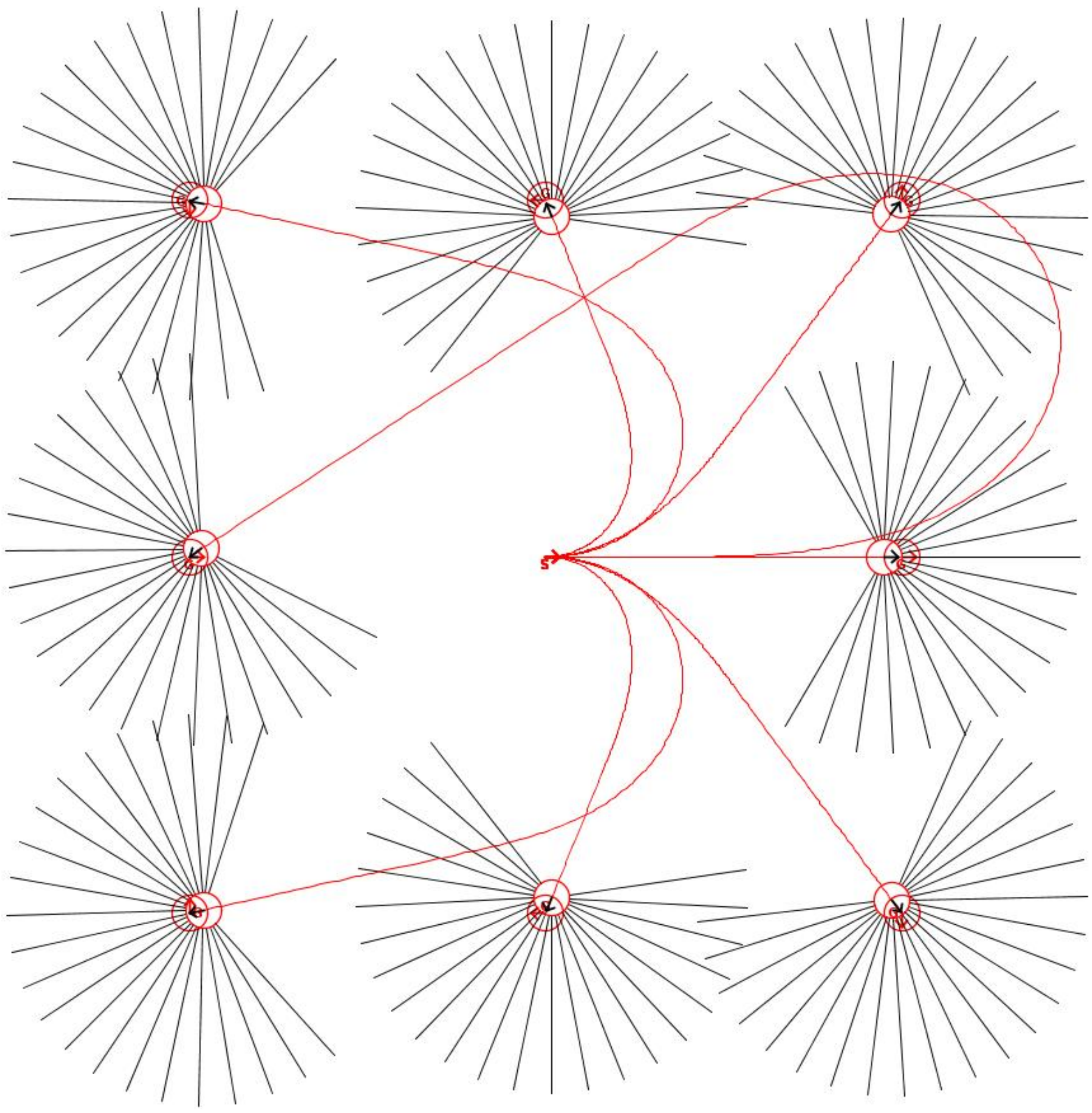


Figure 7.5: Collection of test cases demonstrating successful coordination of the goal location and ball balancing tasks

Since the difference between the goal location task run with or without ball balancing is almost imperceptible, consider the following direct comparison. The robot that has already attained the goal location on the left is unencumbered by the need to consider balance. Lagging behind is the slow moving robot that is coordinating both tasks. The paths generated are very similar, except for a slight difference in curvature. A less severe turn is beneficial to balance as the sideways acceleration component is smaller. This is where the homeostatic mechanism effects a compromise to balance the needs of turning towards the goal and keeping the ball's energy low.

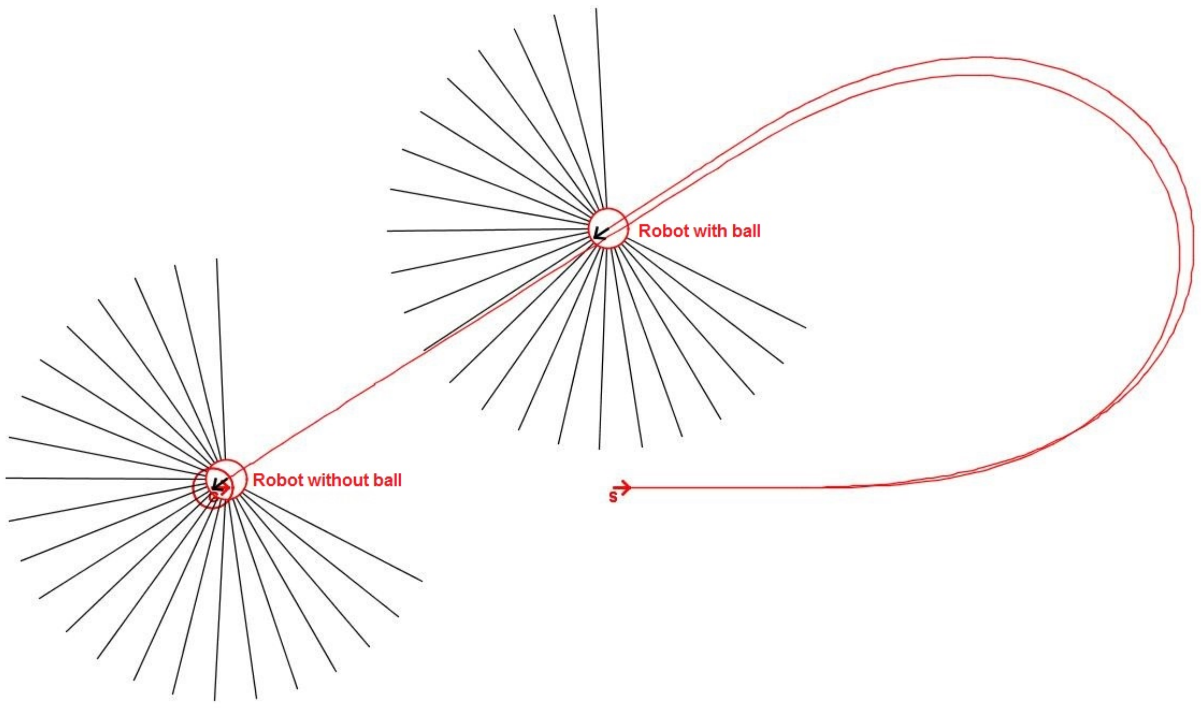


Figure 7.6: Comparison of paths generated with and without ball balancing: the robot not considering balance has already attained the goal on the left whereas the one balancing the ball is lagging behind

The ball's energy variation throughout this journey is displayed in the following graph.

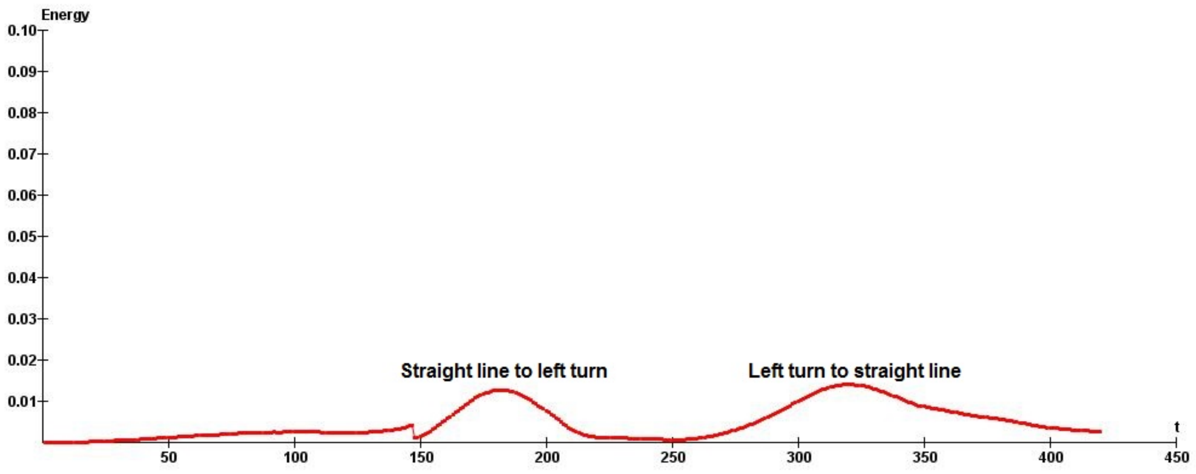


Figure 7.7: Energy graph for the ball-balancing robot in the above experiment

Just how slowly the robot moves can be seen from the horizontal axis, which indicates that the robot took about 420 steps to reach its goal. This compares to just 159 steps without the balancing task. Naturally the physical forces acting on the ball at these speeds are weak. Two humps in the curve mark changes in curvature. The first peak in energy coincides with the robot transitioning from an almost straight line to a left turn. Then, when the path straightens out again, the second spike occurs due to another change in the direction of acceleration. Even at the apex of these humps, energy barely reaches 0.02 joules. For reference, the task does not fail until the ball reaches around 2.45 joules at the rim of the bowl.

The low energy is explained partially by the absence of a motivation for taking on more risk, but is also evidence of the coordination mechanism fulfilling its purpose. All in all, the system behaves as it should even if not as one may have expected initially. Compromise is evident in the speeds being selected as well as in the curvature of the generated path. Both tasks are coordinated and brought to successful completion in all experiments conducted.

7.3.2 Obstacle navigation with ball balancing

More severe conflict arises in the combination of obstacle navigation and balancing. Obstacle avoidance and contour joining are grouped together due to their symbiotic relationship. So in fact there are three tasks being coordinated in the following experiments.

Despite the continued lack of an incentive to move at high velocities, tension between the tasks is increased due to the need of following more constrained paths around the obstacles. While making the required turns, the robot experiences radial accelerations which cause larger centrifugal forces to act on the ball. An example of this is given below.

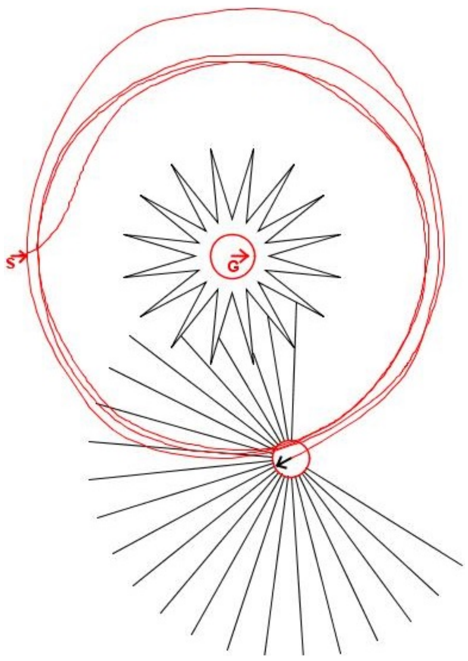


Figure 7.8: Following the contour of a star-shaped obstacle while balancing

At first the robot moves slowly, explaining the almost flat-lining energy graph shown below. After completing a few laps of the obstacle, the robot is in better alignment with the contour and can accelerate. An increase in speed may be motivated by obstacle avoidance. This is the only task that has a strong preference for high velocities as its proximity-based urgency falls off more rapidly when the robot takes large steps away from the obstacle. On top of that, the slight oscillations of the ball in the bowl can at some times favour a change in speed with both acceleration and deceleration being able to becalm the ball depending on its position.

Eventually however, the acceleration leads to increased agitation of the ball. Energy peaks at values in excess of 0.7 joules, which is significantly more than was reached in the previous experiments involving the goal location task. When energy levels rise too high, the controller decreases both curvature and speed. This behaviour repeats in cycles as shown by the energy curve between time steps 650 and 1050.

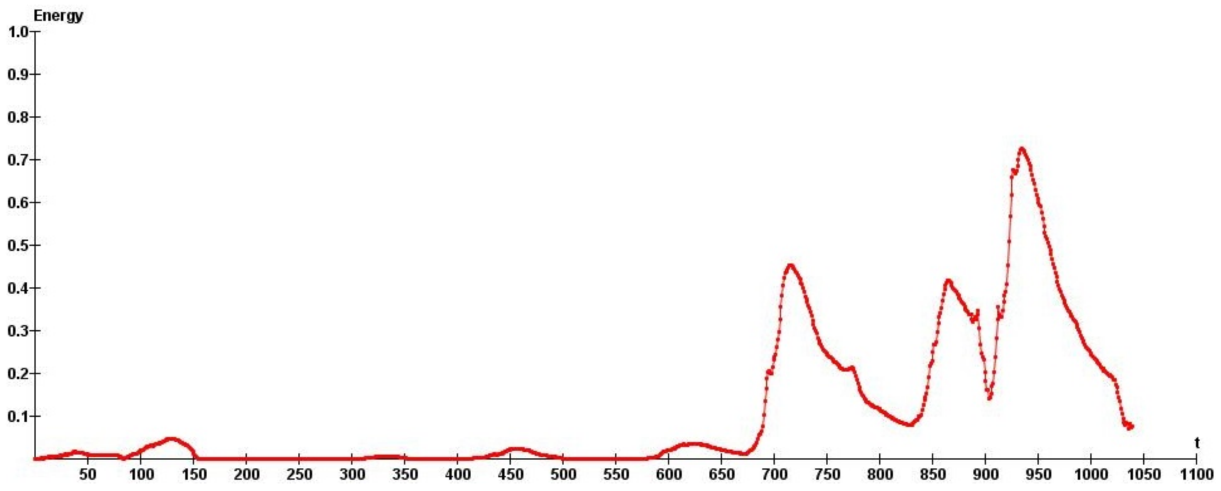


Figure 7.9: Graph of the ball's energy as the robot circumnavigates the star-shaped obstacle shown above

The robot is not always able to turn back onto the contour after a manoeuvre designed to rebalance the ball. In the next experiment, the controller deems it too dangerous to switch from a severe right turn to a left turn. As a result, it leaves the U-shaped obstacle it was following. Having reached free space, the obstacle navigation tasks no longer influence behaviour and the robot eventually comes to a stop with the ball at rest.

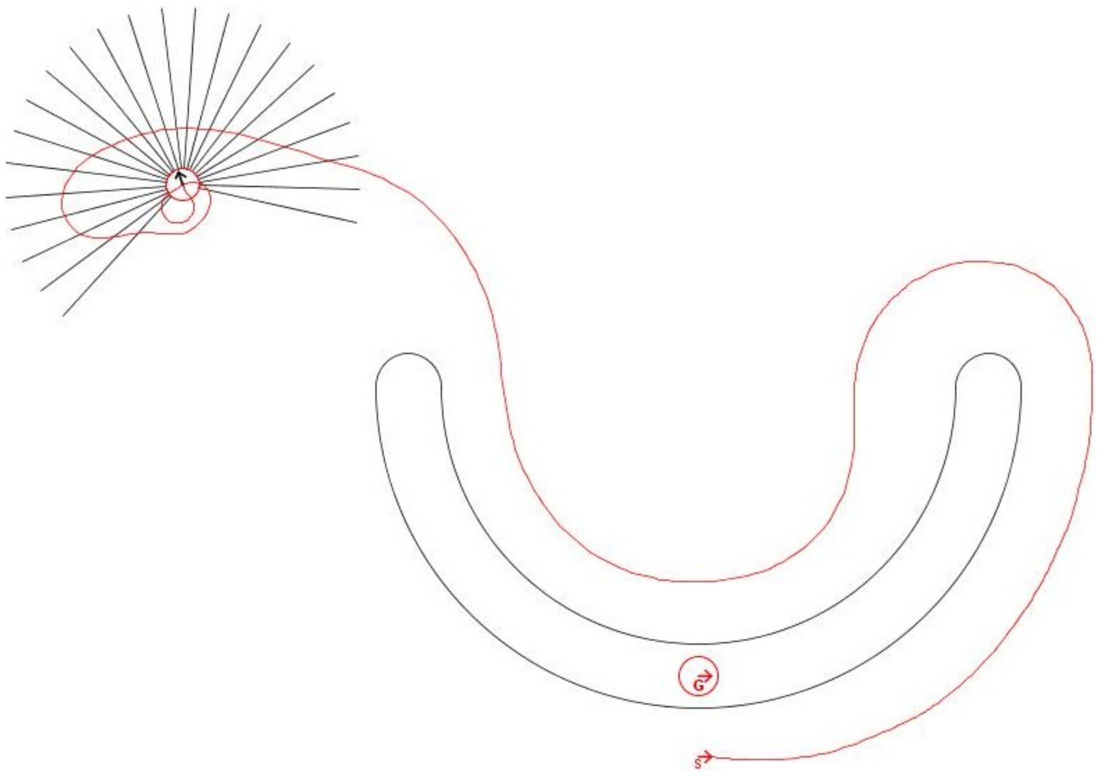


Figure 7.10: Following the contour of a U-shaped obstacle while balancing

The corresponding energy graph shows one initial peak caused by the robot accelerating away from a start position close to the obstacle edge. Energy rises again between time steps 340 and 400 as the robot goes from a left turn to a right turn inside the U-shape. The ball is now too perturbed for the vehicle to go back into a tight left turn. So instead of continuing along the obstacle edge, the controller opts for a more gradual turn that leads into free space.

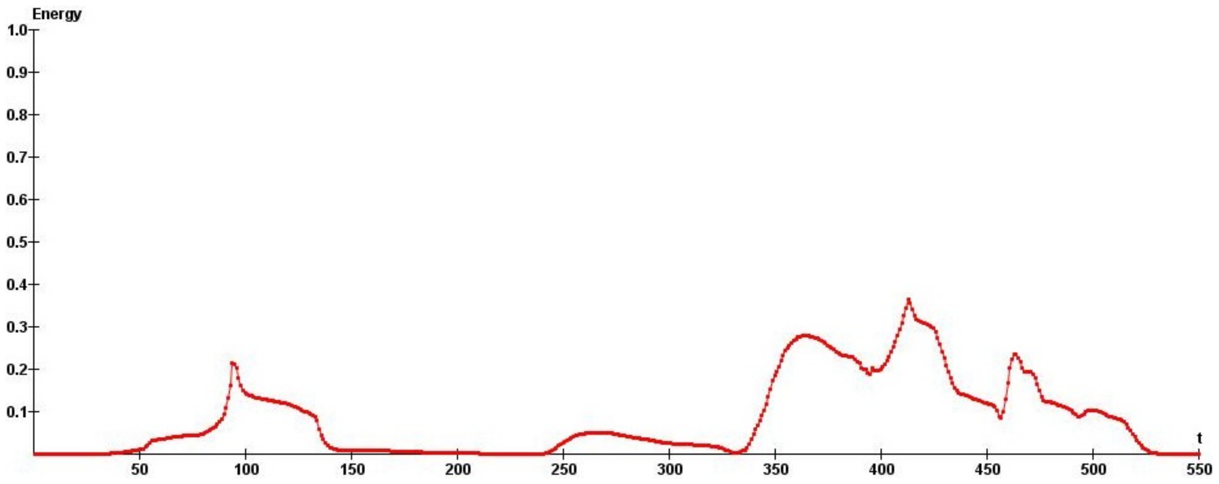


Figure 7.11: Graph of the ball's energy as the robot circumnavigates the U-shaped obstacle shown above

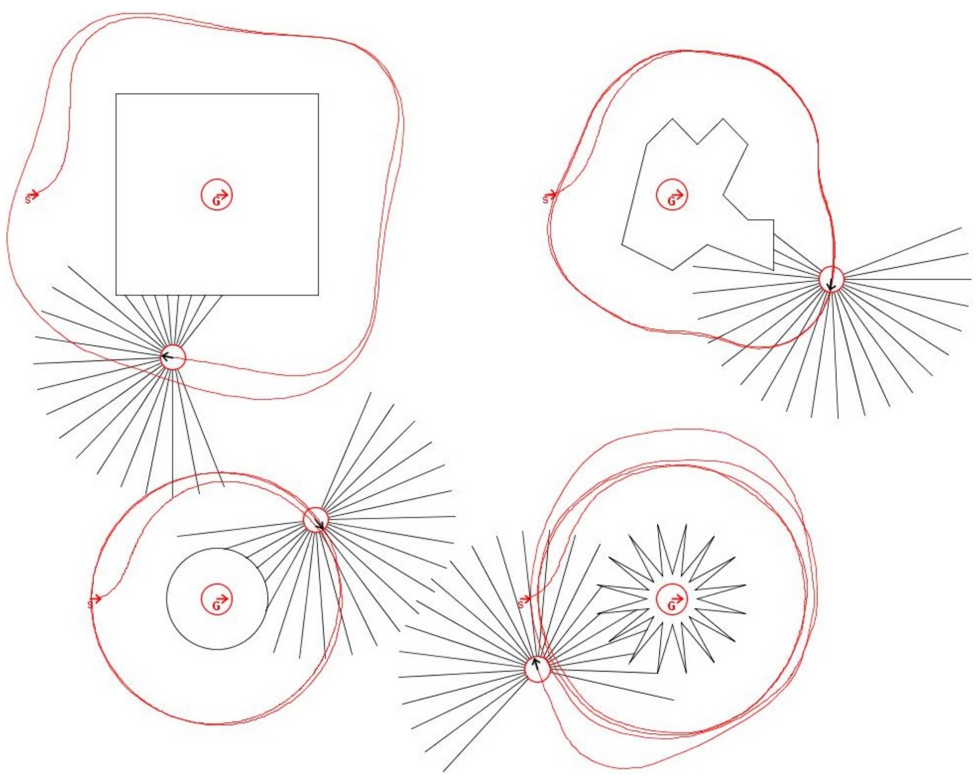


Figure 7.12: Collection of further test cases demonstrating successful coordination of the obstacle navigation and ball balancing tasks

A screenshot of further experiments is shown above. The paths taken around the edges of the polygonal shapes are similar to those produced by attending to contour joining in isolation. Most likely this is explained by there being less radial acceleration when following these straighter edges.

It is then surprising that the path around the circular obstacle is also smooth. The perfect circular shape is a possible explanation. While centrifugal forces will be at work, there is never any change in curvature. This is not the case with the star-shape, although the obstacle occlusion mechanism does its best to smooth out the rough edges. It seems that these small differences are nevertheless enough to have a visible impact. Overall behaviour is often not caused by one factor on its own but by a string of contributing events. A chain reaction may easily be caused. In this case a change in curvature may move the ball into a slightly different position, bringing the robot marginally closer to the obstacle edge. This results in an acceleration away from the obstacle to avoid collision which in turn agitates the ball etc.

Despite this volatility, experiments for the two tasks of obstacle navigation and ball balancing have shown that a compromise can still be found when motion is more constrained due to following a specific obstacle contour as opposed to merely aiming for a point in free space.

7.3.3 Goal location with obstacle navigation

The last pair of tasks to be tested is the goal location task in conjunction with obstacle navigation.

A collage of successful experiments demonstrates the ability to navigate differently shaped obstacles without clinging to their contour for longer than is conducive. This attests to the practicality of the obstacle engagement and disengagement conditions.

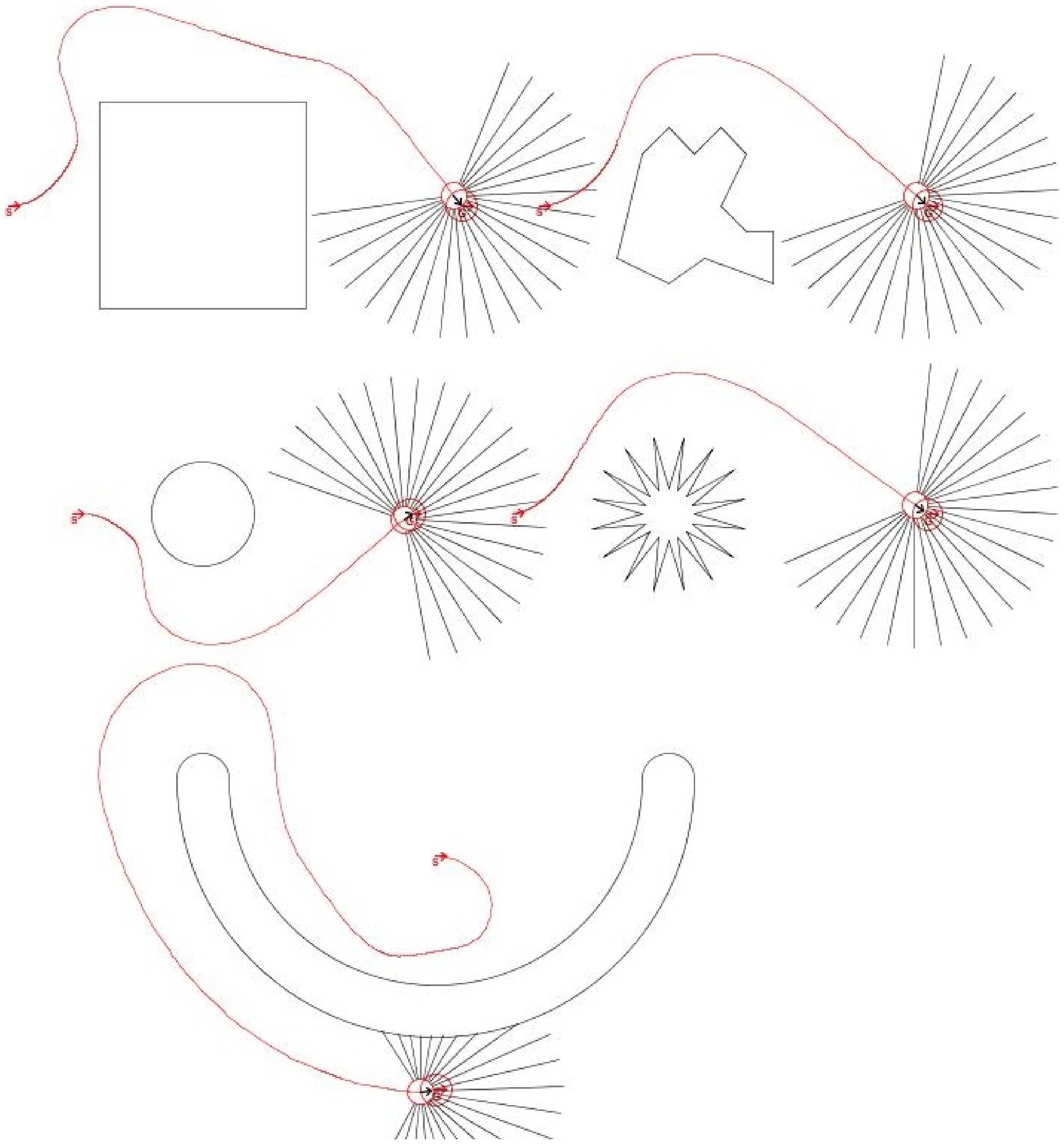


Figure 7.13: Collection of test cases demonstrating successful coordination of the goal location and obstacle navigation tasks

The recurring complaint is speed of execution. Not only is there a lack of a sufficiently powerful motivator for acceleration, the controller actually benefits from selecting low speeds. This is because small arcs allow for more fine-grained (and seemingly superior) compromises to be made. At the heights of contention between the two tasks, the vehicle slows to a crawl.

A further point of critique is that the robot ventures dangerously close to the edge of obstacles before decidedly turning away. While the coordination mechanism does eventually prioritise collision avoidance, the pull towards the goal location seems overly strong.

A possible culprit is the formula for the homeostatic mortality index. Despite reaching infinity within the $[0,1]$ interval over which it is defined, its slope ascends gradually to begin with. The system is therefore lenient with low to medium urgencies which may not be sufficiently penalised. This could lead to early signs of warning being blithely dismissed. Any adjustments to the formula would of course apply to all tasks equally, meaning the effect may cancel out.

Even if the symptoms could be slightly alleviated in this way, the problem unfortunately runs deeper. The following experiments substantiate the suspicion that a lack of foresight is to blame.

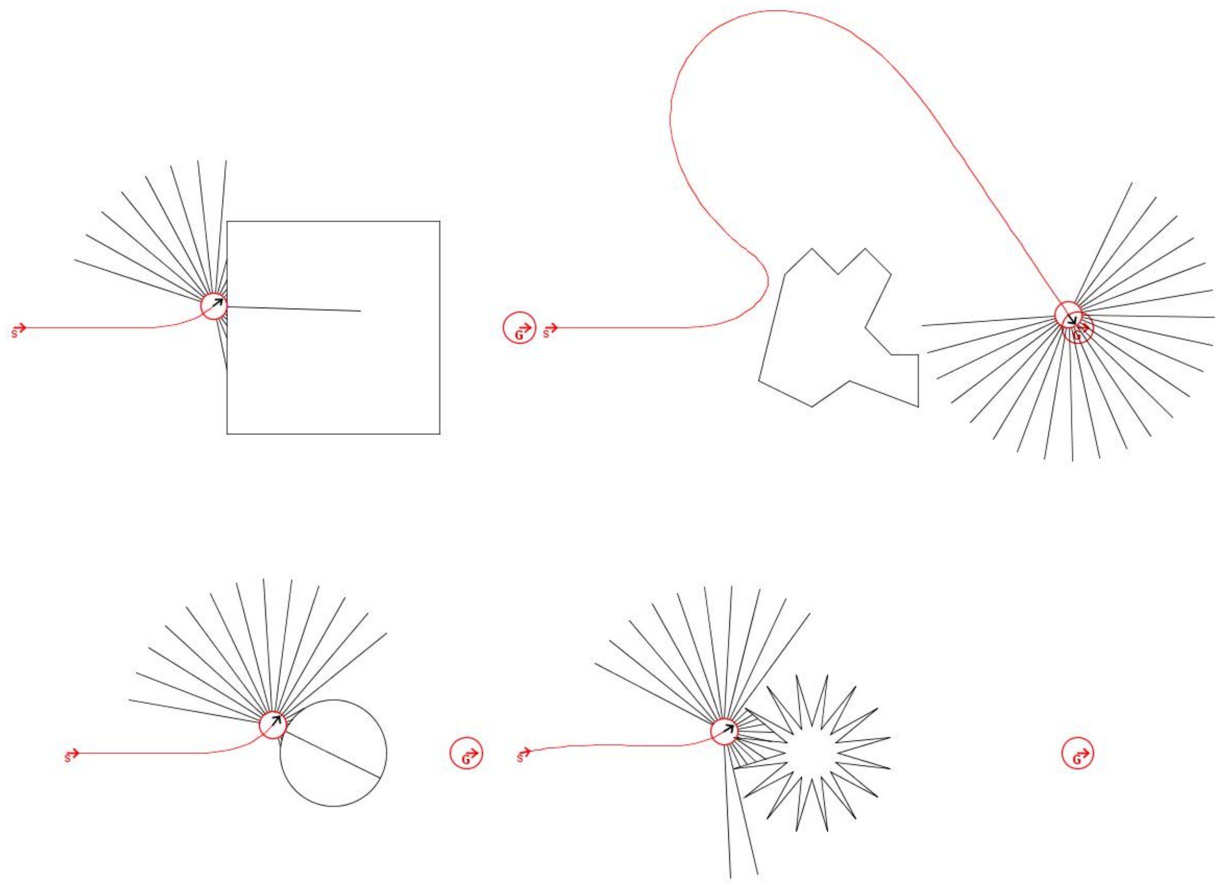


Figure 7.14: Experiments showing the impact of lacking foresight in the planner at high speeds

The only difference to the previously successful tests is that all start configurations were moved left so that the robots start in free space. Weak as their incentives may be, the robots do accelerate before reaching the obstacle. In three out of four cases the higher velocities lead to collision.

Slow growing or not, the HMI can and will reach infinity. So although the robot may pass by an obstacle closer than desired, the system should do everything in its power to prevent an actual collision.

As it turns out, the homeostatic mechanism *does* do everything it can, but breaking the acceleration limit is not within its power. That is not to say these problems are unsolvable. If the robot were to turn earlier, failure could easily be averted. Early turning is in fact precisely what the contour joining task was designed for, so what is stopping it from doing its job?

The reason is not to be found in the coordination system itself, which prioritises tasks in accordance with the urgencies they report. Instead, the problem is that urgency is under-reported due to lack of foresight. Planning only one step ahead, the controller fails to realise that turning must be initiated even earlier given the speed the vehicle is travelling at. In other words, the AI does not account for the time it takes to accelerate or decelerate. This leaves the robot in a situation in which it must either defy its physical limitations or brace for impact. As the former is not possible, the latter results.

A temporary fix may be to somehow inject the concept of speed management into the system either by modifying the contour joining task's urgency heuristic or by introducing a separate task. Ultimately though, the problem is a more general one and could reoccur in any context, i.e. with other tasks that requite some form of extended lookahead. In the long run, a more general solution, such as a multi-step planner, is required. The fourth hypothesis was successfully upheld for all individual tasks, but is not sustainable when multiple tasks are to be coordinated under realistic conditions.

7.3.4 Coordinating all tasks simultaneously

Finally, all four tasks are coordinated together.

With some apprehension, the same experiments that caused collision even without the balance task are repeated with it activated. Despite the complexity of the problem being increased, its addition is initially beneficial. Having to be mindful of the ball prevents the fatal acceleration towards the obstacles. This is an example of a coincidental synergy between tasks.

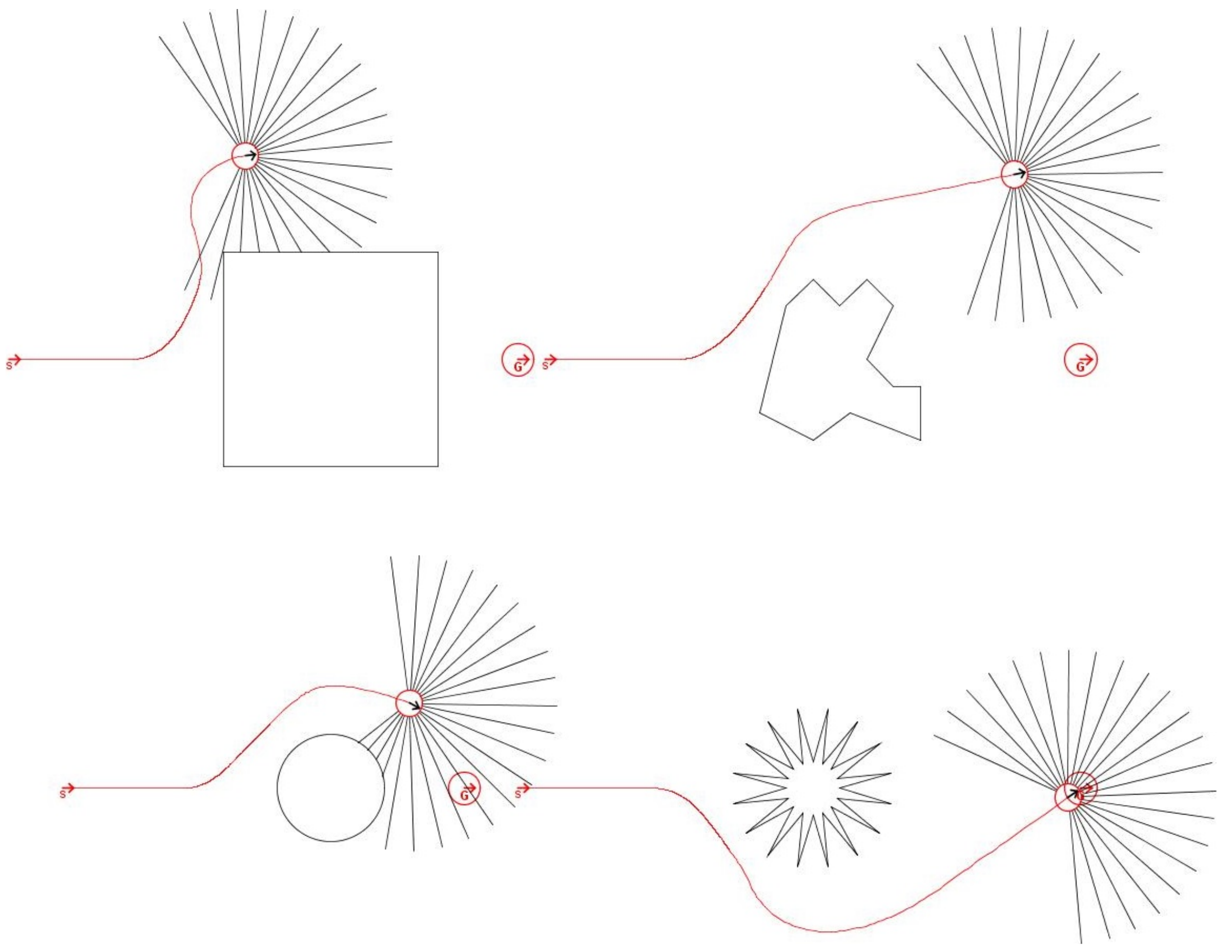


Figure 7.15: Collection of test cases showing the behaviour resulting from the simultaneous execution of all tasks

Although collision is avoided, only one of the experiments is completed with all tasks being satisfied.

The star-shaped obstacle is successfully navigated with the robot reaching its goal having averted all but very minor perturbations of the ball. Keeping the ball's energy low is not a challenge to begin with as the robot suffers from the usual pattern of slow movement already discussed. This is why the energy curve (see figure 7.16) is almost completely flat, with only a small spike marking the point of increased acceleration that occurs once the goal is in view.

One may get the impression that the other robots are also on course to reach their respective goals. Yet, in the screenshot above, balance has already been lost due to a sudden spike in energy. Possible reasons for this are explored in the following.

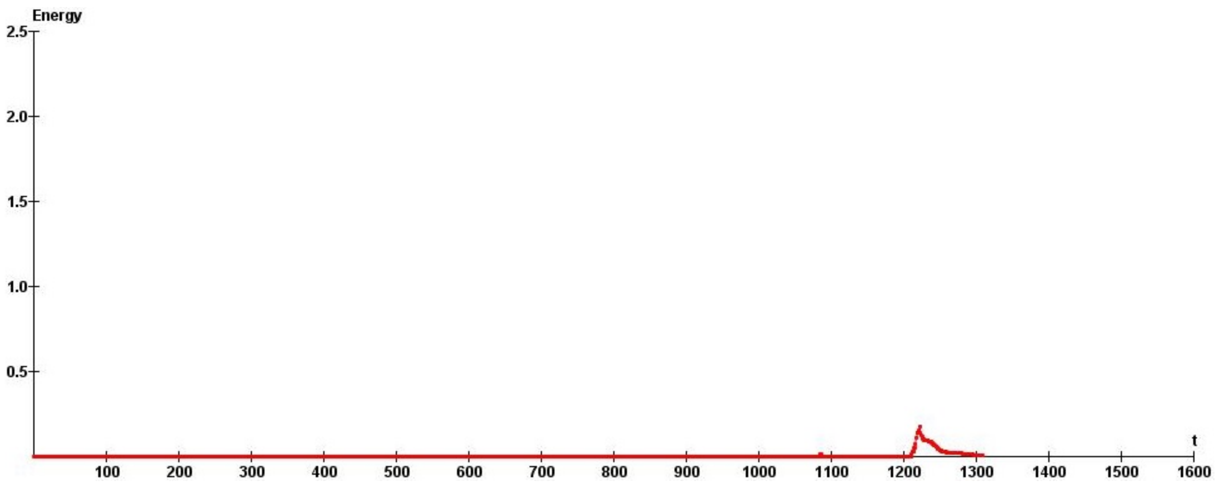


Figure 7.16: Energy graph corresponding to the successful experiment shown above where the robot navigates the star-shaped obstacle with all tasks in play simultaneously

The energy graphs for the failed experiments are very similar, which is why only that of the circular obstacle is given as an example.

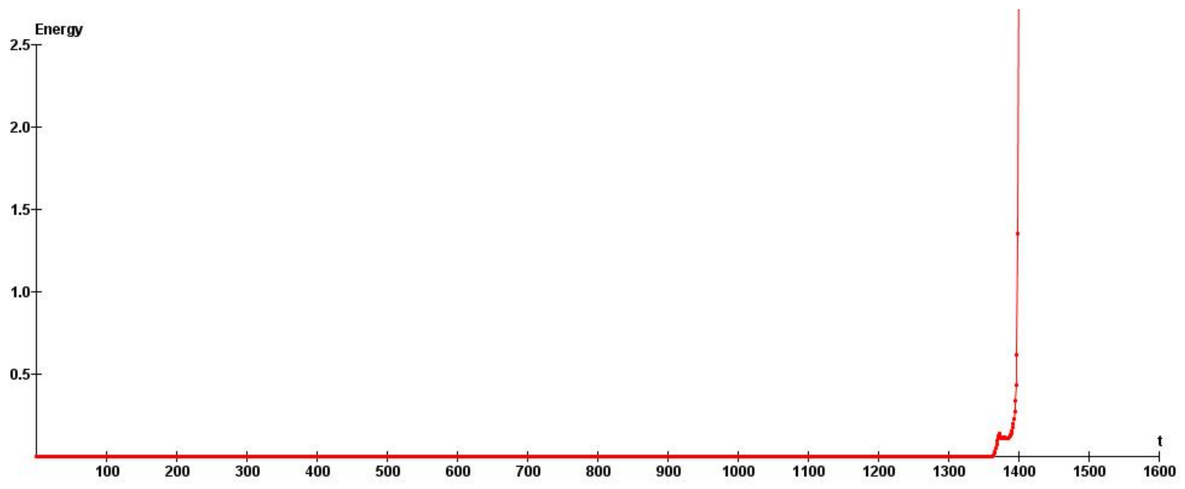


Figure 7.17: Energy graph corresponding to the failed experiment shown above where the robot navigates a circular obstacle with all tasks in play simultaneously

As previously observed, the robot can get dangerously close to obstacles obstructing its path to the goal location. After very slow movement close to the edge of the obstacle, the robot finally faces free space again. At this point the collision avoidance task's pronounced preference for acceleration dominates overall behaviour. Proximity urgency is the only one with a powerful bias towards high speeds when it has the chance to increase obstacle distance. Not having the foresight to predict long-term consequences for the ball's balance, the robot recklessly accelerates. At the same time it also turns towards the goal, adding radial acceleration.

The same holds true for the successful case however, so there must be an additional reason to explain the intensity of the energy spike. Two possible explanations are given, although neither is conclusive.

The first possibility is that the controls required to becalm the ball are not within the realisable region of control space. Accelerating to flee the obstacle coupled with the turn towards the goal location restricts sampling to similar controls to those just executed. These may not include an action that generates a force in the opposite direction to the ball's motion. Then, the controller is not only prohibited from selecting controls that counteract the forces on the ball, but might even be forced to select controls that exacerbate the problem. Of course, the robot should never be in this impossible situation to begin with. In this case, a lack of foresight would be to blame.

It is strongly suspected that the above issue can and does occur in this and possibly other situations. Yet, the abrupt hike in energy over the course of just 10 time steps seems unnatural. If this were true, it would suggest that energy could be created out of nowhere by an error in the physics engine. No such error could be found however. There is one case in which a singularity occurs which might be responsible. Recall the formula for the radius of horizontal rotation.

$$r_\varphi = r_\theta \ \sin\theta \tag{7.1}$$

At the bottom of the bowl, $\sin\theta = 0$ which implies $r_\varphi = 0$. Later we divide by this radius, causing a potential division by zero in the formula:

$$a_\varphi = \frac{\vec{a}_{eff} \cdot \hat{\varphi}}{r_\varphi} \quad rad/s \tag{7.2}$$

This case is handled by setting $r_\varphi = \varepsilon$ (some small constant) in this situation. All evidence points towards this solution having the desired effect. After all, the ball starts at the exact bottom of the bowl in every test case and there has never before been an issue with this.

Lastly, the problem may stem from the discrete time steps with which the physics simulation works. The ball effectively jumps to its predicted location after each 16 millisecond interval. All experimentation shows that fluid and visually realistic behaviour is achieved with re-computation of the ball's position at this frequency. So in the end, the cause of the energy spike remains elusive.

7.4 Interpretation of Results

The preceding section makes best efforts to describe and explain the phenomena observed throughout the series of tests conducted. Given the multitude of factors at work, it is acknowledged that not all interpretations and inferences are beyond doubt. There may also be further issues not yet discovered due to the endless possible combinations of parameters involved in setting up test cases. The following briefly summarises key insights gained based on the present understanding of this complex system.

Combinations of pairs of tasks involving balance were most successfully coordinated. The robot avoids unnecessary contentions between the tasks by driving slowly. While this may not have been the expected solution, the controller cannot be faulted for taking advantage of the absence of a deadline.

Initial experiments involving goal seeking in conjunction with obstacle navigation were promising. Obstacles are engaged as needed and their contours left when safe to do so. Paths are smooth but take on slightly more risk than desired by leaving only a small gap between the vehicle and the obstacle. Symptoms could potentially be alleviated by modification of the HMI formula.

The real problem is a lack of foresight and the discovery that the fourth hypothesis can no longer be upheld. If an obstacle is encountered at speed, the contour joining task's urgency does not report a sufficiently large urgency to convey the need for initiating a turn. This is because doing so would require a longer term plan to adjust the robot's speed over multiple steps.

Only one of the experiments with all tasks competing at the same time was successful. The rest were able to avoid obstacles and even started out on a good path towards the goal location but then lost balance. Reasons for failure could only be speculated, but a lack of foresight is likely responsible to some degree here as well.

The biggest concern is the increase in computational complexity that comes with the apparently unavoidable switch to a multi-step planner. It was for this reason that the fourth hypothesis about local search was made to begin with. Much effort was put into ensuring that each task could succeed in its own goals when executing in isolation. Essentially, the idea was to make sure that each task has optimal substructure or at least a weaker, non-optimal version of the concept whereby local solutions are part of a global solution. It was naïvely hoped that the task coordination system would then automatically inherit this property, provided all individual tasks possessed it. This may have been the case if it were not for the restriction on acceleration and deceleration, which seem to play a big part in dashing this hope.

On a more positive note, the homeostatic coordination mechanism itself was never implicated in any of the failed experiments. Control samples associated with the lowest HMI rating were consistently selected at every stage and that rating was faithfully calculated from the urgencies reported by individual tasks. It is these task-specific urgency heuristics that were found to be at fault as they did not reliably represent future ramifications of taking certain actions in the present. Since the coordination mechanism depends on task urgencies to accurately reflect the danger a task is in, it cannot be expected to produce good compromise controls when those urgencies are under-reported due to lack of foresight. In other words, the issue lies with task solutions that are plugged into the coordination framework rather than with the conflict resolution technique it employs. This gives reason to hope that the system holds more potential that could be unlocked with the resolution of the identified issues.

CHAPTER EIGHT

CONCLUSION AND FUTURE WORK

The final chapter concludes and evaluates the work undertaken as part of this thesis. A brief summary reminds the reader of initial objectives and what was done to meet them by discussing the contributions made. The research question is addressed separately, before the thesis culminates in a section on future work and possible improvements.

8.1 Summary

This thesis has presented work within the field of action selection and some adjacent areas. Specifically, the challenge of creating a general, task-agnostic coordination mechanism for mobile robots was addressed. The development of such a system was motivated by prospects of increasing robot autonomy and unlocking some of the still untapped hardware potential. A plug and play architecture was designed for reusability and with the aim of shortening the development cycle of robot software, thereby also cutting costs and making the final products more economical.

Work towards this goal began with a literature review in the form of a new taxonomy of existing action selection mechanisms. The taxonomy distinguishes between interleaved-, parallel- and simultaneous multi-tasking. SMT is the most complex of these as it requires all tasks to be considered in the selection of each action. Contentions inevitably arise between tasks competing for control of the same actuators. Attempts at coordination so far struggle with the assignment of relative priorities that accurately reflect how much attention each task warrants at a given time.

The principle of homeostasis was appraised as a potential basis for defining a universal scale on which the urgencies of all tasks can be fairly rated and compared. The main contribution of this PhD is the design and implementation of a multi-tasking framework that incorporates this idea. Each task submits an urgency value in a standard range, without having to consider the needs of other tasks or even be aware of their presence. These urgencies are combined to form the so-called homeostatic mortality index, an overall measure of coordination success. The HMI is based on an exponential function that prevents individual tasks suffering in the pursuit of higher average satisfaction. Homeostatic mortality reduction is the process by which actions representing viable compromises are selected from a pool of realisable controls. The framework does not prescribe a particular algorithm for searching control space, meaning that the sampling approach taken here can be replaced by other methods as required.

In order to evaluate the developed system, demonstrator tasks were constructed to reach a goal location, prevent collision, follow a contour around obstacles and balance a ball on top of a differential drive robot.

The goal location task is guided by an urgency heuristic defined in terms of strain energy. Smooth, goal-connecting paths are generated between (x, y, ϕ) start configurations and (x, y) goals. Biarcs provide a means of strain estimation and can be constructed in $O(1)$ time by choosing pairs of arcs that approximate the turn-distribution of clothoids.

Obstacle navigation was decomposed into the two tasks of collision prevention and contour joining. The urgency of the former is based solely on distance to the closest obstacle, while the latter reuses the solution to the goal location task to smoothly align with a contour following the course of the obstacle. Thanks to insights borrowed from Bug Algorithms, obstacles are engaged as needed and can be left as soon as it is safe to do so.

Ball balancing was introduced as a novel task for mobile robots. A physics model was designed from the ground up to simulate and predict the behaviour of a ball in a spherical bowl atop a moving robot. Various forces act on the ball as the robot accelerates, decelerates and changes direction. The ball is kept within its vessel by selecting actions that reduce its potential and kinetic energies, on the basis of which the task's urgency is defined.

Being able to solve these tasks in compliance with the structure prescribed by the task coordination framework vindicates both the first and second hypotheses.

Having assembled a group of suitable tasks for demonstration, the coordination system was put to the test with the aim of verifying the third hypothesis. Experiments were conducted using a robot simulator programmed in Java, after having unsuccessfully tried to integrate the developed physics model into the formerly commercial application, Webots.

Results were most encouraging for pairs of tasks, which were generally well coordinated. This incites confidence in the third hypothesis, but the single-step planner's lack of foresight quickly led to the collapse of the fourth. Planning only one step ahead caused collision in some of the experiments coordinating goal seeking with obstacle navigation. Only one test with all tasks active at the same time was completed successfully, with the others struggling for reasons not yet fully understood. The homeostatic mechanism appears not to be responsible for any of the limitations discussed at length in Chapter 7. In addition to the planner's lack of foresight, urgency heuristics for individual tasks do not accurately reflect the future consequences of taking certain actions, leading to decisions being made with imperfect information. Implications for the research question are discussed below.

8.2 Answering the Research Question

The research goal of this thesis was to answer the question:

Can a general, task-agnostic framework be devised to coordinate multiple tasks being simultaneously executed by a single robot through homeostatic conflict resolution?

Despite mixed results, the question can be tentatively answered in the affirmative. The homeostatic task coordination mechanism fulfils the requirement for genericity and was not found to be responsible for any of the failed experiments. On the contrary, it did succeed in the absence of issues caused by other parts of the system. Evidence of sensible compromises being struck was found when coordinating the balancing and goal location tasks together. Both speed and curvature were adjusted to reduce the ball's energy on the journey to the robot's destination. A similar strategy was successfully employed for balancing the ball while navigating obstacles. In the combination of goal seeking and obstacle navigation, the robot did venture further into the safety-zone around obstacles than desired. Nevertheless, the goal was attained in initial experiments. Until problems concerning the single-step planner's lack of foresight crystallised, the empirical evidence was quite promising indeed. This suggests that if the identified problems were removed, the homeostatic mechanism could be used to its full potential. The erroneous assumption regarding the sufficiency of local search made by the fourth hypothesis appears to be the main obstacle in the way of a satisfactory verification of the developed theory and a more definitive answer to the research question. At the very least, further investigation is warranted. Ideas for how to proceed are given in the following section.

8.3 Future Work

The scope of this thesis was initially underestimated and proved hard to contain. Solving each of the demonstrator tasks could have been a project in its own right, without even considering the overarching coordination aspect that ties them all together. Time had to be spread thinly to cover all areas, leaving room for improvement in each. The following identifies some aspects that are deemed most worthy of further attention and improvement.

8.3.1 Improving Existing Tasks

The goal location task is the least developed of the demonstrators. Its ability to reach a goal orientation was sacrificed for lower complexity. Even if it does require more computation, regaining the capability to select (x, y, ϕ) goals is an attractive prospect. There are many existing solutions that provide this feature. It would be interesting to measure their actual run-times and, if necessary, explore the possibility of reducing their complexity, e.g. through an approximation.

Obstacle detection does not use a feature extraction technique and cannot distinguish between different obstacles. In more cluttered environments, the ability to follow a specific obstacle may be required. The merge point finding algorithm can also be refined, although this would also come at the price of higher complexity.

The physics model developed for ball balancing does not account for rotational inertia or the diameter of the ball. While adding these factors is unlikely to have a significant impact, it would further improve the realism of the simulation. More importantly, the cause of the energy spike in the last of the coordination tests has not yet been found. The error may or may not be in the physics model, but it would be good to know either way.

8.3.2 Adding New Tasks

The possibility of adding new tasks to the pool of demonstrators is even more interesting than improving on existing solutions.

The absence of one task in particular has already been noted. Without a time-sensitive task, the robot is able to avoid conflict by moving slowly. The inclusion of a task to discourage this strategy could put the system under more stress. This would allow the limits of the coordination mechanism to be investigated and better understood. It would also remove the need for the artificial motivation to accelerate that had to be built into the goal location task (see Section 4.4.5). Incrementing curvature to make the robot drive along a zero-strain line would no longer be necessary as the motivation to reach the goal would be provided by the time-sensitive task.

Moreover, the *discount factor* (DF = 0.97) could be removed from equation 4.26, thus simplifying the location task's strain based urgency function. The task would still rely on a heuristic, but that heuristic would be purely defined in terms of path strain. Not mixing in factors relating to time, which have nothing to do with the shape of the path, would make for a much cleaner solution. Being able to reach the goal location in a given amount of time may also help demonstrate that the system is capable of real-time planning. The implemented controller already operates in real-time, as is evidenced by its ability to simulate ball physics accurately using very short time intervals. Seeing a robot move fast might, however, be a more obvious or effective way of showcasing this ability. The exact details of a time-sensitive task could prove quite difficult to specify. It is not as simple as setting a deadline because the robot would have to be able to revise the time needed as and when necessary. For example, the discovery of new obstructions en route to the goal would call for an extension.

A resource management task would also be an interesting addition. Two different ways of managing the robot's battery life have already been used as examples in the taxonomy of tasks. The first is a multi-phase task consisting of two single-phase task, namely driving to a charge point (category 1) and connecting the robot to the power supply (category 2). The second way of ensuring the robot will not run out of battery is to use an energy rationing task. In the context of SMT, this category 3 task is of more interest since the above version is more of a task allocation than a coordination problem. Energy rationing would involve defining an urgency mapping that reflects which actions most effectively use the remaining energy before the next charge point is reached. This may involve selecting shorter paths or driving at the speed resulting in best fuel/energy economy. The robot simulator, which does not yet model resource constraints, would have to be adapted to support this task.

The tasks implemented so far have used heuristics to map internal and external state variables to urgency values. Further tasks, including those suggested, could showcase the ability to use other methods. Alternatives include using a lookup table or training an artificial neural network to perform the mapping instead of heuristics.

8.3.3 Improving the Coordination System

To really improve on the coordination mechanism, its single-step planner must be swapped for one with the capacity to plan further ahead. This is likely a major project and could, in the end, be thwarted by complexities outside the realm of real-time planning.

As already mentioned, it may be possible to improve coordination by further refinement of the formula for the homeostatic mortality index. Proper analysis of the benefits of different formulae will likely not be possible without first overcoming the more severe issues of lacking foresight, which will otherwise distort results.

Finally, the addition of a scheduler for breaking down multi-phase tasks into single-phase tasks would open the door to exploring a broader range of tasks.

8.3.4 Further Testing

One can never be sure enough that such a complex system behaves as it should. With an endless number of possible test cases to choose from, there are plenty left that could not be explored within the scope of this thesis. To make better use of the coordination mechanism's generic design, it could also be evaluated using different types of robots. The realism of the system could be put to the test by moving from simulations to tests with physical robot. The journey from simulation to physical robotics is known to be treacherous, but a number of considerations were already made at the outset to facilitate this transition. As per the requirements laid out in Section 3.2, the framework was designed to make as few assumptions as possible, adapt to sensor information and compute controls in real-time. The implementation also makes sure to select realisable controls that obey the vehicle's equations of motion. Together, these features provide a basis for further exploration.

REFERENCES

[1] ARKIN, R. C. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *International Conference on Robotics and Automation* (New York, NY, USA, Mar 1987), vol. 4 of *ICRA*, IEEE, pp. 264–271.

[2] ARKIN, R. C. Dynamic replanning for a mobile robot based on internal sensing. In *International Conference on Robotics and Automation* (New York, NY, USA, May 1989), vol. 3 of *ICRA*, IEEE, pp. 1416–1421.

[3] ARKIN, R. C. Homeostatic Control For A Mobile Robot: Dynamic Replanning In Hazardous Environments. In *Mobile Robots III* (Bellingham, WA, USA, Mar 1989), W. J. Wolfe, Ed., vol. 1007, SPIE, pp. 407–413.

[4] ARKIN, R. C. Towards the Unification of Navigational Planning and Reactive Control. In *American Association for Artificial Intelligence Spring Symposium on Robot Navigation* (Stanford, CA, USA, Mar 1989), Stanford University, pp. 1–5.

[5] ARKIN, R. C. *Behavior-based Robotics*, 1 ed. MIT Press, Cambridge, MA, USA, 1998.

[6] ARKIN, R. C., AND MACKENZIE, D. Temporal coordination of perceptual algorithms for mobile robot navigation. In *Transactions on Robotics and Automation* (New York, NY, USA, Jun 1994), vol. 10, IEEE, pp. 276–286.

[7] ASHBY, W. R. *Design for a Brain : The origin of adaptive behaviour*, 2 ed. Chapman & Hall, London, UK, 1960.

[8] BALLARD, D. H. Generalizing the Hough Transform to Detect Arbitrary Shapes. In *Pattern Recognition* (Oxford, UK, 1981), vol. 13, Pergamon Press Ltd., pp. 111–122.

[9] BENJAMIN, M. *Interval programming: A multi-objective optimization model for autonomous vehicle control*. PhD thesis, Brown University, Providence, RI, USA, Jan 2002.

[10] BENJAMIN, M., GRUND, M., AND NEWMAN, P. Multi-objective optimization of sensor quality with efficient marine vehicle task execution. In *International Conference on Robotics and Automation* (New York, NY, USA, May 2006), ICRA, IEEE, pp. 3226–3232.

[11] BENJAMIN, M., LEONARD, J. J., SCHMIDT, H., AND NEWMAN, P. M. An Overview of MOOS-IvP and a Brief Users Guide to the IvP Helm Autonomy Software. Tech. Rep. MIT-CSAIL-TR-2009-028, Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA, USA, Jun 2009.

[12] BENNETT, A. A., AND LEONARD, J. J. A behavior-based approach to adaptive feature detection and following with autonomous underwater vehicles. In *Journal of Oceanic Engineering* (New York, NY, USA, Apr 2000), vol. 25, IEEE, pp. 213–226.

[13] BERTOLAZZI, E., AND FREGO, M. A Note on Robust Biarc Computation. In *Computer-Aided Design and Applications* (Aurora, IL, USA, Nov 2017), vol. 16, CAD Solutions LLC, pp. 822–835.

[14] BLOCH, V., DEGANI, A., AND BECHAR, A. *Task characterization and classification for robotic manipulator optimal design in precision agriculture*. Wageningen Academic Publishers, Wageningen, Netherlands, 2015, pp. 313–320.

[15] BOLTON, K. M. Biarc curves. In *Computer-Aided Design* (Amsterdam, Netherlands, 1975), vol. 7, Elsevier Inc., pp. 89–92.

[16] BOMEY, N. Uber self-driving car crash: Vehicle detected Arizona pedestrian 6 seconds before accident. https://eu.usatoday.com/story/money/cars/2018/05/24/uber-self-driving-car-crash-ntsb-investigation/640123002/, 2018. Accessed Aug 2018.

[17] BOSTON DYNAMICS. Changing your idea of what robots can do. https://www.bostondynamics.com/, 2018. Accessed Aug 2018.

[18] BROOKS, R. A. A Robust Layered Control System for a Mobile Robot. In *Journal on Robotics and Automation* (New York, NY, USA, Mar 1986), vol. 2, IEEE, pp. 14–23.

[19] BROOKS, R. A. Intelligence Without Reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1991), vol. 1 of *IJCAI*, Morgan Kaufmann Publishers Inc., pp. 569–595.

[20] BROOKS, R. A. Intelligence Without Representation. In *Artificial Intelligence* (Amsterdam, Netherlands, Jan 1991), vol. 47, Elsevier Inc., pp. 139–159.

[21] BROOKS, R. A. *Cambrian Intelligence: The Early History of the New AI*. MIT Press, Cambridge, MA, USA, 1999.

[22] BROWNLEE, J. The pole balancing problem: a benchmark control theory problem. Tech. rep., Swinburne University of Technology, Melbourne, Australia, Jul 2005.

[23] CANNON, W. B. *The Wisdom of the Body*. W.W. Norton & Company, Inc., New York, NY, USA, 1932.

[24] CHOI, J.-W., CURRY, R., AND ELKAIM, G. Path Planning Based on Bézier Curve for Autonomous Ground Vehicles. In *Advances in Electrical and Electronics Engineering – IAENG Special Edition of the World Congress on Engineering and Computer Science* (New York, NY, USA, Oct 2008), IEEE, pp. 158–166.

[25] CHOSET, H., M LYNCH, K., HUTCHINSON, S., KANTOR, G., BURGARD, W., KAVRAKI, L., AND THRUN, S. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, USA, Jan 2005.

[26] CHRISTENSEN, H. I., AND PIRJANIAN, P. Theoretical methods for planning and control in mobile robotics. In *International Conference on Conventional and Knowledge Based Intelligent Electronic Systems* (New York, NY, USA, May 1997), vol. 1 of *KES*, IEEE, pp. 81–86.

[27] CYBERBOTICS. Webots: robot simulator. https://cyberbotics.com/. Accessed Jul 2019.

[28] DECUGIS, V., AND FERBER, J. Action Selection in an Autonomous Agent with a Hierarchical Distributed Reactive Planning Architecture. In *International Conference on Autonomous Agents* (New York, NY, USA, 1998), vol. 2 of *AGENTS*, ACM, pp. 354–361.

[29] DENISCO-RAYOME, A. Dossier: The leaders in self-driving cars. https://www.zdnet.com/article/dossier-the-leaders-in-self-driving-cars/, 2018. Accessed Aug 2018.

[30] DI PAOLO, E. A. Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. In *From Animals to Animats* (Cambridge, MA, USA, Jan 2000), vol. 6 of *SAB*, MIT Press.

[31] DI PAOLO, E. A. Evolving Robust Robots Using Homeostatic Oscillators. In *Cognitive Science Research Paper 526* (Brighton, UK, Apr 2002), COGS, University of Sussex.

[32] DI PAOLO, E. A. Organismically-inspired robotics: homeostatic adaptation and teleology beyond the closed sensorimotor loop. In *Dynamical Systems Approach to Embodiment and Sociality* (Adelaide, Australia, Jan 2003), K. Murase and T. Asakura, Eds., Advanced Knowledge International, pp. 19–42.

[33] DRUMWRIGHT, E., HSU, J., KOENIG, N., AND SHELL, D. Extending Open Dynamics Engine for Robotics Simulation. In *Simulation, Modeling, and Programming for Autonomous Robots* (Berlin, Heidelberg, Nov 2010), N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Eds., no. 38–50, Springer.

[34] DUBINS, L. E. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. In *American Journal of Mathematics* (Baltimore, MD, USA, 1957), vol. 79, Johns Hopkins University Press, pp. 497–516.

[35] DUDEK, G., AND JENKIN, M. *Computational Principles of Mobile Robotics*, 2 ed. Cambridge University Press, Cambridge, UK, Jan 2010.

[36] ERLHAGEN, W., MUKOVSKIY, A., BICHO, E., PANIN, G., KISS, C., KNOLL, A., SCHIE, H., AND BEKKERING, H. Goal-directed imitation for robots: A bio-inspired approach to action understanding and skill learning. In *Robotics and Autonomous Systems* (Amsterdam, Netherlands, May 2006), vol. 54, Elsevier, pp. 353–360.

[37] EUREKA! Smooth moves for service robotics. http://www.eurekamagazine.co.uk/design-engineering-features/technology/smooth-moves-for-service-robotics/143300/, 2016. Accessed Oct 2018.

[38] EVERAERE, P., AND GRISLIN-LE STRUGEON, E. Continuous Preferences for Action Selection. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (Setúbal, Portugal, Jan 2011), vol. 2 of *ICAART*, SciTePress, pp. 54–63.

[39] FULL MOTION DYNAMICS. Ball and Plate PID control with 6 DOF Stewart platform. https://www.youtube.com/watch?v=j4OmVLc_oDw, 2012. Accessed Sep 2018.

[40] GADANHO, S. C. Learning Behavior-Selection by Emotions and Cognition in a Multi-Goal Robot Task. In *Journal of Machine Learning Research* (Jul 2003), vol. 4, JMLR.org, pp. 385–412.

[41] GADANHO, S. C., AND CUSTÓDIO, L. M. M. Learning Behavior-selection in a Multi-goal Robot Task. In *Perception and Emotion Based Reasoning* (Ljubljana, Slovenia, Jan 2003), vol. 27, The Slovene Society Informatica, pp. 175–183.

[42] Garrido, S., Moreno, L., Abderrahim, M., and Blanco, D. FM2: A Real-Time Sensor-Based Feedback Controller For Mobile Robots. In *International Journal of Robotics and Automation* (Calgary, Canada, Oct 2009), M. Kamel, Ed., vol. 24, ACTA Press, pp. 3169–3192.

[43] Gat, E. Artificial intelligence and mobile robots. MIT Press, Cambridge, MA, USA, 1998, ch. Three-layer Architectures, pp. 195–210.

[44] Gerkey, B. P., and Matarić, M. J. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. In *The International Journal of Robotics Research* (London, UK, 2004), vol. 23, Sage Publications Ltd., pp. 939–954.

[45] Gim, S., Adouane, L., Lee, S., and Dérutin, J.-P. Clothoids Composition Method for Smooth Path Generation of Car-Like Vehicle Navigation. In *Journal of Intelligent & Robotic Systems* (Boston, MA, USA, Oct 2017), vol. 88, Springer US, pp. 129–146.

[46] Gross, C. G. Claude Bernard and the Constancy of the Internal Environment. In *The Neuroscientist* (Thousand Oaks, CA, USA, 1998), vol. 4, Sage Publications, pp. 380–385.

[47] Groth, C., and Henrich, D. Multi-tasking of competing behaviors on a robot manipulator. In *International Conference on Intelligent Robots and Systems* (New York, NY, USA, Nov 2013), IEEE/RSJ, IEEE, pp. 3057–3064.

[48] Halliday, D., Resnick, R., and Walker, J. *Fundamentals of Physics*. Halliday & Resnick Fundamentals of Physics. John Wiley & Sons Canada, Ltd., Hoboken, NJ, USA, 2010.

[49] Hamann, H. *Swarm Robotics: A Formal Approach*, 1 ed. Springer International Publishing, Basel, Switzerland, 2018.

[50] Hameed, I. A. Coverage Path Planning Software for Autonomous Robotic Lawn Mower using Dubins' Curve. In *International Conference on Real-Time Computing and Robotics* (New York, NY, USA, Jul 2017), RCAR, IEEE, pp. 517–522.

[51] Hernández-Sosa, D., Lorenzo, J., Domínguez-Brito, A. C., and Isern, J. A Proposal of a Homeostatic-Adaptive Control for a Robotic System. In *Workshop de Agentes Físicos* (Spain, Apr 2006), J. Cabrera-Gámez and V. M. Olivera, Eds., WAF, Universidad de Las Palmas de Gran Canaria, pp. 75–82.

[52] Hertig, L., Schindler, D., Bloesch, M., David Remy, C., and Siegwart, R. Unified state estimation for a ballbot. In *International Conference on Robotics and Automation* (New York, NY, USA, May 2013), ICRA, IEEE, pp. 2471–2476.

[53] Horn, B. K. P. The Curve of Least Energy. In *ACM Transactions on Mathematical Software* (New York, NY, USA, Dec 1983), vol. 9, ACM, pp. 441–460.

[54] Hougen, D., Fischer, J., and Johnam, D. A neural network pole balancer that learns and operates on a real robot in real time. In *Proceedings of the MLC-COLT Workshop on Robot Learning* (New Brunswick, NJ, USA, Jul 1994), Rutgers University, pp. 73–80.

[55] Humphrys, M. *Action selection methods using reinforcement learning*. PhD thesis, University of Cambridge, Cambridge, UK, 1996.

[56] Iizuka, H., and Di Paolo, E. Extended Homeostatic Adaptation: Improving the Link between Internal and Behavioural Stability. In *From Animals to Animats* (Berlin, Germany, Jul 2008), M. Asada, J. C. T. Hallam, J.-A. Meyer, and J. Tani, Eds., vol. 10 of *SAB*, Springer, pp. 1–11.

[57] J. Lumelsky, V., and A. Stepanov, A. Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape. In *Algorithmica* (Basel, Switzerland, Nov 1987), vol. 2, Springer International Publishing, pp. 403–430.

[58] J. Lumelsky, V., Mukhopadhyay, S., and Sun, K. Dynamic path planning in sensor-based terrain acquisition. In *Transactions on Robotics and Automation* (New York, NY, USA, Aug 1990), vol. 6, IEEE, pp. 462–472.

[59] Jaeger, H., and Christaller, T. Dual Dynamics: Designing Behavior Systems for Autonomous Robots. In *Artificial Life and Robotics* (Berlin, Germany, Sep 1998), vol. 2, Springer, pp. 108–112.

[60] Jakubczyk, K. Approximation of Smooth Planar Curves by Circular Arc Splines. https://pdfs.semanticscholar.org/f749/b6faca7c4486f8ede7aa633c494f37b3e86e.pdf, 2012. Accessed Feb 2020.

[61] Jones, E., Browning, B., Dias, M., Argall, B., Veloso, M., and Stentz, A. Dynamically Formed Heterogeneous Robot Teams Performing Tightly-coordinated Tasks. In *International Conference on Robotics and Automation* (New York, NY, USA, Feb 2006), ICRA, IEEE, pp. 570–575.

[62] Joyeux, S., Alami, R., and Lacroix, S. A Software component for simultaneous plan execution and adaptation. In *International Conference on Intelligent Robots and Systems* (New York, NY, USA, Oct 2007), IEEE/RSJ, IEEE, pp. 3038–3043.

[63] Juster, J. *Speech and Gesture Understanding in a Homeostatic Control Framework for a Robotic Chandelier*. PhD thesis, MIT, Cambridge, MA, USA, Sep 2004.

[64] KAMON, I., AND RIVLIN, E. Sensory-based motion planning with global proofs. In *Transactions on Robotics and Automation* (New York, NY, USA, Jan 1998), vol. 13, IEEE, pp. 814–822.

[65] KAWABATA, K., MA, L., XUE, J., ZHU, C., AND ZHENG, N. A path generation for automated vehicle based on Bézier curve and via-points. In *Robotics and Autonomous Systems* (Amsterdam, Netherlands, Aug 2015), vol. 74, Elsevier Inc., pp. 243–252.

[66] KHATIB, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *The International Journal of Robotics Research* (Cambridge, MA, USA, Mar 1986), vol. 5, MIT, pp. 90–98.

[67] KIM, Y. H., PARK, J. B., SON, W. S., AND YOON, T. S. Modified Turn Algorithm for Motion Planning Based on Clothoid Curve. In *Electronics Letters* (Stevenage, UK, Oct 2017), vol. 53, Institution of Engineering and Technology, pp. 1574–1576.

[68] KNIPS, G., ZIBNER, S. K. U., REIMANN, H., AND SCHÖNER, G. A Neural Dynamic Architecture for Reaching and Grasping Integrates Perception and Movement Generation and Enables On-Line Updating. In *Frontiers in Neurorobotics* (Lausanne, Switzerland, Mar 2017), C. Tetzlaff, Ed., vol. 11, Frontiers Media S.A., pp. 9:1–14.

[69] KORSAH, G., STENTZ, A., AND DIAS, M. A comprehensive taxonomy for multi-robot task allocation. In *International Journal of Robotics Research* (Cambridge, MA, USA, Oct 2013), vol. 32, MIT, pp. 1495–1512.

[70] KOTOVSKY, K., HAYES, J. R., AND SIMON, H. A. Why are some problems hard? Evidence from Tower of Hanoi. In *Cognitive Psychology* (Amsterdam, Netherlands, 1985), vol. 17, Elsevier Inc., pp. 248–294.

[71] KOŠECKÁ, J., AND BAJCSY, R. Discrete Event Systems for autonomous mobile agents. In *Robotics and Autonomous Systems* (Amsterdam, Netherlands, Apr 1994), vol. 12, Elsevier Inc., pp. 187–198.

[72] KUMAR, P. R., AND BANDYOPADHYAY, B. Variable Gain Super Twisting Controller for the Position Stabilization of Stewart Platform. In *IFAC Proceedings Volumes* (Amsterdam, Netherlands, Mar 2014), vol. 47, Elsevier Inc., pp. 115–121.

[73] LAUWERS, T., KANTOR, G., AND HOLLIS, R. A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive. In *International Conference on Robotics and Automation* (New York, NY, USA, Jun 2006), ICRA, IEEE, pp. 2884–2889.

[74] LAUWERS, T., KANTOR, G., AND HOLLIS, R. One is enough! In *Robotics Research* (Berlin, Germany, May 2007), S. Thrun, R. Brooks, and H. Durrant-Whyte, Eds., vol. 28, Springer, pp. 327–336.

[75] LEIDNER, D., BORST, C., DIETRICH, A., BEETZ, M., AND ALBU-SCHÄFFER, A. Classifying compliant manipulation tasks for automated planning in robotics. In *International Conference on Intelligent Robots and Systems* (New York, NY, USA, 2015), IEEE/RSJ, IEEE, pp. 1769–1776.

[76] LENSER, S., BRUCE, J., AND VELOSO, M. A Modular Hierarchical Behavior-Based Architecture. In *RoboCup 2001: Robot Soccer World Cup V* (Berlin, Germany, 2002), A. Birk, S. Coradeschi, and S. Tadokoro, Eds., Springer, pp. 423–428.

[77] LIN, L. J. Scaling Up Reinforcement Learning for Robot Control. In *International Conference on International Conference on Machine Learning* (San Francisco, CA, USA, 1993), vol. 10 of *ICML*, Morgan Kaufmann Publishers Inc., pp. 182–189.

[78] LIN, P. Relationships with Robots: Good or Bad for Humans? `https://www.forbes.com/sites/patricklin/2016/02/01/relationships-with-robots-good-or-bad-for-humans/#47f376407adc`, 2016. Accessed Aug 2018.

[79] LISCANO, R., AND GREEN, D. Design And Implementation Of A Trajectory Generator For An Indoor Mobile Robot. In *International Conference on Intelligent Robots and Systems* (New York, NY, USA, Oct 1989), vol. 2 of *IEEE/RSJ*, IEEE, pp. 380–385.

[80] LOW, K. H., LEOW, W. K., AND JR, M. Integrated Planning and Control of Mobile Robot with Self-Organizing Neural Network. In *International Conference on Robotics and Automation* (New York, NY, USA, Jun 2003), vol. 4 of *ICRA*, IEEE.

[81] LU, L. Planar quintic G^2 Hermite interpolation with minimum strain energy. In *Journal of Computational and Applied Mathematics* (Amsterdam, Netherlands, Jan 2015), vol. 274, Elsevier Inc., pp. 109–117.

[82] LUGO-CÁRDENAS, I., FLORES, G., SALAZAR, S., AND LOZANO, R. Dubins path generation for a fixed wing UAV. In *International Conference on Unmanned Aircraft Systems* (New York, NY, USA, May 2014), ICUAS, IEEE, pp. 339–346.

[83] MACKENZIE, D. C., ARKIN, R. C., AND CAMERON, J. M. Multiagent Mission Specification and Execution. In *Autonomous Robots* (Hingham, MA, USA, Mar 1997), vol. 4, Kluwer Academic Publishers, pp. 29–52.

[84] MAES, P. The Dynamics of Action Selection. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1989), vol. 2 of *IJCAI*, Morgan Kaufmann Publishers Inc., pp. 991–997.

[85] MAES, P. Situated Agents Can Have Goals. In *Designing Autonomous Agents* (Cambridge, MA, USA, 1990), P. Maes, Ed., MIT Press, pp. 49–70.

[86] MOIOLI, R. C., VARGAS, P. A., VON ZUBEN, F. J., AND HUSBANDS, P. Evolving an Artificial Homeostatic System. In *Advances in Artificial Intelligence - SBIA 2008* (Berlin, Germany, 2008), G. Zaverucha and A. L. da Costa, Eds., Springer, pp. 278–288.

[87] NEAL, M. J., AND TIMMIS, J. Timidity: A Useful Emotional Mechanism for Robot Control? In *Informatica - special issue on perception and emotion based control* (Ljubljana, Slovenia, Jan 2003), vol. 27, The Slovene Society Informatica, pp. 197–204.

[88] O'BRIEN, M. J., AND ARKIN, R. C. An Artificial Circadian System for a Slow and Persistent Robot. In *From Animals to Animats* (Basel, Switzerland, 2018), P. Manoonpong, J. C. Larsen, X. Xiong, J. Hallam, and J. Triesch, Eds., vol. 15 of *SAB*, Springer International Publishing, pp. 149–161.

[89] PIEGL, L., AND TILLER, W. *The NURBS Book*, 2 ed. Springer, Berlin, Germany, 1997.

[90] PIRJANIAN, P. Behavior Coordination Mechanisms – State-of-the-art. Tech. rep., University of Southern California, Los Angeles, CA, USA, Oct 1999.

[91] PIRJANIAN, P., AND MATARIĆ, M. Multiple Objective vs. Fuzzy Behavior Coordination. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation* (Heidelberg, Germany, Nov 2001), D. Driankov and A. Saffiotti, Eds., Physica-Verlag HD, pp. 235–253.

[92] PIRJANIAN, P., AND PERRAM, P. J. *Multiple Objective Action Selection & Behavior Fusion using Voting*. PhD thesis, Aalborg University, Aalborg, Denmark, Aug 1998.

[93] PRIDAY, R. What's really going on in those Boston Dynamics robot videos? https://www.wired.co.uk/article/boston-dynamics-robotics-roboticist-how-to-watch, 2018. Accessed Aug 2018.

[94] QUARTERONI, A., SACCO, R., AND SALERI, F. *Numerical Mathematics (Texts in Applied Mathematics)*. Springer, Berlin, Germany, 2006.

[95] QUISPE, A. C. H., AMOR, H. B., AND CHRISTENSEN, H. I. A Taxonomy of Benchmark Tasks for Robot Manipulation. In *Robotics Research* (Cham, Switzerland, Jan 2018), vol. 1, Springer International Publishing, pp. 405–421.

[96] RAO, A. S., AND GEORGEFF, M. P. Modeling Rational Agents within a BDI-Architecture. In *International Conference on Principles of Knowledge Representation and Reasoning* (San Mateo, CA, USA, 1991), J. Allen, R. Fikes, and E. Sandewall, Eds., vol. 2 of *KR*, Morgan Kaufmann publishers Inc., pp. 473–484.

[97] RAVANKAR, A., RAVANKAR, A. A., KOBAYASHI, Y., HOSHINO, Y., AND PENG, C.-C. Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges. In *Sensors* (Basel, Switzerland, Sep 2018), vol. 18: 3170, MDPI.

[98] RIEKKI, J. *Reactive Task Execution of a Mobile Robot*. PhD thesis, Oulu University, Oulu, Finnland, 1999.

[99] ROBOTICS BUSINESS REVIEW. Factory Robot Kills Worker in India. https://www.roboticsbusinessreview.com/rbr/factory_robot_kills_worker_in_india/, 2015. Accessed Aug 2018.

[100] ROECKEL, M. W., RIVOIR, R. H., AND GIBSON, R. E. A behavior based controller architecture and the transition to an industry application. In *International Symposium on Intelligent Control Intelligent Systems and Semiotics* (New York, NY, USA, Sep 1999), IEEE, pp. 320–325.

[101] ROSENBLATT, J. K. DAMN: A Distributed Architecture for Mobile Navigation. In *Lessons Learned from Implemented Software Architectures for Physical Agents : Papers from the 1995 Spring Symposium* (Palo Alto, CA, USA, Mar 1995), H. Hexmoor and D. .Kortenkamp, Eds., AAAI Press, pp. 167–178.

[102] ROSENBLATT, J. K. *DAMN: A Distributed Architecture For Mobile Navigation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[103] ROSENBLATT, J. K. Optimal Selection of Uncertain Actions by Maximizing Expected Utility. In *Autonomous Robots* (Hingham, MA, USA, Aug 2000), vol. 9, Kluwer Academic Publishers, pp. 17–25.

[104] RUSS L. SMITH. Open Dynamics Engine. http://www.ode.org/, 2001. Accessed Dec 2019.

[105] SAFFIOTTI, A. The uses of fuzzy logic in autonomous robot navigation. In *Soft Computing* (Berlin, Germany, Dec 1997), vol. 1, Springer, pp. 180–197.

[106] SAFFIOTTI, A., RUSPINI, E. H., AND KONOLIGE, K. Using Fuzzy Logic for Mobile Robot Control. In *Practical Applications of Fuzzy Technologies* (Boston, MA, USA, 1999), H.-J. Zimmermann, Ed., Springer US, pp. 185–205.

[107] SANCHEZ FIBLA, M., BERNARDET, U., AND VERSCHURE, P. F. M. J. Allostatic control for robot behaviour regulation: An extension to path planning. In *International Conference on Intelligent Robots and Systems* (New York, NY, USA, 2010), IEEE/RSJ, IEEE, pp. 1935–1942.

[108] SCHMICKL, T., THENIUS, R., STRADNER, J., HAMANN, H., AND CRAILSHEIM, K. Robotic Organisms: Artificial Homeostatic Hormone System and Virtual Embryogenesis as Examples for Adaptive Reaction-Diffusion Controllers. In *IROS Workshop–Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo-Hybrid Systems* (New York, NY, USA, 2011), S. Kernbach and R. Fitch, Eds., IEEE.

[109] SCHÖNER, G., AND DOSE, M. A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion. In *Robotics and Autonomous Systems* (Amsterdam, Netherlands, Dec 1992), vol. 10, Elsevier Inc., pp. 253–267.

[110] SCHÖNER, G., AND SPENCER, J. *Dynamic thinking: A primer on dynamic field theory*. Oxford Series in Developmental Cognitive Neuroscience. Oxford University Press, New York, USA, 2015.

[111] SEARLE, J. R. Minds, brains, and programs. In *The Behavioral and Brain Sciences* (Cambridge, UK, 1980), vol. 3, Cambridge University Press, pp. 417–424.

[112] SICILIANO, B., AND KHATIB, O. *Springer Handbook of Robotics*, 2 ed. Springer, Berlin, Germany, 2016.

[113] SMITH, T. ROS.org | About ROS. http://www.ros.org/about-ros/. Accessed Jul 2019.

[114] SONG, B., TIAN, G., AND ZHOU, F. A comparison study on path smoothing algorithms for laser robot navigated mobile robot path planning in intelligent space. In *Journal of Information & Computational Science* (Thailand, Dec 2010), vol. 7, Binary Information Press, pp. 2943–2950.

[115] STEWART, D. A Platform with Six Degrees of Freedom. In *Proceedings of the Institution of Mechanical Engineers* (London, UK, Jun 1965), vol. 180, Institution of Mechanical Engineers, pp. 371–386.

[116] STOVOLD, J. *Distributed Cognition as the Basis for Adaptation and Homeostasis in Robots*. PhD thesis, University of York, York, UK, 2016.

[117] SWATMAN, R. Video: Watch 1,069 dancing robots break world record. http://www.guinnessworldrecords.com/news/2017/8/video-watch-1-069-dancing-robots-break-world-record-487675, 2017. Accessed Aug 2018.

[118] TAIPALUS, T. *ActionPool: A Novel Dynamic Task Scheduling Method for Service Robots*. PhD thesis, Aalto University School of Science and Technology, Helsinki, Finnland, Nov 2010.

[119] TAIPALUS, T., AND HALME, A. An action pool architecture for multi-tasking service robots with interdependent resources. In *International Symposium on Computational Intelligence in Robotics and Automation* (New York, NY, USA, Dec 2009), CIRA, IEEE, pp. 228–233.

[120] TIMMIS, J., AND TYRRELL, A. On Homeostasis in Collective Robotic Systems. In *Artificial Immune Systems* (Berlin, Germany, 2010), E. Hart, C. McEwan, J. Timmis, and A. Hone, Eds., Springer, pp. 307–309.

[121] TOWLE, B. A., AND NICOLESCU, M. N. An Auction Behavior-based Robotic Architecture for Service Robotics. In *Intelligent Service Robotics* (Berlin, Germany, Jul 2014), vol. 7, Springer, pp. 157–174.

[122] UPADHYAY, S., AND RATNOO, A. Continuous-Curvature Path Planning With Obstacle Avoidance Using Four Parameter Logistic Curves. In *Robotics and Automation Letters* (New York, NY, USA, Jul 2016), vol. 1, IEEE, pp. 609–616.

[123] VARGAS, P., MOIOLI, R., VON ZUBEN, F., AND HUSBANDS, P. Homeostasis and evolution together dealing with novelties and managing disruptions. In *International Journal of Intelligent Computing and Cybernetics* (Bingley, UK, Aug 2009), vol. 2, Emerald Publishing Limited, pp. 435–454.

[124] VARGAS, P. A., MOIOLI, R. C., DE CASTRO, L. N., TIMMIS, J., NEAL, M., AND VON ZUBEN, F. J. Artificial Homeostatic System: A Novel Approach. In *Advances in Artificial Life* (Berlin, Germany, Sep 2005), M. S. Capcarrère, A. A. Freitas, P. J. Bentley, C. G. Johnson, and J. Timmis, Eds., Springer, pp. 754–764.

[125] VIG, L., AND ADAMS, J. Multi-robot coalition formation. In *IEEE Transactions on Robotics* (New York, NY, USA, Sep 2006), vol. 22, IEEE, pp. 637–649.

[126] WEIR, M., BUCK, A., AND LEWIS, J. POTBUG: A Mind's Eye Approach to Providing BUG-Like Guarantees for Adaptive Obstacle Navigation Using Dynamic Potential Fields.

In *From Animals to Animats* (Berlin, Germany, 2006), vol. 9 of *SAB*, Springer, pp. 239–250.

[127] WIENER, N. *Cybernetics: Or Control and Communication in the Animal and the Machine*, 2 ed. MIT Press, Cambridge, MA, USA, May 1961.

[128] WILLIAMS, H. Homeostatic plasticity improves continuous-time recurrent neural networks as a behavioural substrate. In *International Symposium on Adaptive Motion in Animals and Machines* (Ilmenau, Germany, Jan 2005), vol. 3 of *AMAM*, ISLE.

[129] WOLINSKI, D., AND LIN, M. Generalized WarpDriver: Unified Collision Avoidance for Multi-Robot Systems in Arbitrarily Complex Environments. In *Robotics: Science and Systems* (Pittsburgh, PA, USA, Jun 2018), H. Kress-Gazit, S. Srinivasa, T. Howard, and N. Atanasov, Eds., Carnegie Mellon University.

[130] YEN, J., AND PFLUGER, N. A Fuzzy Logic Based Robot Navigation System. In *Proceedings of the AAAI Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots* (Palo Alto, CA, USA, Jan 1992), AAAI Press, pp. 195–199.

[131] YEN, J., AND PFLUGER, N. Fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. In *Transactions on Systems, Man, and Cybernetics* (New York, NY, USA, Jun 1995), vol. 25, IEEE, pp. 971–978.

[132] YOGESWARAN, M., AND S.G., P. An Extensive Review of Research in Swarm Robotics. In *Proceedings of the World Congress on Nature and Biologically Inspired Computing* (New York, NY, USA, Jan 2010), NABIC, IEEE, pp. 140–145.

[133] YONG, J.-H., AND CHENG, F. Geometric Hermite Curves with Minimum Strain Energy. In *Computer Aided Geometric Design* (Amsterdam, Netherlands, Mar 2004), vol. 21, Elsevier Science Publishers B. V., pp. 281–301.

[134] YOON, D.-Y., OH, S.-R., PARK, G.-T., AND YOU, B.-J. A biologically inspired homeostatic motion controller for autonomous mobile robots. In *International Conference on Robotics and Automation* (New York, NY, USA, Sep 2003), ICRA, IEEE, pp. 3158–3163.

[135] ZHANG, Y., AND PARKER, L. IQ-ASyMTRe: Forming executable coalitions for tightly coupled multirobot tasks. In *IEEE Transactions on Robotics* (New York, NY, USA, Jan 2013), vol. 29, IEEE, pp. 400–416.