<table>
<thead>
<tr>
<th>Title</th>
<th>A market framework for collaborative robot exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>O’Beirne, Declan</td>
</tr>
<tr>
<td>Publication Date</td>
<td>2014-04-07</td>
</tr>
<tr>
<td>Item record</td>
<td><a href="http://hdl.handle.net/10379/4423">http://hdl.handle.net/10379/4423</a></td>
</tr>
</tbody>
</table>

Downloaded 2020-10-03T22:23:00Z

Some rights reserved. For more information, please see the item record link above.
A Market Framework for Collaborative Robot Exploration

Declan O’Beirne
Supervisor: Dr. Michael Schukat
College of Engineering and Informatics
National University of Ireland, Galway

April 7, 2014
Abstract

Robot exploration involves navigation through and generating a map of a novel environment. The generated map may facilitate further robot operations or may be of use to humans. Multi-robot exploration involves the deployment of a group of robots to explore a novel environment with the intention that using multiple robots will provide benefits in terms of efficiency, accuracy, robustness or a combination of these. Additionally, using multiple robots may allow more basic, low-cost hardware to be used, where the increased volume of sensor data and appropriate mechanisms for cooperation can result in improved performance over a single expensive robot.

This research considers the problem of using miniature low-cost robots for real-world tasks that have up until now only been possible using robots with accurate sensors and odometry and ample power and processing capacity. The issues of low quality sensor data, inaccurate odometry, low processing capacity and limited power are approached through the adoption of a flexible, distributed control system. The task approached in this research is that of collaborative exploration, although the system has been designed to be extensible to further tasks.

A scalable behaviour-based control system is presented that is lightweight enough to be deployed on embedded processors while still enabling robots to react effectively to changes in the environment and to other robots. A market framework is used to support efficient task selection for robots as well as the formation of coalitions to achieve close coordination between robots where profitable. Experiments with physical robots demonstrate the viable solutions developed to the many technical issues that low-cost robots raise while further experiments in simulation extrapolate these finding to greater
number of robots in more challenging environments.
Acknowledgements

During this work I have received a great deal of support from members of the Department of Information Technology at NUI Galway. Foremost, Dr. Michael Schukat has been an inspiring and inexhaustibly helpful influence over the course of this research. While providing direction, inspiration and making his expertise available whenever requested, his affable character and infinite patience have made this work a great learning experience and an enjoyable journey.

Additionally, my fellow researchers at the Department of Information Technology at NUI Galway, aside from having become close friends, have provided moral support, technical guidance and feedback and at times accommodation throughout this process. My family too have been unwavering in the support and encouragement over the past years, while my girlfriend Laura has been an enormous help and relentlessly patient.

Thanks also go to the Irish Research Council for Science, Engineering and Technology with whose funding this work was carried out.
Contents

1 Introduction 17
  1.1 Motivation ............................................. 17
  1.2 Open Research Questions ............................... 18
  1.3 Hypotheses and Contributions ........................... 20
  1.4 Research Methodology ................................... 21
  1.5 Thesis Overview ........................................ 23

2 Autonomous Low-Cost Robot Control 25
  2.1 Related Work ........................................... 26
    2.1.1 Motivational Systems for Autonomous Robots ....... 26
    2.1.2 Behaviour-Based Mobile Robot Control Architectures 28
    2.1.3 Map Representation .................................. 30
    2.1.4 Representing Uncertainty in Robot-Localisation and
          Mapping ................................................ 34
  2.2 Control System for Low-Cost Robots .................... 37
    2.2.1 Profit-Based Control ................................ 37
    2.2.2 Extensible Behaviour Architecture .................. 40
    2.2.3 Control System Architecture ........................ 47
    2.2.4 Map Representation for Low-Cost Robots .......... 58
  2.3 Summary of Contributions ................................ 65

3 Distributed Control in Collaborative Clusters of Au-
   tonomous Robots ........................................... 67
  3.1 Related Work ........................................... 67
    3.1.1 Coordination in Multi-Robot Teams ................ 67
3.1.2 Market-Based Multi-Robot Control ........................................ 74
3.2 Collaborative Framework Design ............................................. 78
  3.2.1 Market-Based Collaborative Multi-Robot Exploration ............... 78
  3.2.2 Combining Map Data from Multiple Autonomous Robots ........... 83
3.3 Summary of Contributions ................................................... 87

4 Autonomous Exploration in Low-Cost Robots ................................. 89
  4.1 Related Work ................................................................. 90
    4.1.1 Distributed Control in Multi-Robot Exploration Teams .......... 90
    4.1.2 Visual Robot Mapping ................................................. 92
  4.2 Low-Cost Robot Exploration ............................................... 95
    4.2.1 Accurate Mobile Robot Simulation .................................. 95
    4.2.2 Distributed Computation for a Team of Low-Cost Robots ....... 96
    4.2.3 Visual Obstacle Detection .......................................... 97
    4.2.4 Local-Area Exploration ............................................ 109
    4.2.5 Wide-Area Exploration ............................................ 119
  4.3 Exploration Results ...................................................... 134
    4.3.1 The Effect of Market Values on Robot Behaviour ................ 135
    4.3.2 Exploration with Multiple Robots .................................. 142
    4.3.3 Experiments with Physical Robots ................................ 144
  4.4 Summary of Contributions ............................................... 147

5 Distributed Collaborative Exploration for Low-Cost Robots 151
  5.1 Related Work ................................................................. 152
    5.1.1 Cooperative Localisation in Multi-Robot Teams .................. 152
  5.2 Behaviour-Based Collaborative Exploration for Low-Cost Robots .... 155
    5.2.1 Relative Localisation with Low-Cost Visual Sensors ........... 155
    5.2.2 Supervision ......................................................... 161
    5.2.3 Loop Closing ....................................................... 167
  5.3 Collaborative Exploration Results ...................................... 172
    5.3.1 Experiments with Physical Robots ................................ 184
  5.4 Summary of Contributions ............................................... 184
6 Conclusion

6.1 A Lightweight, Extensible Behaviour-Based Robot Control Architecture ........................................ 187
6.2 Exploration Technique for Low-Cost Robots ............... 188
6.3 A Market-Framework for Collaborative Exploration with Autonomous Robots ................................. 189
6.4 Publications ............................................................... 190

A Low-Cost Robot Hardware ............................................. 191

B Robot Calibration .......................................................... 193

C Simulation of Robot Motion Error .................................. 197
## List of Figures

2.1 Behaviour interface ........................................... 43  
2.2 Behaviour management ....................................... 45  
2.3 Robot control loop ........................................... 49  
2.4 Software architecture ........................................ 53  
2.5 Example low-cost robot platform ............................ 62  
2.6 Exploration in simulated environment ...................... 66  
3.1 Coalition Arbiter structure .................................. 84  
3.2 Allocating coalitions ......................................... 85  
4.1 Timings from experiments on physical robots using distributed computation ........................................... 98  
4.2 Training images for obstacle detection ..................... 99  
4.3 Camera setup on physical robots ............................. 100  
4.4 Image sensitivity to lighting conditions .................... 102  
4.5 Descriptions of lighting conditions for obstacle detection ........................................... 104  
4.6 Descriptions of obstacle appearances ...................... 107  
4.7 Occupancy calculation in test images ...................... 109  
4.8 Local-Area Exploration state diagram ....................... 115  
4.9 Calculating an exact path given constrained movements ........................................... 116  
4.10 Exact planning for 2 moves .................................. 118  
4.11 Accurate path planning ....................................... 120  
4.12 Exploration targets mapped using accurate path planning ........................................... 121  
4.13 Exploration targets mapped using basic path planning ........................................... 122  
4.14 Proportion of local maps explored ............................ 127  
4.15 Room ratios relative to area explored ....................... 128
5.15 Maps generated with 4 robots with no experiment timeout . . 180
5.16 Maps generated with 8 robots with no experiment timeout . . 181
5.17 Errors in adjusted map scans after loop-closing . . . . . . . 182
5.18 Reduction in errors in adjusted map scans . . . . . . . . . . 183
5.19 Loop closing with 2 physical robots . . . . . . . . . . . . . 185

B.1 Example raw calibration data . . . . . . . . . . . . . . . . . . 194
B.2 Example calibration results for a physical robot . . . . . . . 196
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Map storage requirements</td>
<td>63</td>
</tr>
<tr>
<td>4.1</td>
<td>Categorising occupancy training images</td>
<td>105</td>
</tr>
<tr>
<td>4.2</td>
<td>Experiment results with varying maximum permitted error</td>
<td>137</td>
</tr>
<tr>
<td>4.3</td>
<td>Experiment results with varying battery power</td>
<td>138</td>
</tr>
<tr>
<td>4.4</td>
<td>Experiment results with varying wide-area exploration profit</td>
<td>140</td>
</tr>
<tr>
<td>4.5</td>
<td>Experiment results with varying n robots</td>
<td>142</td>
</tr>
<tr>
<td>4.6</td>
<td>Experiment results per robot with varying n robots</td>
<td>144</td>
</tr>
<tr>
<td>5.1</td>
<td>Robot appearance classification results</td>
<td>156</td>
</tr>
<tr>
<td>5.2</td>
<td>Map adjustment error</td>
<td>170</td>
</tr>
<tr>
<td>5.3</td>
<td>Maps generated using independent exploration</td>
<td>172</td>
</tr>
<tr>
<td>5.4</td>
<td>Maps generated using collaborative exploration limited by iterations</td>
<td>173</td>
</tr>
<tr>
<td>5.5</td>
<td>Maps generated using collaborative exploration limited by localisation error</td>
<td>178</td>
</tr>
<tr>
<td>5.6</td>
<td>Loop-closes in collaborative exploration experiments</td>
<td>179</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The field of autonomous mobile robotics has the potential to play important roles in a variety of real-world applications. While some roles have already seen limited application of such robots in the real world, for example autonomous robot vacuum cleaners, broad adoption of mobile robots for carrying out suitable tasks is yet to be achieved. With population aging in many countries in the developed world, service robots may in the near future be of vital importance for performing tasks for which it would otherwise be prohibitively expensive to employ a human.

Autonomous mobile robots are well suited in particular to repetitive tasks where appropriate behaviour can be modelled using a simple, robust control schema, while robots can potentially make use of accurate sensor data to perform a task more efficiently or accurately than a human, for example calculating an optimum path to follow while cleaning a floor or harvesting crops. Robots can be deployed in dangerous or inhospitable environments. Bomb disposal robots have surely been directly responsible for saving the lives of humans that would otherwise have had to fill this role. As such robots require a human operator, developing autonomous behaviour would increase the number of tasks for which such robots could be deployed. In other environments it would simply be impossible to send a human, so robots provide
a hitherto unavailable capability. For example, using humans to partake in prolonged planetary exploration experiments would be more expensive than sending robots by many orders of magnitude.

1.2 Open Research Questions

As a team of small inexpensive robots may be more cost-effective than a single expensive robot, research in the area of autonomous mobile robotics has increasingly focused on deployment of such teams and on the use of distributed control of multi-robot teams as opposed to control based on a rigid deliberative schema from a single agent. Teams of such robots have been demonstrated in simulations or in extremely simplified environments when performing basic tasks. Such experiments frequently do not take account of the practical problems that the use of low-cost robots raise.

A necessary capability of robots operating in real-world environments is to be able to determine where they are in relation to objects of interest in the environment. An area that has received much attention is that of simultaneous localisation and mapping (SLAM), where robots build a map of a novel environment and simultaneously perform localisation within the environment. Teams of low-cost autonomous mobile robots will not achieve widespread adoption in real world applications until they can perform this task effectively. For this to happen a number of important issues must first be addressed:

- Robots should be able to act autonomously — Approaches such as those by Rekleitis et al. [1] and Grabowski and Khoela [2] specifically consider the task of using cooperative localisation to counteract the unbounded growth of localisation error when mapping an environment. However, they demand a high level of coordination between robots. An approach developed by Rothermich et al. [3] allows more distributed control, with coordination arising by having robots competing to perform tasks. However, this approach considers the deployment of robots equipped with omni-directional range sensors where mapping is carried
out implicitly by observing free space between robots, meaning that robots are forced to move in formation and thus cannot navigate to arbitrary points as they see fit or switch between performing different tasks as they see fit.

Other cooperative localisation approaches such as the one described by Mourikis and Roumeliotis [4] do not use robots as stationary robots but rather continuously exchange relative location measurements between a group of robots. It has been shown though that localisation accuracy in such approaches is dependent on the accuracy of robots' self-localisation [5]. Thus, the benefit is minimal for low-cost robots with inaccurate dead-reckoning, while the requirement of omni-directional range sensors means that cost per robot cannot be kept to a minimum.

- Robots must be designed to be adaptable to changes in the environment and should ideally be capable of performing different tasks — Robot control mechanisms should therefore avoid a monolithic or deliberative design. It should instead be modular in nature and be adaptable, allowing a robot to employ different reasoning when the environment or the mission demands it.

Behaviour-based robot control systems have been shown to provide an effective, robust and scalable solution in this context [6]. Additionally, a number of systems have been developed to support distributed coordination between such robots, for example the Hoplites architecture developed by Kalra et al. [7], where robots communicate their plans with each other in order to avoid duplication of effort. While reactive behaviour-based approaches are well suited to inherently distributed tasks or tasks which can be accomplished through emergent complex behaviour from basic control schemas, other tasks may only be carried out effectively through explicit cooperation. The cooperative exploration system presented by Zlot et al. [8], for example, uses an auction to distribute targets between robots to ensure efficient operation without requiring a deliberative planner. However, this approach still requires that robots must obey the decisions arrived at by the auc-
tioneer, meaning that robots are not free to choose alternative tasks and act in a truly autonomous way. A system for cooperation between robots demonstrated by Vig et al. [9] uses a market framework to form coalitions between robots in order to perform tasks that require close coordination, while still allowing robots to act autonomously as they can determine when to cooperate. However, the system requires that tasks are defined by a central agent which holds auctions to allocate tasks to robots. In order for a team of robots to operate completely autonomously, the robots themselves should be able to determine what tasks to perform and when to cooperate.

- The limited capabilities of such robots in terms of storage, computational power, communication bandwidth and battery power suggests that a system controlling such robots should be designed from the ground up with these constraints in mind — While SLAM techniques using robots with accurate self-localisation, laser-range sensors and high processing power have been demonstrated to great effect [10], such techniques are not possible on low-cost robots built using off-the-shelf hardware components and embedded processors

1.3 Hypotheses and Contributions

Given the open research questions relating to the deployment of teams of low-cost robots listed in the previous section, this research aims to examine a number of hypotheses:

- Low-cost robots can be used to perform complex tasks in challenging real-world environments. Cheap, off-the-shelf components can be used to perform a task as well as robots with, for example, expensive laser range finders.

- Collaboration between such robots can increase the capabilities of such robots in terms of efficiency, the quality of the output of the mission and the robustness of the team to failure.
The work presented in this thesis includes a number of contributions that verify that these hypotheses may be correct and are worthy of future research:

- Behaviour-based control architecture — Allows robots to behave autonomously while being deployable for multiple different tasks and react effectively to changes in the environment. The system is also sufficiently lightweight in its implementation so that it can be run on robots with small embedded processors.

- Exploration for robots with low-cost hardware components — Constraints imposed by low-cost off-the-shelf components include low-resolution, noise-prone sensor data, limited battery power and limited storage capabilities. To this end, an exploration and mapping approach was developed that works within the control architecture described in the previous point. This approach allows accurate calculation of actions required to carry out tasks where low-granularity actuators are in use. The approach can robustly calculate occupancy and detect objects using low-cost cameras, while the limited storage capabilities are addressed by allowing robots to share generated map data with an external agent.

- Market framework for autonomous robot collaboration — maintains high autonomy in robots, without any imposition that they must collaborate in any specific way, but rather allows robots to decide themselves when to collaborate based on the expected profitability.

1.4 Research Methodology

In order to verify the hypotheses listed in the previous section it was necessary to build a number of robots using low-cost off-the-shelf components. Details on the hardware components of the 3 robots and of the construction and calibration process and are provided in the appendix. While the problems approached in this work relate to arbitrarily large teams of robots, using a small number of physical robots meant that solutions to specific
real-world problems could be verified while application of the solutions to different environments and to larger numbers of robots could be carried out in simulation. The robots were built on small, differential drive wheeled or caterpillar track bases, typically about 20 cm in length. The robots were equipped with low-power ARM processors built-in with Bluetooth sensors in addition to low-resolution monocular cameras. It was found during experimentation that the high levels of noise meant that images taken at any great distance were unusable for detecting obstacles or classifying objects. The cameras were therefore mounted at an angle to the group, as illustrated in figure 4.3. This allowed that the entire image could be used for obstacle detection, while also meaning that robots could detect obstacles at close proximity.

The physical robots were first used to demonstrate the viability of running an advanced control system on a low-power processor to control motion and processing of sensor data. The robots were used to determine the odometry error that could be expected using such low-cost components. Additionally, sensor data captured by these robots allowed development of techniques to overcome the typical problems that low-cost modules are typically prone to, i.e. low resolution and high noise levels. The robots were then used for single-robot exploration and mapping. In order to develop advanced autonomous behaviours it was necessary to be able to debug robot code and run large numbers of experiments quickly. To this end an accurate simulation framework was developed. This framework was designed to be a thin layer between the robot code and a system to visualise activity and to simulate sensor data, meaning that the code run in simulation and on physical robots could be largely identical. Thus, code could be tested in simulation before being deployed on robots, while output from experiments on physical robots could be rerun in simulation to verify performance and debug problems encountered.

The simulation environment was used to develop the behaviour based robot control system, providing robots with more advanced reasoning capabilities, and to develop a collaboration framework. As the majority of code is shared between simulation and robots, the control system and collaboration
framework could be deployed onto physical robot to verify the results seen in simulation and to carry out exploration and mapping of real world environments. In order to allow robots to perform cooperative localisation, it was necessary to support detection of other robots in visual data and accurate relative localisation. So that high levels of accuracy could be achieved, the dimensions of each robot were measured and made known to the other robots. In addition, each robot was given a different appearance, allowing robots to robustly distinguish between different robots in visual sensor data. Physical exploration experiments were carried out in a flat, indoor, office-like environment. Maps were generated with and without employing collaboration to verify the increased accuracy that could be achieved.

1.5 Thesis Overview

The remainder of the thesis is divided into separate chapters for each of the main research areas that this work touches on. Each chapter contains a discussion of the state-of-the-art research in the corresponding area, followed by a description of the contributions made in this work.

Chapter 2 discusses control mechanisms for autonomous mobile robots, examining various approaches to robot decision-making and motivation. The profit-driven control architecture developed in this work is then described, along with the approaches to state estimation and map representation developed specifically for use with low-cost robots.

While still concerned with robot control, chapter 3 focuses on control in multi-robot teams. Particular attention is paid to the increasingly common approach of market-based multi-robot control. The market-based collaborative exploration framework presented in this work is then described. In addition, novel agent-based approaches to distributed coordination and map building are outlined in sections 3.2.1 and 3.2.2.

Chapter 4 deals with the task of robot exploration, i.e. exploring and generating a map of a novel environment. The state-of-the-art in robot exploration, in particular multi-robot exploration, is discussed. Thereupon, the appearance-based visual occupancy mapping technique presented in this
work is described, along with the training and testing systems that this en-
compasses. The behaviours which robots implement as part of the control
framework described in chapter 2 in order to carry out exploration are expli-
cated subsequently, including an explanation of a novel planning technique
for visual exploration, followed by a discussion of experimental results.

Chapter 5 then describes the state-of-the-art in robot exploration with
cooperative localisation. A visual relative localisation technique is presented,
along with a description of behaviours that allow robots to perform cooper-
ative localisation in an autonomous and efficient manner. Following this,
results from experiments using collaborative exploration are presented.
Chapter 2

Autonomous Low-Cost Robot Control

This chapter describes the control system put in place to facilitate the stated goal of using collaboration between multiple low-cost robots to improve the efficiency and capabilities of the robots. Robot motivation systems in the literature are discussed in section 2.1.1, along with a discussion of behaviour-based robot control frameworks in section 2.1.2. Map representation approaches are described in section 2.1.3, followed by state representation approaches in section 2.1.4.

Section 2.2 presents a control system for low-cost autonomous robots. A detailed description of the behaviour framework is provided in section 2.2.2, along with the reasoning behind design decisions made and an explanation of the important novel contributions that the system includes. The remaining sections in the chapter then describe the control architecture and software design 2.2.3, state representation 2.2.3 and map representation 2.1.3 systems that the framework incorporates, followed by a summary of contributions.
2.1 Related Work

2.1.1 Motivational Systems for Autonomous Robots

A robot displaying autonomous behaviour should, by definition [11], be able to behave independently of external control, i.e. such a robot should be empowered to determine what to do, not merely given instructions and left to implement them, or more rudimentary yet, given explicit actions to perform. Indeed, for an autonomous robot to truly emulate human behaviour, it should be able to learn in an open-ended manner throughout its lifetime [12]. To this end the field of developmental robotics works towards a system where an intrinsic development program develops mental capabilities by interacting with its environment [13]. Such ends are beyond the scope of the research presented here, but they are nonetheless goals that any autonomous system should strive towards. However, even within such research in the field of developmental robotics, in order to initiate the developmental progress, the robot must be imbued with an inherent value system.

Huang and Weng present a system modelling punishment, reward and novelty, which are relevant value systems that they have identified in humans [14], and have demonstrated working systems applied to navigation and object recognition [15]. An approach put forward by Lehman et al. [16] foregoes providing robots with explicit instructions or even objectives, but instead allows robots to come up with their own objectives by motivating them to develop novel behaviours.

In related work by Oudeyer et al., an intelligent adaptive curiosity system [12] is presented, based on the observations that development in humans proceeds incrementally and is autonomous and active. In this system, robots choose actions that will bring the greatest gain in terms of learning, using a reinforcement learning system. The robots are therefore ultimately motivated to maximise a metric describing their learning progress.

In the field of robot exploration, there has been much related work in employing a logical base for motivating robot action. Parker et al. use emotions to improve the effectiveness of multiple robots working in a group
For example, acquiescence will allow robots to avoid greedily attempting to achieve a goal while impatience prompts robots to undertake a goal when it deems that it can achieve it more effectively than another robot.

An approach put forward by Nourbakhsh et al. [18] uses moods to control robot behaviour, with a fuzzy state model used to transition between moods, while similarly, Arkin et al. use an ethological model of insect behaviour as inspiration for a control system. Here, a deliberative module coordinates between fear, hunger and greed. Stoytchev et al. [19] extend this work by combining deliberative, reactive and motivational layers in the robot control system, where robots can act under the influence of curiosity, frustration, homesickness and anger.

As the focus on the research presented here is on the use of multiple robots in concert, it is useful to define a logical model within which multiple robots can reason. A popular approach in the literature is to model the value that agents assign to actions or resources as a currency [20][21][22][23], thus when a group of agents work to perform such actions and create and consume such resources, this leads to the definition of an economy as put forward by Dias et al. [24], i.e. a population of agents producing an output. Dias et al. compare robot or agent control systems with centralised control, i.e. where robots are controlled from an external agent, to socialist or communist economies. Such systems may display a number of characteristics that will lead to inefficiencies; the next chapter on multi-robot control contains a full discussion on this topic.

An alternative to such centralised systems is a free market system in which an economy operates. In this domain, a free market may be defined as a collection of software agents interacting through a price system [21]. Here, robots are free to choose actions that will return the greatest gain. Implementing a market framework will involve specifying a direct mapping between the actions of a robot and the value of the output. Similarly, any physical or conceptual resource that a robot has and can expend should be modelled as having an accompanying monetary value. Here, a resource could correspond to time, location, processing capacity, communication bandwidth, battery power, data storage, etc. [21].
In related work, Dias et al. [24] describe an architecture for translating tasks to a market domain. Task outcomes, for example the map data generated from the task of exploring terrain, are mapped to revenue values, while the operations required to perform such tasks, for example moving, sensing, computing, communicating, are mapped to cost values. The robots are thus provided with the goal of maximising profit, defined as revenue − cost. Kraus et al. [22] presented a comparable approach where agents act in order to increase the overall effectiveness of a group, instead of acting for their own personal benefit. In contrast, Rosenschein and Zlotkin [23] describe self-motivated agents as those that operate without a notion of global utility but that work only for their own gain, to which end they are provided with concept of personal utility which they try to maximise. Such purely self-motivated agents will therefore only act benevolently if it is in their best interest to do so. In work by Zlot et al., a robot exploration system that used a market framework to control targets explored was shown to display a dramatic increase in exploration efficiency over another comparable approach [8], where efficiency is measured as area mapped relative to distance travelled.

2.1.2 Behaviour-Based Mobile Robot Control Architectures

Centralised, or deliberative, robot control is an approach where a single control module is used to determine actions. This module typically implements a control loop where sensory information is gathered, the robot's model of the world is updated based on this information, the robot then determines the optimal action to take given its state and the state of the environment and finally the robot executes the action [25][26][27][28][29][30][31].

Advantages of this approach over those discussed later in this section are that complex goals and constraints can be considered when determining robot actions, and multiple goals can be coordinated at the same time [32][33]. However, this approach requires an accurate model of the environment and considerable processing capabilities to analyse the environment in the context
of the robot's stated goals. Additionally, this approach may not be flexible
to changes in the environment or in other entities that will impinge on the
robot's effectiveness. For example, a change in the robot's capabilities such as
an actuator, sensor or communication medium failing, or additional robots
acting in a competitive or cooperative manner being added to or removed
from the environment. Such eventualities would have to be foreseen within
the design of the control system in order for the robot to continue to operate
effectively, while the size and complexity of the system would likely impose
limits on the hardware on which it could be deployed.

The reactive or behaviour-based robot control schema was introduced by
Brooks et al. [34]. This approach foregoes a planning process but instead the
robot's actions are controlled exclusively by sensing and behaviour modules,
where each behaviour model can have access to sensory information and
information from other behaviours, and behaviours are triggered according
to a predefined schema.

Contention between behaviours is often managed by employing a priority
hierarchy [34] or by combining the desired actions as output from the different
behaviours, for example, by using a motor schema approach as introduced
by Arkin et al. [35]. Such systems are useful in supporting quick reaction
to changes in the environment, while not requiring a full model of the envi-
ronment to perform high level planning. Additionally, they typically require
less computation and are thus suited to low-cost robots.

Shortcomings of this approach include an inability to perform complica-
ted tasks due to the lack of a planning module and coordination between
robots, and possible inefficiency when compared to centrally coordinated
robots as each robot will only consider information from the environment in
its immediate vicinity.

Hybrid approaches aim to marry the benefits of deliberative control with
behaviour systems [36] [37]. A number of hybrid approaches in the literature
adopt an architecture where a high level planning or deliberation module
looks at an overall model of the environment and passes instructions to lower
level behaviour modules [38][39][40][41][42].

Other approaches make use of a state-hierarchy, with higher level layers
considering a more comprehensive view of the environment to determine high level goals and lower level layers considering smaller windows of the environment, or representations of reduced dimensionality, to consider more immediate goals [43][44].

An alternative approach is to use a deliberation layer to analyse the environment and generate a model of reduced dimensionality that can be used directly by a reactive layer [45][46].

A common configuration in these approaches is to have a deliberation layer to analyse the overall environment state, a task execution layer to carry out selected tasks and a reactive layer to mediate between these layers or determine additional tasks deemed necessary in response to the robot’s immediate environment [47].

2.1.3 Map Representation

There are various approaches to representing a map of a robot’s environment in the literature. This section will describe the most common and effective approaches, and clarify the approach used in the exploration system put forward in this work.

The occupancy mapping technique was introduced by Moravec and Elfes [48, 49, 50]. This approach represents an environment as a grid, where each cell corresponds to a square area of the environment. In the mapping approach presented in this thesis, for example, a scale of 2 centimeters by 2 centimeters is used. Each cell stores a probability value that indicates the likelihood of that area of the environment being either occupied by some static obstacle or free space. Thus for each cell \( c \), the map will record the probability of that cell being occupied \( p(c) \), with 0 indicating absolute certainty of the cell being not occupied, 1 indicating that the cell is definitely occupied, and 0.5 indicating that there is no prior knowledge about the occupancy of the cell. Frequently in the literature, as well as in the grid representation used in this research, there is no prior knowledge assumed about the environment.

The occupancy mapping algorithm seeks to calculate \( p(c) \) from a sequence
of observations $z_{1:t}$. The problem of updating a cell given observations $z_{1:t}$ from the poses $x_{1:t}$ can be written as

$$p(c|x_{1:t}, z_{1:t}) = \frac{p(z_t|c, x_{1:t}, z_{1:t-1}) \cdot p(c|x_{1:t}, z_{1:t-1})}{p(z_t|x_{1:t}, z_{1:t-1})} \tag{2.1}$$

If it is assumed that $z_t$ has no dependency on $z_{1:t-1}$, then by using Bayes’ rule this equation can be updated to give

$$p(c|x_{1:t}, z_{1:t}) = \left[ 1 + \frac{(1 - p(c|x_t, z_t)) \cdot p(c)}{(1 - p(c))} \cdot \frac{(1 - p(c|x_{1:t-1}, z_{1:t-1}))}{p(c|x_{1:t-1}, z_{1:t-1})} \right]^{-1} \tag{2.2}$$

If the prior probability of a cell being occupied is assumed to be 0.5, then $\frac{p(c)}{(1 - p(c))}$ can be cancelled out in this equation. It is then left to the model applied to the sensor to determine how an individual sensor reading should update the occupancy probability of a cell, i.e. how $p(c|x_t, z_t)$ should be calculated. Stachniss [51] provides a thorough derivation for this formula.

As grid maps do not model features, they can be used to model unknown or unstructured environments, and allow direct access to grid cells. A disadvantage, however, is the large memory requirement. Due to the high dimensionality of the representation, carrying out high level analysis is inefficient. Path planning can be achieved more efficiently if the grid were converted to a graph representation, such as a Voronoi graph [52].

An extension of this model, coverage maps, introduced by Stachniss et al [51], aims to avoid the discretisation errors inherent in occupancy maps. In situations where a grid cell is not parallel to an obstacle such as a straight wall in an indoor environment, the occupancy estimate will converge to one when measurements are combined from multiple perspectives. Coverage maps overcome this coarseness by representing for each cell a full posterior distribution of the occupancy estimate. This could theoretically include any form of distribution, such as a mixture of Gaussians, but Stachniss uses a histogram of all possible values. This approach offers accuracy benefits when noisy sensors which create a lot of false measurements are used, though the representation has a larger memory requirement, as richer data structures must be used to
Feature maps provide a more concise representation than grid maps as they consist of features extracted from sensor data. As a result they scale better to very large environments. Feature maps can also be adjusted more easily upon receiving new information and are thus suited to mapping techniques where the exact state of the map is determined over time. The act of localising a robot within a feature map is a more efficient one than within a grid map in terms of the reduced dimensionality of the data. This task reduces to determining the spatial relationship between the robot and one or more landmarks [53], and if the landmarks are indistinguishable, to locating a corresponding set of features in the map [54, 55].

Such features should be salient and readily detectable again. A limitation of feature-based representations is that they can only model a certain set of features. Additionally, as not all sensor data is utilised, accuracy may be compromised. In structured indoor environments features may be corners, doorways, walls, etc. A technique often seen in the literature is to fit planes to data in 3-D space based on assumptions of a structured indoor environment [56]. A number of authors avoid the restriction on the set of available features by using a learning mechanism to determine distinctive features from sensors data [57, 58, 59, 60, 61, 62].

Where visual sensors are employed, appearance-based representations can be created. Dellaert et al. [63] construct a mosaic of the appearance of the ceiling in an indoor environment using an intensity measurement. This imposes strict assumptions on the environment though. More flexible approaches compute a compact description of sensor data using, for example, neural networks [64] or PCA [65]. A shortcoming of such techniques is the susceptibility of global sensor information to degradation caused by varying lighting conditions, occlusion, etc. Local feature models avoid some of these vulnerabilities. Se et al. [66] use a feature detector based on a difference of Gaussians to detect features that are scale and rotation invariant [67]. Rocha et al. [68] implement a 3-D coverage map, where stereo sensor used as 3-D range sensors to construct a visually realistic map.

Mapping techniques that update map data in an incremental manner are
well suited to creating a map from a sequence of known poses. They do not maintain pose information for each sensor reading, thus the process of updating the map involves low storage and computation overheads. However, they are not readily applicable to backwards correction when the robot comes upon new information that informs it that the location of map data previously collected should be adjusted. A number of techniques have been put forward to deal with the problem of accumulated error in a robot’s pose. This will typically come at the cost of storage and computation overhead [69].

Smith et al. [70] introduced the concept of a stochastic map. This method allows uncertain spatial relationships between a robot and entities in an environment to be modelled as a network, and allows the positions of landmarks to be updated as new sensor data is obtained. A Kalman filter is used to maintain relationships, where the state vector includes the position of the robot along with the features in the map and a covariance matrix of covariances between features.

Lu and Milios [71] present the consistent pose estimation technique, where range sensor scans are matched against an a priori map to correct localisation errors. A number of approaches extract features from occupancy data such as lines, corners or histograms to simplify the task of matching map patches, but require that assumptions are made about the nature of the environment [69, 72, 73, 74]. Gutmann et al. [75] use local registration to correct localisation errors over short distances by analysing disparities between adjacent range scans, along with global correlation to compare the occupancy grid surrounding the current robot pose to possible matching map segments.

As the size and detail of a map increase, the complexity of maintaining a coherent map will become intractable. A number of approaches use a more abstract representation of a map, a topological map, to reduce the dimensionality of the problem, while also facilitating path finding and higher level behaviours. A topological map is a more concise description of a map at a higher level, where abstract concepts like path fragments or intersections between paths are represented [76]. The scale of the local maps are typically not pre-determined but dependent on the characteristics or complexity of the
terrain [77].

Duckett and Saffiotti [78] create a topological structure by joining overlapping sections between local occupancy maps. Lankenau et al. [79] build a topological map of connections between local metric maps by matching corners where paths intersect. Bosse et al. [77] model local areas as a set of landmark features, and adjacency between local areas is calculated by matching features. To deal with potential misalignment of map sections, Dudek et al. [80] build an exploration tree that maintains all possible models of the environment, where local areas are modelled as nodes and a graph maintains possible connections between nodes. Kuipers et al. [76] use a hybrid approach where a typical metric map is used over smaller-scale areas and supports robot localisation, local path planning and obstacle avoidance. A topological map over the entire environment is then generated over the course of the experiment. An extension to this approach by Pierce and Kuipers [81] uses a learning technique to generate a sensorimotor system to facilitate more efficient path planning and calculating optimum exploration strategies.

2.1.4 Representing Uncertainty in Robot-Localisation and Mapping

In order for a mobile robot to move through an environment, it must be able to calculate its location, its destination pose, and the operations required to get there. In a real-world environment though, a robot will typically have at least slightly erroneous information for these three points. For example, the robot’s 2-D location relative to a specified global reference frame may be off, as may the destination pose. Also, the actions that the robot predicts will bring it there may also be slightly off due to the limitations of floating point accuracy, to inaccuracies in the robot actuators, or due to slippage or unexpected artifacts in the environment. For a robot to operate effectively, it should therefore have a mechanism for estimating the extent of all such errors.

Seminal papers in the literature used odometry alone to estimate robot location [82, 83]. Odometry involves using sensors that are apart from the
mechanism moving the robot to measure distances moved. For mobile robots, this typically involves measuring wheel rotations, although for robot platforms where cost, weight and complexity must be kept to an absolute minimum, such mechanical sensors may not be appropriate.

Thus, even if the distance that a robot has actually moved does not exactly match the intended distance, the odometry sensors will be able to maintain a more accurate estimate of the robot location. Odometry measurements will still be prone to two sources of systematic error: there will always be a degree of error in the calibration of the actuators, e.g. the radius of a wheel being measured, and there will always be an error associated with the actual measurement of the actuator, i.e. the granularity of the measurement will be finite. Additionally, odometry will be prone to errors from wheel slippage due to variance in the frictional properties of the surface being traversed. Therefore, if a mobile robot moves and estimates its location by odometry alone, the error in the estimate will grow continuously.

Thus, in order to restrict the unchecked growth of location estimate error proprioceptive measurements must be augmented with exteroceptive measurements, i.e. measurements of some external entity, which the robot will be able to incorporate into its location estimate. Various papers in the literature discuss the use of artificial landmarks [84, 85, 86] or GPS measurements [87, 88] to this end.

Much current research makes use of an observation made in a seminal paper by Smith and Cheeseman [89] that errors in robot position and in map data are often correlated. Thus the map that the robot has generated could be used for localisation, i.e. simultaneous localisation and mapping (SLAM).

Approaches to combine estimates from different measurements include Markov frameworks, originally introduced by Fox et al. [90], or maximum likelihood techniques [91]. A large number of approaches use some form of an extended Kalman filter (EKF) [89]. A Kalman filter uses a series of measurements over time to estimate an unknown value, such that the accuracy of the estimate is improved over that of a single measurement. Updating a Kalman filter involves making an estimate of the state of a system and of the associated uncertainty. When measurements of the state are taken, the estimate
is updated using a weighted average approach. Therefore, the estimates can be updated recursively, where at each step only the current measurements and the current estimates of the state are required. Kalman filters assume a linear relationship between variables in the system, and also that error has a normal distribution. Therefore, the state at time $k$ is estimated from the state transition model $F$ applied to the state at time $k-1$, plus the error or noise inherent in the state estimate step $Q$. The update step then estimates the state based on an observation $z$ and an observation model that is used to translate the actual state space to the measurement space, along with a model of the observation noise $R$[92] [93]. As Kalman filters assume a linear system, a nonlinear version, the extended Kalman filter, provides a more appropriate method for estimating location [89].

As Kalman filters demand that landmarks are recognised robustly, i.e. not ambiguous, they are frequently used in conjunction with particle filter techniques to track multiple hypotheses. A particle filter uses a sample-based representation of a probability density function to track a variable of interest over time. The immediate advantage of this approach is the natural representation of non-Gaussian and multi-modal distribution functions. The filter iteratively undergoes an update step that changes the value of a variable being tracked and an observation step that imposes constraints on the variable according to a model of the sensor data and of the variable being tracked. Particles with weight below a defined threshold are then periodically eliminated and new particles added in a resampling phase. Specifically for the case of tracking the pose, $X$, or position and orientation of a mobile robot, the value being tracked is $X^k = \begin{bmatrix} x^k & y^k & \theta^k \end{bmatrix}^T$ at each time step $k$. This is modelled as a set of $M$ particles ($S^k = \begin{bmatrix} X^k_j & w^k_j \end{bmatrix} : j = 1...M$), where each particle $j$ has an associated weight that defines its effect on the final estimate [94].

Extensions to these approaches include using a reduced set of landmarks for the update step [95], such as Compressed EKF [96], using a sparse information matrix instead of a covariance matrix to maintain robot and landmark positions [97, 98], or making use of the observation that landmarks are inde-
dependent given known robot poses, FastSLAM \[99, 100, 101, 102, 103\], or in more recent work FastSLAM 2.0 \[10\].

In such approaches the environment is typically represented as an evidence grid \[90, 104\], or as a joint state vector of landmark positions \[105, 106, 107\]. Popular approaches include using features of the environment, e.g. walls or corners identified in map data \[108, 103, 75\] or, when using visual sensors, salient features extracted from images \[109, 110, 111, 112, 113, 114, 115, 116, 117, 118\].

2.2 Control System for Low-Cost Robots

This section presents a control system for low-cost robots. The control system incorporates a profit-based motivation framework, allowing robots to operate in a self-interested or team-interested manner, depending on the behaviours implemented. A behaviour plug-in framework is built on top of this, allowing robot logic to be implemented in a lightweight, modular fashion. The modular design extends to all aspects of robot functionality, allowing the system to be adapted to robots with different sensors, actuators or state representation approaches.

2.2.1 Profit-Based Control

In the control system presented in this work, a robot’s decision making process is founded on the motivation to maximise its personal profit. Income, i.e. that which is received in return for goods and services \[11\], is modelled as a virtual currency. In practical terms, goods equate to data or information pertaining to the environment or some other entity, while services equates to manipulating the environment or objects therein, or performing certain actions that may or may not result in the gathering of raw data or information, for example, navigating along a specified path or maintaining a desired pose. The requirement that a robot receive income in order to act necessitates the existence of an additional agent that will remunerate it. Therefore the system is designed to incorporate one or more agents with which a robot
can communicate and trade.

It is intended that a framework built on these premises will have the following advantages:

- **Accurate modelling of profit and costs** — the control system may be designed to accurately match the goals of the user by adjusting the value associated with resources, e.g. a more time-critical objective could be reflected by placing a greater monetary value on tasks that are completed more quickly, and by associating a greater cost with time, while an objective that places greater emphasis on accuracy may associate a lesser monetary value with tasks completed or data obtained with higher levels of noise or error.

- **Extensibility to different tasks** — the extension of a robot control system to incorporate different tasks may be made easier when a market framework is used. By associating a quantitative value with the tasks, robots will be able to switch to new tasks when appropriate by simply evaluating the profit.

- **Flexibility to changes in the environment or to objectives** — a framework may be designed to reflect changes in the environment through mechanisms that control the values associated with tasks or resources, mirroring how an abundance of a commodity in a free market will typically lower the price associated.

As the actual transfer of revenue may incur an overhead in terms of time and communication bandwidth, the system supports either real time trading or allocation of revenue at specific points during a mission or at its conclusion. Thus, it follows that this system assumes complete trust between agents; if a robot carries out a task for an agent but does not receive immediate payment, it will assume that it will receive full remuneration at a later point in time and continue to perform additional tasks. It is considered beyond the scope of this research to consider less than complete honesty between agents, as this topic has received much attention over recent years [119]. A trader agent is initially equipped with revenue according to the importance placed
on the associated objective. For example in the robot exploration system implemented as part of this work, an agent that trades revenue for a map of the environment will be initially setup with sufficient revenue to pay for the size and quality of map data that the objectives of a mission dictate.

The amount of revenue paid for carrying out a certain task or doing a certain amount of work must either be a constant value known to all agents, or agreed upon by the trader agent and other agents. When assessing the profitability of performing different tasks, a robot can consider gross profit \( p_g \) as the quantity of revenue it will receive. As mentioned in the previous section on modelling robot motivation and as fully explained in subsequent chapters, the architecture put forward in this work involves trading of services and revenue between robots. Thus, for certain tasks a robot will trade some of the gross revenue that it receives from a trader agent with another robot in return for performing a task or communicating data or information. Therefore net profit \( p_n \) for such tasks will be calculated as \( p_g - s \), where \( s \) is expenditure, an example of which may be the proportion of revenue that a robot must attribute to a partner robot that it is using as an artificial landmark to improve the accuracy of the estimated position of map data it has collected.

An essential capability of the collaboration framework presented in this work is the ability of robots to undertake long-term, complex tasks alongside more short-term, reactive, straight-forward tasks. Thus to be able to operate efficiently over the duration of a deployment, the robot should be able to accurately compare the profitability of disparate tasks. Here the profitability associated with undertaking a task is termed its utility \( u \). The utility associated with a task is calculated as a ratio of net profit \( p_n \) to resources expended \( r \), \( p_n/r \).

Any resource that a robot is initially in possession of may be finite and exhaustible, for example time or battery charge. It would therefore be inaccurate for the control system to evaluate tasks by calculating \( p_n - r \), as this would lead to a robot selecting tasks that take a longer time, but may potentially result in lower \( \sum p_n \) at the end of a deployment than if multiple shorter tasks were selected.

An example based on the cooperative exploration system described in
later chapters in this work is the comparison of the close_loop behaviour to the local_area_exploration behaviour. If it is assumed that each iteration of the robot control loop involves a cost in terms of time of \( c_t \), and in terms of battery charge \( c_b \), then selecting local_area_exploration for a suitable target might involve the expenditure of resources \( 5(c_t + c_b) \), and result in a profit of \( 15r \) units of revenue (ignoring other resources for now). While the close_loop behaviour may consume \( 50(c_t + c_b) \) in resources, with a profit of \( 100r \). Thus when comparing \( p_n - r \), in such situations a robot's task selection will always be skewed in favour of longer tasks.

2.2.2 Extensible Behaviour Architecture

As stated in the Introduction, one of the aims of this research is to provide a framework that would allow low-cost robots to employ collaboration to achieve goals that would otherwise not be possible when operating independently. A system that achieves this goal would need to satisfy the following inherent requirements:

- It must allow robots to act effectively in response to changes in the state of the environment, of the group of robots, or of the robot itself.

- It must allow different robots to perform different tasks and act as effectively as possible in response to each task.

- It should allow robots to act in collaboration with other robots when this would be more effective.

- It must be robust to failures in one or more robots.

Given the constrained computational power of each robot, an intuitive characteristic of such a framework is that robots would operate following a simple control system, but by facilitating cooperation or collaboration between robots more advanced capabilities would emerge.

The ability of multiple agents to switch readily between roles means that the system will be robust to individual agents failing, i.e. if a robot that fails
was carrying out a task using a simple, light-weight behaviour, then others in the system that are empowered to adopt this behaviour can readily take over the work.

It therefore follows that a reactive control system would be suitable for performing many of the tasks required of such a robot: collision avoidance, navigation around obstacles, moving to position to observe target, moving to nearby unexplored terrain, etc.

However, a limitation often incurred by low-cost robots is short sensor range. For instance, the robots employed to execute the framework described in this work are equipped with low quality visual sensors that can only dependably detect obstacles or free space within a range of two meters.

Consequently, relying on a reactive system alone to determine actions will quickly lead to inefficient behaviour, illustrated by the following examples:

- Considering only the robot’s immediate vicinity when selecting an exploration target may lead to local maxima, with the robot potentially adopting an inefficient path to explore an environment.

- Navigating around an obstacle by only considering the robot’s next move may lead to a robot navigating down dead ends or continuously navigating in a loop.

- Avoiding other robots without any knowledge of the other robots’ state may require unnecessarily inefficient behaviour, for example constraining robots to always navigate to the left of other robots.

Therefore, a reactive system used to control such robots may be more effective if augmented with more complex planning techniques that consider a greater range of information about the environment.

However, given the computational and power constraints that come with using extremely low-cost robot hardware, such techniques as those employed in the work cited earlier in this section may either be unfeasible or take so much time to compute as to negatively affect the usefulness of the robots.

A solution presented in this work is to use high level behaviours or roles that certain robots adopt to carry out tasks that facilitate the implementation
of primary behaviours by other robots, meaning that the limited capabilities of the low-cost robots are offset by spreading work between the group.

Additionally, as outlined as one of the main contributions of this work, the inability of robots to robustly detect landmarks from sensor data can be offset by utilising communication between robots that can be cheaply implemented in terms of both hardware and computational overhead.

By employing a design comprising different behaviours that deal with information that corresponds to areas of the environment of varying magnitudes and at varying levels of abstraction, the system presented here allows robots to undertake different roles, with some focusing their resources on developing a model of the environment and performing analysis to extract useful information for other robots.

Then other robots can employ high level behaviours to carry out the primary goals of the group, and make use of the greater amount of environmental information that is made available to them. This allows the group of robots to carry out more complex tasks while concurrently employing low level behaviours that react to artifacts in the robot’s sensor data.

**Behaviour Plug-in Interface**

In designing a software architecture for the robot control system required within this framework a primary concern was that the architecture would not negate the advantages of using low-cost robots that could potentially be used in a large, heterogeneous group and employed in different environments, in different configurations and for different tasks.

The software architecture should therefore mirror the characteristics of a group of such robots, namely the software should be readily adaptable to different tasks and should be cheap to update and extend.

To this end a plug-in software architecture was designed. Behaviours implementing a specific interface can be plugged into the system, with a thin management layer controlling activation of the behaviours. It is vital that any behaviour-specific logic is kept out of the management layer. This would undermine the adaptability of the system, and will lead to a monolithic design
of poor cohesion that will over time become swamped with obsolete code. Instead, each behaviour module must only impinge on the control of the robot through a uniform set of functions, thus allowing the robot management software to operate agnostically to whatever behaviour the robot is currently implementing.

The base interface that behaviours must implement is presented in figure 2.1. When adding a new behaviour to the system, minimal changes to the robot control code and to the build system need to be made, with a helper module automatically scanning each behaviour at start up to take care of registering its functions as callbacks.

![Software interface implemented by robot behaviour modules.](image)

Figure 2.1: Software interface implemented by robot behaviour modules.

Adopting such a design offers numerous gains for the effectiveness and scalability of the system:

- The design of the system is simplified, promoting its readability and therefore its reusability for further projects.

- The logic contained in the management layer on top of the behaviours can be maintained at a minimal level, meaning that the system will retain the benefits of a reactive behaviour system over a deliberative system.

- By defining a coherent structure for designing behaviours, many complicated, low level algorithms can be shared between behaviours. Ad-
ditionally, code size and run time performance are further improved as
the management layer can be, for example, designed to use function
pointers when interacting with behaviours, thus avoiding branching in
the code.

- The system can be readily adjusted or extended due to the ease of
adding new behaviours.

**Behaviour Management for Behaviours of Disparate Complexity**

In the system presented in this work, a distinction is made between high level
behaviours and low level behaviours, namely that high level behaviours are
those that a robot is motivated to undertake by its desire for profit, while
low level behaviours are undertaken by necessity, i.e. the robot must defer
to such a behaviour when its ability to operate is imperilled.

Behaviour execution is managed by a behaviour control module, which
acts as a thin layer between the main robot loop and the behaviours that
define its actions. The functionality of this module is illustrated in the flow
chart that follows. The logic described therein fits within the overall robot
control logic as described later in this section.

At any time the robot employs only one high level behaviour. Initially,
this will be the **AVAILABLE** behaviour. At the start of each step the robot
evaluates its current state. Based on the robot’s current state and how
this is processed by the logic within the behaviour, the robot may have
determined that its current goal has been achieved successfully, that its goal
is unattainable and that the robot should revert to another behaviour, or that
the robot’s state requires that another behaviour be adopted immediately,
either temporarily or with the cessation of the current behaviour.

Immediate adoption of another behaviour may be necessary when, for
example, the robot is about to collide with another robot or an obstacle,
in which case **AVOID_COLLISION** should be adopted, or the robot’s path to its
current destination has become blocked, in which case the robot should adopt
**CALCULATE_COMPLEX_PATH**.

Given that the behaviour system presented here employs only one be-
robot.init
robot.updateState
robot.isFinished
robot.cleanUp

behaviourControl.act
currentBehaviour.checkState

failure

success
canSwitchBehaviour
canSwitchBehaviour

behaviourControl.evaluateBehaviours

behaviourControl.processBehaviour

requireOtherBehaviour

behaviourControl.implementBehaviour

behaviourControl.popStack

mostProfitableBehaviour.adopt
requireOtherBehaviour

behaviourControl.pushStack

Figure 2.2: Behaviour management within the robot control module.
haviour at a time, an important ability of the control module is to allow each behaviour to be designed to only deal with the task in which it is interested. For example, if it is determined that the robot should attempt to detect a landmark to close a loop in the map data that it has gathered, then it should adopt the `CLOSE_LOOP` behaviour, and this behaviour should be designed to consider how the landmark should be detected. However, if the robot is to operate effectively it must still react to events such as imminent collisions with another robot, or its path being blocked, or becoming physically stuck. As the system presented in this work employs high level behaviours to perform complex tasks it should be able to record the state of its current behaviour at any time, adopt another behaviour in reaction to events in the environment when absolutely necessary and then return to the original behaviour when the temporary behaviour has handled the event.

To this end a behaviour-stack based approach is implemented that maintains the state of each of the behaviours currently adopted. As stated earlier in this section, this system provides that a robot may only adopt a single high level behaviour at any time, however multiple low level behaviours may be adopted concurrently as their state-transition rules determine necessary. As each behaviour is adopted an object defining its state is pushed onto the behaviour-stack. Then when a robot quits a behaviour, either because it succeeded or failed, then the state is popped off the stack and either the previous behaviour resumed or, if the stack is empty, the `AVAILABLE` behaviour resumed. Alternatively, the state-transition rules may define that failure should cause the robot to quit all current behaviours and immediately adopt the `AVAILABLE` behaviour. This condition may be invoked when a major change to the robot’s state occurs, for example, when the experiment goals changes, the status of the robot’s cooperation with other robots changes, or the robot learns that the state of the environment as currently understood has changed dramatically.

Each behaviour includes a set of rules defining how transitions to other behaviours are handled. This approach further aids in keeping the management layer of the control system as thin as possible, thus ensuring that the system remains easy to expand and to update as tasks or environments re-
quire. For example, if the speed of carrying out tasks is valued above the desired success rate of tasks, then a behaviour's set of rules for handling the detection of an obstacle in its path may be adjusted to quit the current task earlier and move on to the next task.

The behaviour-stack approach presented here, although influenced by it, differs quite a lot from the approach developed by Kaminka and Frenkel [120] both in its design and in the problems it aims to solve. In the work presented here, instead of using social interaction behaviours to define protocols for task allocation, all interaction between behaviours, and indeed between robots as discussed in later sections, are modelled within the behaviour modules. In the robot team structure presented here, there are no robots or agents responsible for coordination, and thus there can be formal representation of how robots can synchronize and how behaviours should be implemented. In this respect, this work improves on the already flexible, distributed approach presented by Kaminka and Henkel, while also allowing, as discussed earlier, that behaviours can be updated or added to or removed from the system.

2.2.3 Control System Architecture

The top-level robot control loop is kept as lightweight as possible. As the original contributions of this work lie in the behaviour modules, and in the implementation of sensor processing, map representation and cooperation between robots, it was intended that the top-level layer would have minimal impact on the operation of these modules, while also meaning that either the software or concepts could be integrated into other projects as easily as possible. The control algorithm, as described in figure 2.3, follows the typical sense-model-plan-act procedure [121][122]. Here, the model step involves processing data available to the robots and updating its representation of the environment, while the plan step involves determining the most suitable behaviour and its implementation. For the simulated and situated robots employed in this research, the data available to a robot includes sensor data and data obtained from other robots. Thus, processing of data involves the steps as illustrated in the accompanying schematic.
With respect to the task of robot mapping, processing of sensor data should enable raw data to be converted to information about the environment that can be incorporated into a map. However, as cooperation is highlighted as a core feature of the work presented here, it is intuitively necessary that robots will perform some form of synchronisation to ensure that map data communicated between robots can be useful to each robot. Therefore, before a robot updates its representation of the environment, it will first check the status of the environment as shared with it by other robots. As described in the next chapter on multi-robot control, an agent or robot is selected to aggregate map data from multiple robots, meaning that synchronising between $n$ robots requires $n$ connections instead of $n^2$. Upon obtaining an up-to-date version of the shared representation of the environment, the robot may then incorporate its recent map data. Additionally, as the map aggregator agent employed in this framework performs analysis on map data to generate various models that may be of use to the other robots, the robot should also update these models to ensure that reasoning will not be carried out based on stale information.

**Control System Software Architecture**

This section will describe a number of well known robot frameworks presented by various robotics research groups, describing the interesting features and the pros and cons of each, and then explain the framework developed as part of this work, along with a clarification of the reasoning behind the major design decisions.

A popular open robotics framework, the Orocos system (Open Robot Control Software) [123] used a CORBA-centric architecture, where different modules were developed to fill certain roles, such as path planning or kinematics, and each module runs on a different process. An advantage of using CORBA is that each module’s interface is language-independent, thus modules for high level behaviour modules could potentially be written in script, while more computationally intensive, low level functional modules could be written in native code. An obvious disadvantage is the fact that CORBA
Figure 2.3: Sensing and modelling within the control module.
has long been superseded by newer more lightweight and user-friendly frameworks. Another project, Orca [124], was developed from Orocos and replaced CORBA with ICE (Internet Communications Engine). While an improvement over CORBA’s API, this still leads to a more complicated development process compared to systems that use newer technologies such as XML-RPC or SOAP, although these do incur a performance overhead, and thus are not suited to very low-cost robots.

Another popular framework, the Player/Stage project [125], supported TCP connections between robots or sensors using standard Unix-like packets, where multiple client connections could be made to a server running on a robot, while robot code could be run in multi-robot simulations using the same framework.

A more recent framework is the ROS system (Robot Operating System) [126]. This again makes use of modules running on different processes, as it is aimed at more powerful service-robots. The ICE interface between modules is replaced with peer-to-peer communication, where a broker is used to set up connections. XML-RPC is used to transfer instructions and data structures between modules. Each module is modelled as a node. A robot architecture can then be configured by defining the interactions between nodes in XML. As the framework is designed to be usable on a variety of different hardware platforms and as the integration between modules is only quite loosely defined, the framework is not particularly suited to development aimed specifically at low-cost robots. Also, the use of RPC and multiple processes will incur a sizeable performance overhead on low-cost, low power processors, many of which are designed with a single core.

The Fawkes framework [127] is more light-weight, but less general than the aforementioned frameworks. It supports the development of modules as DLLs, which can be integrated at run-time. Thus, it incurs less overhead than approaches based on a higher level framework, at the cost of not being able to define a language-independent interface between modules, or having the option of using scripting languages for higher level modules, e.g. AI or behaviour modules that might be more suited to a quicker turn-around time when developing. However, it is arguable that for a typical programmer
working in the area of robotics, the cost of having to incorporate multiple high level frameworks would outweigh the cost of having to write in native code.

An interesting feature of the Fawkes framework is the use of a blackboard module that facilitates communication between plug-in modules as well as communication with external processes or agents.

The CLARAty framework [128] shares parallels with Fawkes. Modules interact through well-defined interfaces in native code. Here abstract classes specify the functionality that a certain module should implement, while specializations of these classes allow the details of implementations on different robot hardware to be abstracted away from the robot logic, e.g. a navigation module will required a different implementation for a hardware platforms employing skid steering to one using two-wheel steering. The architecture is divided into a decision layer and a functional layer, allowing further abstraction of details away from the high level logic that controls robot actions.

In related work, the TASER framework [129] uses Java to allow classes encapsulating data or instructions, referred to as Roblets, to be sent between robots to allow distributed computation. This is a useful facility on robots with limited computational power — one that was developed in parallel with the architecture presented in this work.

For the cooperative exploration approach put forward in this work it was decided to write a custom framework for a number of reasons:

- Tight integration of map representation — as it was a primary goal of the system that robots could be deployed for a broad range of tasks, it was deemed necessary that robots would have visual light sensors, as this would aid in object recognition and generally extracting richer information from the environment. Also, due to cost, power, weight and code-size constraints on physical robot platforms, it was deemed that robots should be equipped with a single sensor only. Therefore, a new map representation approach that was suited to short range visual sensors and to the computational constraints on low-cost robots was required, and although the various open robot frameworks offered modules with useful functionality for mobile robots, such as path plan-
ning and kinematics, these did not allow for tight integration with the data structures required by this representation.

- Optimisation for low-cost platforms — in order to suit a range of robot platforms, from simple low-cost swarm robots to complex, powerful service robots, generic robot frameworks must include additional logic and unnecessary data. The framework put forward in this work is aimed specifically at low-cost robots, and thus is designed to minimise the code-size footprint and to execute optimally on low power, typically ARM-based, processors.

- Cooperation-centric — this framework was designed to facilitate tight integration of cooperation between robots within the various functional or behavioural modules, e.g. a robot’s map update step may incorporate operations to sync with other robots, or a task selection step may incorporate a coalition formation step. Additionally, although other frameworks support record and play back of sensor data such that experiments can be accurately rerun in simulation, the framework presented in this work goes further by also supporting serialisation of all communication between robots and agents.

- Separating logic and data — various other frameworks support a plug-in approach to different functional modules, but do not consider the fundamental design principle of separating logic and data. In this framework, data is modelled in a separate layer, below various functional and high level modules, meaning that the framework may be adapted to use different representations for maps or environment or robot state and that functional modules can be replaced or adapted to work with these.

The software architecture of the robot control system put forward in this work is described in the accompanying schematic illustrated in figure 2.4. The architecture includes control software for robots and for other agents that may be active in an experiment. Examples of other agents include
Figure 2.4: Organisation of software modules within the robot and agent control systems.
the map-aggregator agent and coalition-arbiter, which facilitate more effective cooperation between robots, while the distributed-computation agent provides the service of carrying out intensive computation tasks for robots. These agent types are discussed in full in subsequent chapters.

The robot control module is a relatively thin layer above the behaviour control module and modules that carry out other high level tasks. Map processing is responsible for taking visual sensor data, transforming it into map data and incorporating it into a map in a coherent manner, such that multiple robots can have a shared representation of the environment. The cooperative localisation module also relies on the lower-level sensor processing modules and gives robots the ability to identify each other, or to use other external landmarks for the purpose of localisation. Much of the logic controlling robot activity is encapsulated in the various behaviours which, as described in the previous section, implement a common interface and can be updated or have new behaviours added. The behaviour control module contains minimal logic for managing adoption of behaviours and transitions between behaviours, as the majority of the logic and rules controlling this are defined within the behaviours themselves.

A similarity with other frameworks is the division between high level modules that contain logic specific to a robot or agent and low level modules that perform well-defined, state-less tasks. For example, functionality to compress and decompress maps is used by both robots and map-aggregator agents. The collision detection and kinematics modules are useful to any agent that must verify if robot actions are viable given robot state information and environment information.

The separation of all data from modules responsible for logic or computation allows a clean architecture to be maintained. Maintaining state data and logic in the same module would result in a dependency chain between modules, while avoiding this means that modules can be more easily reconfigured. For example, there should be no logical dependency between a module that translates sensor data into map information, and a module that synchronises a robot’s map with a group of other robots or agents. Therefore, by abstracting environment info into a separate entity these modules
are free to access the same data. Also, by making any function that carries out reasoning to be state-less, i.e. all data that it requires is passed into the function, all modules can be more easily repurposed for use by different agents or in different modes of operation.

State Representation for Computationally Constrained Robots

The state of a robot at time $t$ may be represented by the vector $\text{pose} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, while the uncertainty may be represented by a probability distribution over these values. This distribution may not be available from a robot's sensors and may typically not be necessary to accomplish given tasks. The state is therefore commonly approximated as the first and second moments of this distribution, namely the mean $\bar{X}$ and the covariance matrix of this vector $C(X)$ [89].

In practice, the $\theta$ of even a low-cost robot suitable for swarm deployment can be estimated immediately with minimal error, as outlined in the appendix, while the error in estimation position from dead reckoning gives much greater errors. The robot's location may therefore be maintained as $\text{loc} = \begin{bmatrix} x \\ y \end{bmatrix}$.

Using dead reckoning, the robot's location at time $t+1$ may be calculated by combining the location from time $t$ with the movement at time $t$, $\text{move}_t = \begin{bmatrix} d \\ \theta \end{bmatrix}$. The location at time $t+1$ is thus given by

$$
\begin{bmatrix}
  x_{i+1} \\
  y_{i+1}
\end{bmatrix} =
\begin{bmatrix}
  x_i + d \cos \theta \\
  y_i + d \sin \theta
\end{bmatrix}
$$

(2.3)

It is assumed that error accumulated when undertaking a move is not related to the robot's start location. The uncertainty in the location estimate may therefore be updated by combining the covariance matrix of the robot's location coordinates $C_{\text{loc}} = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{yx} & C_{yy} \end{bmatrix}$ and the covariance matrix of the movement $C_{\text{move}} = \begin{bmatrix} C_{dd} & C_{d\theta} \\ C_{d\theta} & C_{\theta\theta} \end{bmatrix}$.

The updated covariance, $C_{t+1}$, is in the same space as $C_{\text{loc}}$, so the corre-
sponding Jacobian matrix, $J_{loc}$, is $I$. The change in location caused by the movement must be translated from $\begin{bmatrix} d & \theta \end{bmatrix}^T$ to $\begin{bmatrix} x & y \end{bmatrix}^T$, thus the Jacobian matrix is $J_{move} = \begin{bmatrix} \cos \theta & -d \sin \theta \\ \sin \theta & d \cos \theta \end{bmatrix}$. The robot's covariance at time $t + 1$ is thus

$$C_{t+1} = J_{loc}C_tC_{loc}^T + J_{move}C_{move}J_{move}^T$$  \hspace{1cm} (2.4)$$

or

$$C_{t+1} = C_t + J_{move}C_{move}J_{move}^T$$ \hspace{1cm} (2.5)$$

The physical robots used for experimental verification in this research are equipped with compasses for orientation measurement and dead reckoning for proprioceptive location estimation. As described in the appendix, compass modules can be obtained very cheaply and thus can be fitted to low-cost robots without compromising the overall unit price of a robot. For the purpose of location estimation it was decided that in order to evaluate low-cost or swarm robots, no additional hardware or sensors would be used to improve accuracy when this could be achieved through robot behaviour. Thus, a robot's position as it moves to carry out its tasks is estimated using dead reckoning alone. As will be described in later chapters, sensor data and the adoption of cooperative behaviours will also be used to improve these estimates. This research deals only with the situation where the initial positions of robots are known.

In the state estimation system presented in this work, simultaneous localisation and mapping is not implemented. Thus, as robots gather map data, they do not use this to update their position estimates. Additionally, when a robot perceives another robot it will not necessarily use this information to updates its estimate, only doing so when certain conditions are met as described in the detail throughout the rest of the thesis. The approach is therefore technically sub-optimal, as these extra sources of information would improve the estimate accuracy. However, for such low-cost or swarm robots as this research considers, the range of visual sensors used is in the range.
of one meter and of limited scope and quality. A typical domestic or industrial indoor environment will consist of regions of free space of much greater range than this. Therefore, at any time a robot's visual sensor data would only be able to be used to determine if an obstacle or free space is directly before the robot. In the case of a particle filter, for example, this means that the system will likely never converge to a narrow distribution, while the computation and memory resources required to maintain the distribution of particles would overwhelm the robot. With respect to making use of relative position estimates from other robots, a robot will only use an estimate from another robot if it is acting as an artificial landmark for the following reasons:

- The system does not impose any synchronisation on the robots, i.e. robots can move at whatever time intervals they want. Also, as extremely low-cost processors and visual sensors are used, it takes an inordinate amount of time relative to the control loop to process an image. Therefore, it is likely that any observed robot carrying out an active behaviour will be in motion, and that the exact position at the time of sensing would be very difficult to calculate.

- As the only form of proprioceptive state estimation considered here is acutely error-prone dead reckoning, it follows that when exploring, or carrying out another active behaviour, a robot will quickly accumulate a large error, which it may later try to correct through cooperation with other robots, and if a relative measurement is made between two exploring robots, it is likely that their position estimates will not align, thus requiring the robots to consider a multi-modal distribution to accurately model their position. Although an implementation of such a model would likely be possible on a very low-cost processor, it was decided that the simpler model of considering only proprioceptive measurements would lead to smaller code size and lower computation overheads, while allowing the research to concentrate on using explicit cooperation between robots to improve their position estimates.
2.2.4 Map Representation for Low-Cost Robots

When designing a representation for the map generated by a group of robots in this framework, a number of key issues had to be addressed:

- **Local sub-maps** — the limited memory resources on low-cost robots mean that building and using a full map of the environment on a single robot will not be scalable to larger environments. Also, as the sensors with which robots are equipped will typically have limited range, robots will only interact with their immediate environment. Therefore, it is appropriate that the framework allows robots to use local sub-maps, while the overall map of the environment is maintained elsewhere.

  Chong and Kleeman initially proposed the technique of using a set of local sub-maps that a robot generates as it traverses a terrain while relative positions between sub-maps are maintained. As described earlier, a good deal of research has considered useful strategies for building these connections between sub-maps such that localisation errors can be minimised and path planning or high level behaviours facilitated [77, 76, 78, 79, 80].

- **Grid-based maps** — a sizeable fraction of the literature on robot mapping deals with feature maps, where sensor data is analysed to obtain features and the map is comprised of a representation of spatial relationships between features. This requires additional computation on the robot however. Also, the benefit of being able to maintain map data as features such that they can be continuously updated and corrected will not be realised on low-cost robots. Poor quality sensors mean that detecting features will be difficult, while short sensor range means that even if landmark features can be detected. A robot will only be able to sense a small number of features at any time, so obtaining a uni-modal distribution from localisation is likely to be error-prone.

The sub-map techniques mentioned in the preceding point deal with map representations where features are extracted from sensor data, which is typically range data, and where the sub-map consists of spatial
relationships between these features, while the topological map on top of this represents logical connections between sub-maps. As discussed already, the use of low-cost or swarm robot platforms will typically not have the sensor range to be able to detect enough landmarks for localisation, or have the computational capacity or quality of sensor data to be able to robustly recognise features. Therefore, as the map is represented as an occupancy grid, a different approach is required to maintain a set of sub-maps.

Techniques that deal with sub-maps represented as occupancy grids include those presented by Ekvall et al. [130] where doorways and borders between rooms are detected and used as partitions between sub-maps, and by Thrun et al. [131] where Voronoi graphs are generated over local areas and critical points used to determine suitable positions for partitions.

- Distributed computation — the framework presented in this work is designed to be used with multiple low-cost robots working cooperatively. Therefore, it is necessary that robots are able to quickly and easily share maps with each other or submit map data to be integrated into a single map. Also, as multiple robots will be using the same map for path planning and for informing high level behaviours, it follows that the computationally expensive task of generating a high level or topological map should be shared between robots. Sheng et al. [132] present an approach where a team of exploring robots is divided into sub-networks of robots within communication range of each other. Each sub-network then builds an occupancy grid-based local map which is kept up-to-date on each robot by maintaining a map table containing a set of map updates, each with a corresponding robot id and sequence number.

To this end, the framework uses one robot from the group as a map aggregator agent, to which other robots submit sub-maps and from which robots can obtain map data for specific areas. To reduce the complexity and therefore the overhead of this process, sub-maps are aligned to
a grid, with all robots working off the same coordinate system. Areas corresponding in size to the terrain visible to a robot’s sensors are then aligned to a smaller scale grid within local maps.

A similar approach is employed by Zlot et al. [8]. In their technique, a quad-tree structure is used to represent map information, where areas with sufficient unexplored terrain, i.e. interesting areas, are recursively subdivided into four, with the lower limit for the area of leaf nodes corresponding to the robots’ sensor footprint. The approach in this framework follows the same basic premise but uses two levels of detail, corresponding to the WIDE_AREA_EXPLORATION and LOCAL_AREA_EXPLORATION behaviours. Thus, all robots can quickly access information on potential target areas to explore over the entire environment. As discussed in the chapter on collaborative exploration later in this work, complications arise from the strict adherence to a grid when map data is adjusted, but the computational cost of dealing with this issue is not prohibitive.

- Localisation error in exploring robots — Although the exploration framework presented here is designed to be adaptable for use with groups of robots of varying capabilities, a conscious decision was made to equip the robots with visual sensors. The process of translating monocular visual data to map occupancy data differs to that of translating sonar or laser range data. While range data may often be prone to extreme noise due to signals reflecting off walls, visual data is not prone to such error as it is not an active sensing mechanism. When an unambiguous reading is returned from a range sensor, it can be inferred with confidence that the reading corresponds to either free space or an obstacle. Visual data on the other hand is prone to classification errors, where shadows, patterns, artifacts that occur in the camera apparatus, etc. can be confused for obstacles.

Therefore, this framework employs a grid representation more suited to visual data than the occupancy grid model first put forward by Moravec and Elfes [48]. As described in the following section on visual
occupancy mapping, the robot vision method used here can distinguish between obstacles and free space in the test environment with high certainty. The error inherent in transforming from image space to map space is also extremely low, relative to the localisation errors related to dead reckoning in low-cost robots. Therefore, instead of encoding the probability of occupancy in each grid cell, the error in the estimate of the robot’s location from which the sensor data was captured is instead recorded, while occupancy is assumed to be sufficiently certain and stored as a binary value.

The robot platforms used to verify this framework, as pictured in figure 2.5, are made up of a Rogue robotics ATR base [133], a CMU Cam 2 monocular camera [134] and a Gumstix Waysmall Computer-on-module with an Intel/Marvell PXA255 processor [135]. The cameras are mounted at approximately 0.49 radians to the horizontal plane, meaning that on a flat surface the robot can view terrain between approximately 0.22 and 0.44 metres. The section of visible terrain is roughly square, giving an area of 0.04 metres squared, or 484 centimetres squared. As the image capture latency is excessively long, in the order of 5 seconds, the robot control loop must follow the sequence: undertake a move, stop to capture an image, process image, undertake next move, etc. In unexplored terrain, robot moves are limited to the sensor range, such that collisions are avoided, i.e. a robot will only attempt to move across terrain that it knows is free.

The map representation presented in this work was designed in consideration of the points listed above, and of the typical characteristics of low-cost robot hardware. Due to the modular design of the framework, it can easily be adapted to various robot platforms by swapping out the SensorProcessing.processSensorData function, while the map representation can be altered by updating the EnvironmentData module. All other modules can operate independently of the parameters of the map representation.

The occupancy grid cells correspond to a square area of length 0.02 metres. As detailed in the next chapter, this corresponds to a section of an image over which an obstacle can be robustly detected. The environment is
Figure 2.5: Example of low-cost robot platforms used in experiments.

divided into local sub-maps, 84 occupancy cells or 1.68 metres in length. This
was chosen as an area that a robot could map with minimal accumulation of
localisation error. If the local map were centred on an artificial landmark, for
example, a robot could cover the entire area of the map while still remaining
within visual contact of the landmark for localisation.

Local maps are organised in a grid over the environment, with an overlap
of 1/3. As explained below, robots use their local maps for obstacle avoid-
ance, so it is necessary that a robot will always be within the bounds of
its local map as it traverses the environment. An overlap of 56 centimetres
means that a robot can safely avoid obstacles at all times.

The robot control framework presented here dictates that a robot will
always maintain a local map centred at its current location. All process-
ing of map data is limited to the local map, i.e. when a robot submits
map data for sharing with other robots, it does so by taking its local
map, compressing it, and submitting it to a map aggregator agent. The
MapProcessingCore.compressLocalMap function carries out a simple run length
encoding on a local map and gives an average reduction from 7054 bytes
to 507.3 bytes, i.e. a reduction in size of 93%. Robots also use their local
maps for obstacle avoidance. As new map data is incorporated into the local map, the `RobotMapProcessing.updateNavMap` function updates the robot’s navigation map for obstacle avoidance and its higher level navigation grid for path planning.

Within each local map, exploration targets are aligned to a finer-grained grid. Exploration targets are square areas, 0.28 metres in length, which corresponds to the area visible by a robot’s sensors, and are thus useful when evaluating the `LOCAL_AREA_EXPLORATION` behaviour. As these areas align with the local map grid, analysis of exploration targets can be used to infer characteristics of the corresponding local maps.

Aside from the local map, all other environment data that a robot maintains is specific to the behaviour modules employed. The default `LOCAL_AREA_EXPLORATION` and `WIDE_AREA_EXPLORATION` behaviours used in the experiments described in this work demand that a robot maintain a grid of exploration targets corresponding to the local map, as well as a grid of local maps over the environment being explored. An important consequence of the robots themselves not maintaining a full representation of the environment as it is mapped is that the memory requirements on the robots will not grow over time.

In the test environment used to verify the exploration framework, the robots model an environment using a square grid of length 504 cells or 10.04 metres, i.e. covering an area of just over 100 metres squared. Thus, the memory required by each robot to maintain the environment is as depicted in table 2.1.

### Table 2.1: Storage required per robot to maintain a map of the environment.

<table>
<thead>
<tr>
<th>Map Element</th>
<th>Declaration</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local map</td>
<td><code>unsigned char[SQR(LOCAL_MAP_DIMS)]</code></td>
<td>7056</td>
</tr>
<tr>
<td>Local area exploration grid</td>
<td><code>unsigned char[SQR(LOCAL_MAP_DIMS / EXP_TARGET_DIMS)]</code></td>
<td>36</td>
</tr>
<tr>
<td>Wide area exploration grid</td>
<td><code>unsigned char[SQR((ENVIR_DIMS - LOCAL_MAP_DIMS) / (LOCAL_MAP_DIMS / 3) + 1)]</code></td>
<td>225</td>
</tr>
</tbody>
</table>

#### Description:
- **Local map**: Declaration size is calculated based on the square root of the local map’s dimensions.
- **Local area exploration grid**: Declaration size is calculated based on the square root of the local area exploration grid’s dimensions, divided by the exploration grid’s target dimensions.
- **Wide area exploration grid**: Declaration size is calculated based on the square root of the difference between the environment’s dimensions and the local map’s dimensions, divided by the local map’s dimensions, then increased by one.
Representing Localisation Error in Grid-Based Maps

The map grid representation presented in this work aims to combine the benefits of occupancy and stochastic maps as described earlier in this chapter, although it is aimed squarely at low-cost robots defined by short sensor range, poor self-localisation and poor computational and storage capabilities.

Within a local map, localisation error accumulated by a robot will be relatively small; over a single move the location drift would be in the order of 1 centimetre. Map data will therefore be useful for obstacle avoidance when planning movements, as a fast, integer-based, Bresenham algorithm [136] can be used to check paths. Using a grid representation also simplifies the process of updating the map. Occupancy calculated in the camera image space can be quickly transformed to map space, while updating a grid cell can be implemented as a simple inline function in MapProcessingCore.integrateGridCell.

Each cell is stored as an 8 bit unsigned integer and includes an occupancy flag indicating that the cell is either unmapped, occupied, unoccupied or unknown. The value 255 is reserved for cells for which occupancy cannot be determined, while the value 127 is initially set for all unmapped cells. This means that a robot's localisation can then be stored over the range 0 to 127.

In the experiments run as part of this work, a maximum localisation error is imposed by the map aggregator agent, i.e. any map data submitted for integration in the global map with a localisation error above this value would not be accepted, and the submitting robot would be returned no revenue for it. For example, in experiments that were run over a small area of the environment with multiple robots using cooperative localisation, a maximum standard deviation of \( \sigma = 0.2m \) was imposed. This value can be quickly scaled to 127 using integer arithmetic, giving a granularity of 0.00157m. Thus, occupied terrain is stored as 0..126 while unoccupied terrain is stored as 128..254.

Each robot maintains a list of sensor readings for the current local map until that map is submitted, at which time a robot obtains a new map, synchronised with other robots, from the map aggregator agent. Each sensor reading consists of map data from a camera image in compressed form along
with localisation data. Therefore if the robot’s position is updated, for example, due to the detection of an artificial landmark, then the location error for the sensor readings can be updated based on assumptions on how error is accumulated [75]. Also, certain behaviours may need to augment the map with artificial data when evaluating targets. A simple example being the inclusion of assumed map data from other robots on the same map. Thus, being able to redraw a map is a necessity as only valid map data should be submitted to the map aggregator for inclusion in the global shared map.

When a local map is incorporated into the robots’ shared map, the algorithm running on the map aggregator can determine the revenue that should be returned for map data of a certain associated error, and can compare conflicting map data gathered by different robots for the same locations. As described in the later chapters, the map aggregator implementation used in this work maintains a set of floating local maps, allowing for conflicts between maps to be resolved later [137].

2.3 Summary of Contributions

As described in section 2.2.3, a novel robot control framework incorporating a plug-in behaviour architecture is presented. The modular approach to designing behaviours with uniform interfaces and coherent data-centric software architecture mean the framework can be readily adapted and extended. The behaviour control framework within this architecture, as described in section 2.2.2, is based on a novel behaviour-stack approach. This allows a behaviour-based approach to incorporate long term, high level tasks along with short term reactive tasks, where such approaches are typically limited to reactive tasks only.

Section 2.1.3 describes a novel map representation framework, designed specifically for low-cost robots where sensor range is typically limited but large localisation errors are accumulated. The map representation imposes limited memory and computation overheads on the individual robots, while cooperation between robots is facilitated as the map records localisation data along with the occupancy grid.
(a) Robot in simulated environment as rendered by the Visualisation module. The position of the robot’s local map and current target are shown.

(b) Global map constructed from combining local maps from robots. The obstacles in the simulated environment are superimposed onto the map area.

Figure 2.6: Robot mapping a simulated environment. As described earlier in this section, the value in each cell denotes whether or not the cell is occupied, with values less than 127 representing occupied terrain and values greater representing free space. Grid cells with value 127, i.e. the main grey area of the map is unexplored terrain. Cells with value 0 are superimposed obstacles. Cells with values in range 128..254 are mapped free terrain. Cells with values in range 1..126 are mapped occupied terrain.
Chapter 3

Distributed Control in Collaborative Clusters of Autonomous Robots

This chapter discusses the problem of multi-robot control, i.e. coordinating a group of mobile robots to operate in an effective manner. The state-of-the-art in control architectures is presented in section 3.1.1. Section 3.1.2 focuses on market-based approaches, while section 3.2.1 describes a main contribution of this thesis — a market-based framework for collaborative multi-robot exploration. Section 3.2.2 discusses a novel approach to maintaining a map for a team of autonomous robots, followed by a summary of contributions.

3.1 Related Work

3.1.1 Coordination in Multi-Robot Teams

From the mid 90s onwards, research in the field of mobile robotics has increasingly moved from deployment of single robots to multiple robots. There are a number of factors that often make the deployment of multiple robots preferable over a single robot:

- By using multiple robots, a goal may be achieved more quickly by di-
viding it among the robots and executing in a parallel manner. Typical tasks approached in mobile robotics research lend themselves to parallel execution. For tasks that are discrete in nature, for example, retrieving a set number of objects from an environment, multiple robots can be easily coordinated to perform non-conflicting sub tasks. Where tasks are continuous in nature, for example, exploration of an unknown environment, the literature contains an array of approaches that divide the tasks into a set of sub tasks in an optimal manner [138].

- A team of robots, even if homogenous in terms of capability, can be arranged in an exponential number of ways [8]. Multiple robots may therefore offer greater flexibility over a single, more capable robot. A team of robots might also include robots with different capabilities that can be deployed for different jobs, or that will offer increased performance or efficiency by cooperating on tasks. This means that robots can be designed with less general capabilities, but with special capabilities for a specific task. Thus, by making use of multiple, less capable, but lower cost robots, a team of robot may be able to outperform a single, very expensive robot at a fraction of the price [139].

- A task might only be possible when tackled by multiple robots. For example a surveillance task in a complex environment. In this case, using multiple robots provide different viewpoints from which sensors data can be obtained. The physics of some tasks may be inherently suited to multiple robots, for example, moving or interacting with a heavy object.

- For tasks where robustness is paramount, for example, where robots are performing a safety-related task, then multiple robots can be deployed to provide redundancy, i.e. if one robot fails, another can take over its task.

In the early years of research in the area of multi-robot deployment, approaches typically adopted architectures with a centralised control mechanism [140, 141, 142].
For example, in the field of robot exploration, Latimer et al. [143] apply strict formation demands on robots. Robots are assigned to teams, which explore a region of the environment in formation, while the system controls splitting or joining of teams when obstacles or loops are encountered in the environment respectively.

In domains that require that robots maintain specific formations with each other, many systems employ a central coordination system. This may dictate explicitly the moves that each robot can carry out [144], or have a specific model which dictates the roles assigned to robots [1]. Other approaches may allow a robot to determine their movements, but limit robot behaviour and dictate that robots must move and communicate at specific times [145].

Spletzer and Taylor [146] demonstrated a box pushing technique, with the role of and tasks allocated to each robot explicitly controlled to achieve absolute optimal performance.

Where coordination of robots is controlled from a central agent, a number of issues must be addressed before deploying robots to optimise performance [8]. If it is assumed that the requirements of a mission are well defined, then these issues include:

- **Number and capabilities of robots** — while deploying a large number of robots may return benefits in terms of robustness and speed of completion of mission, the cost of adding robots to the team must be weighed against these benefits. The gain in efficiency from parallel execution may be estimated by considering the degree to which the overall task of the mission can be divided into sub-tasks that can be allocated to different robots, dependant on constraints between these sub-tasks. Additionally, if heterogeneous robots are available, then the benefits offered by each robot’s capabilities must be considered.

- **Plan mission** — the stated goal of the mission must be described in terms of a task that the robots can do, and a suitable plan for dividing this task into sub-tasks and executing these must be generated.

- **Allocate tasks to robots** — a robot task mapping should be devised that best fits the requirement of the mission. Individual tasks should
be assigned to single robots, while more complex tasks may need to be assigned to sub-group of robots.

- **Plan task execution** — once the overall mission has been decomposed and assigned to robot, it must be decided how the robot will actually achieve those tasks. The divide between making high-level decisions in a centralised manner while low-level details are dealt with by the robot itself must also be decided.

- **Schedule actions** — the specifics of the mission may required tight control over the execution of tasks by robots. Examples in the literature include dividing a group of robots in teams which move in turn [144, 94], or dictating that all robots must move at the same time [145].

Such systems with centralised control over a large number of agents will be limited by the complexity of the control system. These systems will realistically not be able to process all salient information available to all robots [24]. Achieving this would require all data to be communicated to the control system, along with information describing the state of each agent, with instructions then relayed to each agent. This would lead to potentially restrictive communication overheads, along with a computation bottleneck at the central control system, while requiring a complex design to handle communication and processing in parallel.

In addition, systems with centralised control will be inflexible to changes in the state of the environment. Such a system would be required to analyse the state of each agent and determine appropriate actions. Generating an optimal plan for \( n \) robots would presuppose that the environment is known and static, and that robots will carry out instructions as expected. Thus, to be operable under varying conditions, a centralised system would have to either be excessively complex, possibly displaying exponential complexity [24], and have at its disposable a comprehensive logical model representing a large number of eventualities, or it would have to be overly general and therefore risk being subject to inefficient operation. Additionally, this system would be more susceptible to failure, as a loss of communication with a robot
would render that robot unusable, while a malfunction in the central agent would disable the entire group.

To address these restrictions, research has increasingly turned to distributed approaches, where individual robots in a group act in an autonomous manner. When using multiple autonomous robots, it may be possible to accomplish a complex task using a simple control mechanism on each individual robot. The computational overheads for each robot become tractable, as each robot needs only to evaluate its own sensor data and determine how to act accordingly.

This may allow the system to adapt more quickly to changes in the environment or to the team of robots, for example, if a robot senses a dramatic change in its local environment it may be able to update its behaviour immediately, instead of having to wait for a central controller to evaluate the state of all robots and issue it instructions to do so.

As robotics research has increasingly focused on such distributed approaches, the problem of allocation of tasks or coordination of the robots has arisen as an important issue [138].

Vail and Veloso [147] employ such a task allocation approach where a set of roles are defined with associated priorities and robots are assigned to roles in order of utility, i.e. which robots can carry out a role at the lowest cost and therefore the highest profit.

A similar approach developed by Castelpietra et al. [148] employs a system where tasks are dynamically ordered in terms of priority, and each robot must broadcast information about its state. The highest priority tasks are then assigned to robots based on most suitable robot.

Such systems achieve efficient allocation of tasks to robots, and as they were developed in the robot soccer domain, it is fittingly reactive to a quickly changing environment. However, they still require that the priorities of tasks be dictated to robots, and don’t allow for autonomous decision making in robots at a high level.

Wiegel et al. [149] address some of these issues by developing a more sophisticated role assignment, where robots can choose to adopt roles, and agreements between robots must be arrived at in order to swap roles.
The ALLIANCE architecture developed by Parker et al. [150] presents an alternative approach, which has been shown to outperform greedy allocation of roles [138]. Here, robots have behaviours or moods that affect how they approach assuming roles. Acquiescence dictates that a robot will leave a task to another robot that can perform it more effectively, while impatience motivates robots to take over tasks when it computes that a failure may have occurred in another robot, thus making the system as a whole robust to failure. The allocation approach is computationally expensive though, as a model of the time for each robot to execute each task must be constructed, an \(NP\)-hard problem. A work-around presented by Parker is to use a learning mechanism to estimate utility and cost in a particular environment. While the computation load on robots is still high, the system achieves a reduced communication overhead as each robot can deduce the utility of other robots from their state.

Stroupe et al. [151] demonstrate a similarly distributed approach to coordination, where robots employ behaviour-based control, and each robot plans its actions by first assuming the next actions of other robots in the teams. This approach is advantageous in terms of its minimal communication and computation requirements on robots, although its limitation to only looking at the next actions means that it may avoid optimal long term solutions by focusing on local maxima.

Kalra et al. [7] present the Hoplites framework which addresses this issue by allowing robots to consider long terms plans instead of only immediate tasks. Each robot determines a provisional plan of tasks to carry out and broadcasts this to other robots. Upon receiving other robots’ plans, each robot can then update its own, and then broadcast and re-update iteratively. The system also allows tight coordination between robots, in a similar manner to the TraderBots approach presented by Dias et al [152]. Each robot can weigh up the benefit of adopting a coordinated plan with a number of other partner robots. The other robots are offered revenue in return for cooperating, and once an agreement has been made, must pay compensation if they wish to break the contract.
An approach presented by Chaimowicz et al. [153] similarly allows for reassignment of tasks between robots. In this work a hybrid automaton is used, where the finite automaton modelling different roles is continuously updated based on continuous variables sensed from the environment, and thus the robot’s task selection process is better informed.

In a comparison by Gerkey and Mataric [138], it was shown that online assignment algorithms, i.e. those that do not have information about all tasks beforehand and can react to the inclusion of new tasks on the fly, can achieve an allocation of tasks to robots that is 3-competitive compared to an optimal solution. Iterative approaches on the other hand can typically achieve at least a 2-competitive solution, although both the computational and communication complexities of these approaches for $n$ robots and $m$ tasks is typically $O(nm)$.

A number of approaches attempt to garner the advantages of centralised and distributed approaches. Stenz and Dias developed an approach where robots are controlled in a distributed manner based on a market architecture, that uses centralised planning in certain situations where an optimal solution can be calculated [154]. Koes et al. [155] avoid some of these problems by providing a heuristic solution that, while not optimal, can be followed if time does not permit the calculation of a new solution. To avoid a single point of failure, each robot is aware of the problem and robot group structure and has its own planning capabilities.

In related research, Kaminka and Frenkel [120] present an approach where a traditional control hierarchy incorporating a behaviour graph is used alongside a set of social interaction behaviours which are used for coordination between robots when it is determined that this may improve efficiency.

In a similarly approach by Vu et al. [156] a mechanism is presented for swapping the protocols for allocating tasks between robots based on the mission context.

Whereas the approaches discussed thus far support explicit communication between robots, or in some cases demand it, a number of approaches do not employ explicit cooperation between robots, but nevertheless enable the robots to act in an implicitly coordinated manner [157].
Steel's [158] presents an approach where basic dynamic systems are applied to the task of collecting samples in the area of planetary exploration with the aim of achieving efficient emergent functionality. Drogoul and Ferber [159] similarly demonstrate a foraging system emergent from a basic behaviour schema inspired by ants.

Koenig et al. [160] developed a multi robot exploration technique based on ant behaviour, in which markers are left at points to identify them as explored, mimicking chemical signals used by ants.

While the benefits of the varying degrees of cooperation between robots depend on the characteristics of the task at hand, or on the nature of the robots involved, and can often be quantified in terms of efficiency and accuracy there are also a number of abstract or non-quantifiable advantages to each. Enabling explicit communication between robots suggests that in time, given advances in robot vision and natural language processing, robots will also be able to interact with humans. In many domains, for example, service robots, this will be of obvious benefit. As research advances in terms of the complexity of the tasks that robots can carry out, for example, advancing from the basic mapping tasks typical in current research to more advanced manipulation of the environment in future, it is foreseeable that it will become less likely that robots will be able to infer all necessary information about the environment or about the state of other robots from sensing alone. Conversely, as robots capabilities advance towards mimicking those of humans, it is likely that robots will be required to infer more detailed information from sensor data alone, for example, being able to recognise human facial expressions when operating in a service robot role.

3.1.2 Market-Based Multi-Robot Control

An approach to multi-robot coordination that has recently become more popular is to model task allocation as a market-based economic system. As discussed in section 2.2.1, while market architectures often motivate robots to act in a self-interested manner, this typically will result in efficient operation for the team as a whole. The task of modelling a multi-robot coordination
problem as a market framework can be approached as creating a system that will model robot states, actions and goals and result in desirable behaviour for the team as a whole [8].

Employing a market framework allows efficient operation to emerge. While robots with different capabilities or acting in different roles may cooperate to achieve a goal, robots acting in the same role may compete to determine which is in the best position to carry out a task. Thus, using a market framework allows robots to trade tasks or revenue in order to cooperate and to bid for tasks in order to compete. Therefore, by adopting a market framework, a multi-robot team can be designed and developed to operate as effectively and accurately as possible, while the flexibility of the market should maximise efficiency by minimising cost [24].

Without having a central control system to impose a rigid structure or hierarchy on a team, a market framework may allow robots to self organise to achieve optimal performance. If robots are to form sub groups, or elect leaders amongst themselves, this will be done in a manner that will return the greatest profit and the least cost [24]. Whereas a central coordinator may assume a common cost base for all robots, a distributed market-based system will allow robots to continuously update their cost functions in response to changes in the environment.

Kraus et al. [22] presents a distributed system, i.e. with no central coordination, that allows agents to form groups in order to maximise benefit from their actions in carrying out a task allocated to them. The system presented deals with agents that act with the intention of increasing the overall effectiveness of the group, instead of acting for their own personal benefit.

Conversely, Rosenschein and Zlotin [23] describe a system of self-motivated agents as those that operate without a notion of global utility, but are provided with a definition of personal gain that the try to maximise for themselves. Therefore, the agents are purely self-motivated, and will only act benevolently if it is in their best interest to do so.

In such systems, a robot’s actions may affect on other members of the team. Where a conflict may arise between the interests of different robots,
a common method for arriving at a resolution is to hold an auction. For example, in the context of task allocation, if robots compete with each other for the right to carry out a task, then an auctioneer will compare bids from different robots, allocate the tasks as appropriate and determines the payment that each robot should make. In other cases which do not involve task allocations, but where the team of robots must be in agreement on a decision, each interested robot or agent may rank the different options and a winning option is then selected based on the highest ranking [161].

The role, in general, of an auction is for each item in question to be assigned to the agent that places the greatest value on it [23]. That is, if there are $I$ items and a subset of them $S_i$ are won at auction by agent $i$, which places a value of $v_i$ on that subset, then the auction should aim to maximise $\sum_i v_i(S_i)$. The following are popular auction frameworks, both in the real world and in the area of multi-agent systems [162, 161]:

- **English auction** — any bidder can enter a bit at any time, with the auction ending when no bidder wishes to advance on the current highest bid.

- **Dutch auction** — the price of the item up for auction starts instead at a high price, which is incrementally decreased. The auction can end at any time when any bidder agrees to the current price.

- **Japanese auction** — the item up for bid starts with an initial price of zero. The price is incrementally increased and bidders can opt out of the auction by leaving the room at any time. The auction ends when there is only one bidder remaining in the room, who must pay the final price.

- **First price sealed bid** — each bidder submits a single bid to the auctioneer, with the highest bid winning.

- **Second price sealed bid** — also known as a Vickery model, differs only in that the winning bidder instead pays the price of the next highest bid. Here it is assumed that bidders are more likely to bid their true
valuation of the item instead of simply out-bidding their rivals, thus improving auction efficiency.

A seminal work by Smith et al. [20] presented the Contract Net framework. This system aims to provide more optimal solutions than the greedy approaches discussed in the previous section by supporting negotiation between robots for the right to carry out tasks. Here, agents bid for each task announced, with the task allocated to the first agent with a satisfactory bid. Their framework allows for different specifications for determining a winning bid to be defined per task.

Golfarelli et al. [163] present an adaptation of this approach, but using a barter system for swapping tasks, applied in particular to the problem of path planning. More recent work by Amstutz et al. [164] apply a market framework specifically to the multi-robot coverage problem. Here, whenever a robot comes across a new task, i.e. a novel area of the environment, it holds an auction to determine the most suitable robot to perform that task.

Zlot et al. [8] present a market-based coordination framework, where sets of exploration targets are considered together. Robots bid for the right to explore a set of targets along a path, thus allowing potentially better performance than greedy algorithms that consider only the next task or target for each robot.

Dias et al. [152], in related work, present the TraderBots framework. Here, instead of trading tasks with each other, a revenue model is used. When applied to the travelling salesperson problem, the system allows an OpTrader auctioneer agent to derive a plan for the team, where robots offer to perform a task in return for revenue, typically requiring revenue greater than the estimated cost in carrying out the task, and the system aims to settle at an optimum solution in terms of the revenue required to have all tasks executed.

Kaminka et al. [120], as discussed in section 2.2.2, employs a behaviour state machine to control robot actions. Within this framework, social interactions are defined on a per-behaviour basis, i.e. if a robot is implementing a behaviour where a conflict over a task may arise with other robots implementing the same behaviour, then this subset of robots must hold an auction
to compare bids and determine which gets to perform the task.

Nanjath et al. [165] consider the problem of auctions in dynamic environments or where failure in physics robots is likely, e.g. for search-and-rescue tasks, by monitoring the completion of tasks by robots and holding repeated auctions to re-assign tasks.

Sheng et al. [132] present a multi robot exploration approach using a distributed bidding mechanism which allows robots with limited communication range to cooperate through the formation of sub-networks. Within each sub-network robots hold *bidding sessions* whenever a robot finds new frontier points, with robots submitting bids for the right to explore a frontier points based on a function incorporating cost, information gain and proximity to other robots.

Tovey et al. [166] present a framework for the automatic generation of bidding rules in an auction for task allocation for multi-robot exploration. Here, the value that a robot bids on an item equates to the estimated benefit that the system as a whole will enjoy if the robot is assigned that item. The system then determines optimal strategies for determining winning bids based on the nature of the objectives at hand.

Likhodedov et al. [167] present a framework for simulation auctions and a technique to search the parameter space of possible auctions in order to arrive at algorithms that approach an optimal output.

### 3.2 Collaborative Framework Design

#### 3.2.1 Market-Based Collaborative Multi-Robot Exploration

In the multi robot system put forward in this work, robots are capable of operating completely independently, with no communication or cooperation with other robots. They may still infer information about other robots from sensor data, but will not have access to the information that other robots have. However, when communication is possible, robots will be able to share information, make use of parallel execution of computational tasks and col-
laborate on tasks where required.

Given the limited sensor range of the robots considered in this work, it is difficult for robots to infer information about each other, so it is useful if it can be explicitly shared. If robots were to communicate with each other directly, this would result in each robot making \( n - 1 \) connections with the rest of the team to sync state information. This will either require a multi-threaded implementation on each robot to constantly listen for requests to sync from other robots, or will result in excessive latency for the entire team.

Therefore, in this framework a communication hub is used. This may be an external agent that is connected to the robots, or may be one of the robots chosen to fill this role instead of or alongside its usual tasks. In a similar manner to how a blackboard agent is used in multi-agent system to allow agents to iteratively collaborate on a problem [168], this approach allows robots to share information and iteratively contribute to the map being generated. The communication hub agent implemented in this work maintains an array of \( \text{RobotStateData} \) structures for the team. In addition, a set of corresponding data structures holding sync information informs robots when these structures have been updated. Using the same data structure on each robot to store its own state and to store the state of the team as a whole in the \( \text{RobotGroupData} \) structure, means that syncing can be done simply by using \( \text{memcpy} \) to copy data to and from packages sent over sockets.

As discussed in chapter 5, the exploration mechanism employed by robots allows robots to collaborate to more accurately map areas of the environment. In such instances, closer coordination between robots is required, as the collaboration demands that robots adhere to certain constraints on their behaviours and actions.

A suitable mechanism to facilitate this is to form short-lived coalitions between robots, where membership in a coalition will imply that robots agree to adopt specific behaviours or perform specific actions in order to carry out the task in question.

A number of approaches have been presented that use coalitions to model tight coordination between a subset of robots where this improves efficiency or is required to perform a task that might otherwise not be impossible.
Gerkey and Mataric [169] demonstrate a box-pushing technique, where specific requirements to participate in the coalition are specified, namely two pushers and one watcher, along with a strict formation that must be maintained, in order to successfully complete the task. The implementation makes use of the MURDOCH platform for broadcast communication to determine which robots should take part.

While Gerkey and Mataric’s approach allows efficient selection of robots to the coalition, it does not apply a revenue model to bidding, i.e. the robots are simply instructed to act based on the outcome of the auction, without any concept of why it is in their interest to do so.

Liu et al. [170] present a coalition formation system based on Ant Colony Optimisation (ACO) technique within Swarm Intelligence [171]. Within the framework, robots are modelled as ants. Each robot comes up with possible solutions to the coalition formation problem and provides a pheromone to indicate its satisfaction with a solution. Other ants review the solutions proposed and add pheromones if they too find them satisfactory.

While Liu’s approach allows a group of robots to arrive at an optimal solution in terms of the perceived utility of each robot, in practice it will demand heavy communication overheads, as it reduces to every robot applying a value to every solution.

Vig et al. [9] demonstrate a market-based coalition formation technique, the RACHNA system. Here, service-agents model the capabilities of a robot, while task-agents model may reside on another robot or on an external agent, and are responsible for acquiring the services of other robots, and a revenue, or wage, model is employed to represent the bids. The framework allows for various methods of allocation depending on the stated urgency of a task.

**Coalition Arbitration for Collaborative Exploration**

As it is desired that this mechanism supporting collaboration between robots does not affect the distributed nature of the system, it has been designed such that the formation of coalitions is instigated by robots themselves. Based on the estimated utility of different profit-centric behaviours, a robot may de-
termine that it can obtain more profit by carrying out a task that requires collaboration with one or more other robots than it can by working independently. Thus, the framework put forward in this work allows robots to create coalitions using a market framework.

The framework presented in this work shares a number of similarities with that presented by Vig et al., though it differs in a few implementation details. This system does not rely on a central coordinator to assign tasks to robots, but allows robots that instigate collaboration through the formation of a coalition to assign tasks to willing partner robots.

Coalition formation is instigated by one robot posting a proposal that one or more robots join it to collaborate on performing a task. The proposal must therefore contain information that will allow other robots to determine the potential profit that it might receive as well as what costs would be incurred. Also, similarly to the Contract Net protocol presented by Smith et al. [20], further details such as the expiration time and the capabilities required to bid may be specified.

Given a proposal for a coalition, other robots must post bids in order to join. A bid should model the robot's desire to take part in the coalition, which in turn should be based on the estimated utility.

As discussed in section 2.1.1, all robot actions in this system should be motivated either by necessity or by the desire for profit. Therefore, the different types of coalition proposals should all describe the manner in which revenue will be obtained. The types of proposals are as follows:

- The proposing robot may state that it will agree to pay revenue to a partner robot in return for a resource or for performing some task.

- The proposing robot may describe a task that it will perform in collaboration with a partner robot, for which the robots will share revenue received from a third party.

- The proposing robot may state that it will perform a task or adopt a behaviour that will facilitate a partner robot in carrying out another, presumably profitable, task, if the partner robot will agree to share the received revenue.
In the case of the *SUPERVISION* behaviour described in section 5.2.2, the proposing robot defines the details of the collaborative localisation that it will facilitate. Upon reading the proposal, other potential partners can estimate the $profit-cost$ ratio that they would enjoy if they were to join the coalition. If they determine that this will be more profitable than working alone or joining or forming another coalition, then they can post a bid declaring the share of revenue that they receive during the coalition that they are willing to allocate to the proposing robot.

Depending on the behaviour in question, potential partner robots may post bids based on different strategies, for example, aiming to just outbid other robots or simply aiming to bid as low as possible. In the case of the collaborative localisation coalitions employed in the experiments described in this work, in order to simplify the bidding process, robots are obliged to calculate a bid according to a fixed function. Therefore, each potential partner will only post a single bid.

In an effort to further cut down on the communication overheads on robots, the framework includes a Coalition Arbiter (CA) agent. Again, this agent may reside on one of the robots, or on an external agent that is connected to the robots; an approach similar to the *OpTrader* employed by the TraderBots system [152]. In practice, it may be practical to host the agent alongside the communication hub agent, such that robot data and coalition data can be stored in the one location.

Thus, instead of the proposing robot being requiring to act as auctioneer, it can furnish the Coalition Arbiter agent with instructions on how to determine a winning bid. The instructions may be a set of rules and/or a flag indicating what function to use. In the case of the collaborative localisation coalitions described in later chapters, a first price sealed bid auction is used, as it is assumed that robots will calculate bids in direct proportion to utility. The CA agent can then post information regarding the outcome of the auction instead of having to initiate contact with all parties involved.

The Coalition Arbiter agent accepts both *Proposal* and *Bid* structures from robots. Matching bids and proposals are combined to form potential coalition structures. The potential coalitions are added to a priority queue according
to the rules of the auction. In this case, a potential coalition will be added to the queue for allocation either when all robots have viewed the proposal, or when the stated expiry time has expired.

Once the expiry time on a proposal has been reached, i.e. the auction is over and no further bids may be accepted, the potential coalitions are added to a priority queue according to a fitness function.

Each robot is permitted to bid on multiple proposals, while each proposal may accept bids from multiple robots. If a pair of robots are allocated to a coalition, this will affect the potential profitability of the coalition to other robots. Therefore, to avoid robots bidding on inaccurate proposals, the coalition arbiter enforces that once a coalition has been formed, all other bid, proposals and potential coalitions involving the two robots in question should be purged.

So that robots are allocated to the most profitable coalitions, a mechanism must be in place for the comparison of potential coalitions. The approach implemented here uses a heuristic that considers the amount of map data that is likely to be generated, and the accuracy of that map data. For the coalitions used in this system, the amount of map data is estimated from the state of the supervision area in question. The grid of supervision areas maintained by the Map Aggregator agent, as explained in section 3.2.2, allows the proportion mapped, $m$, in the area specified to be read by all agents. The estimated accuracy of the map data is estimated based on the accuracy of the supervisor robot’s location estimate $e_s$. These values are weighted by coefficients $k_m$ and $k_s$ according to the requirements of the mission. The profitability of a potential coalition is thus estimated as $k_m m + k_s \left(1 - \frac{e_s}{e_{\text{max}}} \right)$, where $e_{\text{max}}$ is the maximum permitted localisation error permitted in the mission.

3.2.2 Combining Map Data from Multiple Autonomous Robots

While there are examples in the literature of techniques to combine map data from multiple robots in a coherent manner [172, 8], the use of low-cost
Figure 3.1: Structure of Coalition Arbiter agent which accepts proposals to form coalitions from robots along with bids on these proposals, and assigns robots to coalitions according to rules specified.
Figure 3.2: Pseudo-code for allocation of coalitions from proposals and bids. When creating a new coalition, all bids, proposals and other potential coalitions pertaining to the two robots in question must be cleared. Additionally, any existing coalitions that they had partaken in must be removed, with flags set to alert other affected robots of the change of state.
robots in this work requires a novel approach. As described in section 2.1.4, the limited storage and computational capabilities of low-cost robots impose a threshold on the size of the environment that can be modelled.

The sub-map approach developed in this work means that the robots can explore an arbitrarily large environment without an increase in memory requirements. As each robot works only with sub-map centred on its current location, it is necessary for the maps from different robots to be maintained at another location. This role is performed by a Map Aggregator agent, a software agent that accepts local maps from robots and incorporates them into a global map.

Modelling Map Data as a Resource

As explained in section 2.2.1, the robot control architecture presented in this work determines actions based on utility, which is calculated as the ratio of profit / cost. For behaviours focusing on exploration described in chapter 4, LOCAL_AREA_EXPLORATION and WIDE_AREA_EXPLORATION, profit equates to map revenue received from the Map Aggregator in return for map data. These behaviours estimate the revenue they will earn for mapping targets based on the area of terrain mapped and the certainty in the location of the map data.

The behaviour implementations in this work use the same equation to calculate revenue as on the Map Aggregator agent, so the utility of exploration targets can be calculated precisely. Here, revenue is calculated as $k_m n \left(1 - \frac{e}{e_{max}}\right)$, where $n$ is the number of occupancy grid cells accepted by the Map Aggregator in the submitted local map, $e$ is the error magnitude encoded in each grid cell, $e_{max}$ is the maximum permitted error magnitude, and $k_m$ is a map revenue coefficient defined for the mission. As described in section 4.2.4, error magnitude is a representation of localisation uncertainty, chosen in this implementation as the area of the error ellipse for $1\sigma$ corresponding to the covariance matrix describing the uncertainty in the robot’s location estimate. Specifically, error values are stated as $area/\pi$, allowing the user who sets up the experiment to more clearly visualise the expected
error.

For each local map submitted, the Map Aggregator decompresses the map and compares it to the current data for that cell in the local map grid. As described in section 3.2.2, the global map is maintained as a set of compressed local maps which the Map Aggregator can use to build the occupancy grid for a specified section. Once the current area has been built, the Map Aggregator can calculate the number of occupancy grid cells in the submitted map with lower uncertainty, for which it will calculate the appropriate revenue to return to the submitting robot.

Maintaining a Coherent Global Map

For each robot to accurately determine the most profitable exploration targets, its local map should be up-to-date, containing all terrain already mapped by other robots. Comparing its own map to the current map of the corresponding area on the Map Aggregator in each iteration of the robot control loop would place an excessive load on each robot in terms of computation and communication bandwidth. Therefore, synchronisation information for each robot is maintained on the Map Aggregator.

At the incorporation of each new local map, a data structure for each robot is updated to flag whether or not it should re-sync its local map. To minimise the computational load on the Map Aggregator (which may be implemented on one of the robots in the group) updating synchronisation data for each robot is carried out using a simple algorithm. For each robot, the Map Aggregator maintains a bounding box around each submitted map since that robot last synchronised. During each iteration of its control loop a robot can simply check the bounding box against its current local map and request that the Map Aggregator sends it an up-to-date map when appropriate.

3.3 Summary of Contributions

In this chapter, a novel approach to coordinating multiple autonomous robots using a market framework was presented in section 3.2.1. This system allows
robots to operate completely autonomously, without imposing any obligations on robots in terms of having to cooperate or having to carry out specific tasks. The Coalition Arbiter agent allows robots to cooperate without any external interference while reducing the communication overhead and delay involved in setting up coalitions.

The Map Aggregator agent described in section 3.2.2 is a novel approach to combining map data from multiple autonomous robots. The agent allows map data to be combined in a coherent manner, such that robots can share information about the environment without high communication overheads. In addition, the agent supports updating the map upon receiving new information from robots. The basis in a market-framework means that multiple such agents can be deployed along side a team of robots and that multiple tasks can be carried out in parallel, e.g. searching, box-pushing, etc., with the market dictating which robots perform which tasks.
Chapter 4

Autonomous Exploration in Low-Cost Robots

Although the framework presented in this work is designed to be applicable to any robotic tasks, the implementation presented here to validate the system focuses on exploration, i.e. traversing through an unknown environment and generating a map. In particular, the task of multi-robot exploration is addressed, where the coordination of robots must be addressed alongside the tasks involved in single-robot exploration. This work specifically deals with exploration of indoor environments. It is assumed that robots will be deployed over flat terrain, but no further assumptions are made about the structure of the environment. Robot positions are assumed known at the start of a mission, but throughout the mission no known artificial landmarks are assumed to be available for localisation.

The remainder of this chapter includes a description of the state-of-the-art in exploration and multi-robot exploration in section 4.1.1, followed by a discussion of the visual robot mapping techniques in section 4.1.2. The simulation framework developed in this work for creating and testing exploration techniques for low-cost robots is presented in section 4.2.1. A novel appearance-based visual mapping system developed for use with miniature robots with low-quality visual sensors in described in section 4.2.3. A description of the exploration behaviours developed as part of the plug-in ar-
chitecture is then provided, followed by experimental results in section 4.3, and a summary of contributions.

4.1 Related Work

4.1.1 Distributed Control in Multi-Robot Exploration Teams

A comprehensive discussion of state of the art research in multi-robot coordination is provided in section 3.1.1. This section discusses coordination approaches that deal specifically with the task of multi-robot exploration. Further approaches that utilise cooperative localisation between robots when exploring are covered in section 5.1.1.

A common approach in the literature, the frontier-exploration approach was introduced by Yamauchi et al. [173]. In this approach, robots share an occupancy grid representation of the environment. The frontier between explored and unexplored terrain is examined, and robots are directed to explore their nearest frontier point. Burgard et al. [174] put forward a related approach where robots consider the likelihood of a target having been explored by other robots. Additionally, the likely free space available at a point is estimated. Koenig et. al. [175] present a study based on a similar greedy exploration approach which demonstrates that the upper bounds of navigation cost to such targets is not excessive compared to an optimal approach.

Mataric et al. [176] describe a system using a potential field over an environment for multi-robot coverage to direct robots away from obstacles and towards unexplored areas. Latimer et al. [143] describe a system which performs analysis of the environment to determine its geometric shape, and divides this into sections which are then assigned to robot. This approach achieves high parallelism and thus efficient exploration, but demands that robots are coordinated from a central agent, and that the structure of the environment can be determined and analysed. An approach by Howard et al. [177] iteratively assigns robots to target frontier points in the environment as
they are discovered. The technique models robots as interconnected nodes in a network, where assignment of robots to targets is controlled by rules defined by the network structure, which provides that robots will not be blocked by each other when trying to get to targets. While providing results approaching a greedy algorithm with full environment knowledge, this technique does require close coordination between robots.

While such approaches are well suited to smaller numbers of robots in relatively static environment, their centralised nature impedes their adaptability to new tasks or environments and their robustness to failure in robots or in communication between robots. A number of approaches have considered distributed control in multi-robot exploration. Allowing robots greater autonomy will potentially result in a more robust, scalable approach which may respond more quickly to changes in the environment.

Payton et al. [178] use the concept of pheromones in guiding robot exploration. Where robots should avoid exploring the same terrain, a gas expansion model is used to influence robots, while a guided growth model is used to direct robot towards areas of interest. A behaviour-based approach developed by Cepeda et al. [179] uses a simple control mechanism to allow robots to adopt locate open area, disperse, explore, etc.

Market frameworks have become increasingly popular in recent research as they offer efficient exploration by providing an accurate model of benefit and cost with which to direct robot actions [8]. Simmons et al. [180] present a multi robot exploration approach where robots use a market framework and bidding protocol to determine which robots get to explore which frontier points. The framework developed by Zlot et al. [8] auctions sequences of exploration targets to robots. The approach demonstrates a marked improvement relative to greedy exploration of single exploration targets. Rekleitis et al. [181] use a similar auction framework for the problem of coverage of an unknown environment. Here, tasks allocated to robots are sections from the decomposed geometric structure of the environment based on the current map. Amstutz et al. [164] put forward a boundary coverage technique developed specifically with a group of miniature robots in mind. Miniature robots often have noisy sensors and self localisation capabilities, while com-
munication within the group may be unreliable. The technique deals with these limitations by re-auctioning tasks when it has been determined that a robot has failed, an approach similar to that presented by Nanjanath et al. [165], as discussed in section 3.1.1.

4.1.2 Visual Robot Mapping

In the field of robot mapping there has been an increasing amount of research employing visual sensors [182, 183, 109, 110, 184, 185, 186, 187]. Early research in the area of robot mapping [49] used sonar and laser range sensors, with sonar offering a low-cost option but noisy measurements, and laser offering much higher accuracy but at a higher cost and with the restriction of being slow to use [109]. Over time interest turned to visual sensors as they are low-cost, have low power requirements (so they can be used with small-scale robots), offer high resolution data, can be used to model human perception and can also be used for complementary tasks such as object recognition. Visual data is more complex than range data though, and it is more difficult to determine range and detect landmarks from images. Thus, early uses of visual sensors for robot Simultaneous Localisation and Mapping (SLAM) used artificial landmarks [188].

In order to progress toward operating in novel, unexplored environments, much research in visual SLAM has made use of feature detection approaches from computer vision to allow robots to detect and track visual features that occur naturally. A seminal technique was the Harris feature detector [189]. Harris et al. developed an effective approach to detecting corner features in images and made use of these features in a tracking system where a Kalman Filter was used to calculate camera motion and the positions of environment features in 3-D.

Mikolajczyk and Schmid [190] adapted the system to use a Harris-Laplacian detector, which allowed features to be detected or tracked independently of scale, and developed the Harris-Affine detector [191], which is particularly suited to detecting and tracking features from different poses [187]. Shi and Tomasi’s [192] technique detects salient features based on tex-
ture over a region of an image, where useful regions are determined based on how well they can be tracked. Sim et al. [184] made use of a related technique, the Kanade Luca Tomasi (KLT) [193] approach, in conjunction with a Machine Learning (ML) approach to determine salient features.

Building on this body of research, Lowe [194] introduced the Scale Invariant Feature Transform (SIFT) technique, which has since become ubiquitous in the visual SLAM literature. In this technique, the image gradient around a point is used to create a robust description, independent of scale or orient. The technique first detects potentially interesting or salient corner points using a difference-of-Gaussian function. Location, scale and orientation models are then fit to stable points and a descriptor for the point is generated from these features. When matching features, image data is transformed to match the descriptor's orientation and scale, thus providing a robust mechanism to, for example, track features over consecutive frames.

Another, more recent, technique, FAST features, developed by Rosten and Drummond [195], is many times quicker than straightforward corner detectors. It uses a less advanced corner detector, but employs Machine Learning (ML) to adapt the detector and achieve results of comparable efficacy.

Implementations of these corner-based feature detectors in visual SLAM systems typically assume the ability to detect a large number of features, often 100 to 1000 per frame [196]. Tracking features requires that features first be extracted, matched against a database of features from previous frames to track motion and determine persistent features and then update the database. Efficient database management techniques, using search trees for optimal retrieval, facilitate this in real time. However this is still only feasible on desktop machines or robot platforms with powerful processors, rather than the low power, low-voltage processors required by low-cost robots.

A number of approaches aim to make use of additional information from images, namely colour and intensity, in order to extract more salient features and facilitate tracking, shape from motion, etc. using a lower number of features [197, 198, 199, 200]. The VOCUS attention system detects salient features based on contrast and uniqueness [201]. Given an input image, the
system obtains intensity, colour and orientation responses across the image at 3 different scales. Gabor filters at 4 orientations are used to calculate a feature orientation response. The output is three conspicuity maps, one for intensity, colour and orientation respectively. The uniqueness of features is calculated by assessing the number of similar responses in each image that exceed a specified threshold. A saliency map is generated by calculating this uniqueness scalar for each map and then using this value to weight the combination of the three maps. Alternative approaches to this uniqueness calculation consider the region around the point as well [202], while others consider uniqueness over a number of frames [202].

Frintrop et al. [110] present an active vision technique for visual SLAM using the VOCUS system. The technique uses attention regions in images as landmarks for localisation. Regions of interest (ROI) are calculated by detecting points of maximum brightness in the saliency map and growing regions around these. The ROIs are used to direct an active vision mechanism such that the most salient features are used for tracking, thus improving the robot’s localisation accuracy.

Konolige et al. [203] consider feature extraction for the purpose of visual odometry in outdoor environments where corners are obviously less prevalent. They define Center Surround Extrema (CenSurE) features as those with a dark centre and light surround or vice versa. These are detected using a Laplacian-of-Gaussian (LOG) function, i.e. the second derivative over an image smoothed with a Gaussian filter.

While the techniques listed above use visual data in the context of generating a geometric model of the world, i.e. a map containing spatial relationships between features or landmarks, an alternative approach is to model the environment in terms of its appearance, i.e. mapping equates to generating a reduced dimensionality of the appearance of an environment from a particular pose. Dudek and Zhang [64] use a neural network to generate a compressed model of appearance. Other approaches use a Principle Component Analysis (PCA) model [204], while Dellaert et al. [63] take images of the ceiling in an indoor office-like environment while mapping and use a specific appearance model to aid localisation.
4.2 Low-Cost Robot Exploration

4.2.1 Accurate Mobile Robot Simulation

Section 2.2.3 describes the architecture of the robot control system for low-cost robots developed as part of this work, and explains why this system offers unique advantages over the state-of-the-art approaches, namely:

- Lightweight implementation — The system minimises computational requirements and code-size by using a stripped-down socket-based communication protocol as opposed to the CORBA, ICE or XML-RPC implementations used in the literature. Even though it is useful to maintain an abstract interface between communicating entities, low-cost robots will be severely hampered by such frameworks, while a cleanly designed sockets-based protocol can still be implemented on different platforms and easily updated and extended.

- Extensible plug-in architecture — While other systems provide clean designs allowing modules to be swapped and added to a control system [126], the system implemented here supports even greater adaptability by allowing the entire robot control logic in the form of a set of behaviour modules to be swapped or updated in addition to modules implementing generic functionality such as path planning or state estimation.

- Market framework — The system provides for motivation of robot actions through a market framework by implementing tight integration with external agents, such as the Map Aggregator agent and Coalition Arbiter agent, in the MapProcessing and behaviour modules respectively. Leveraging the benefits of such a framework, the system allows robots to be employed in a distributed manner while still being able to perform different tasks, share information and cooperate to improve performance where applicable.

As the novel contributions of this system apply to low-cost robots, it is important that it can be tested and verified on such robots. While the
physical outputs of robot experiments can be measured to verify techniques and analyse efficiency, the difficulty of development on low-cost robots means that optimising and iteratively updating the system is not straightforward.

To this end, a simulation framework was developed that provides accurate simulation of robot actions and sensor data in addition to supporting analysis and replay of experiments run on actual robots. The system was also designed to impose minimal impact (or messiness) on robot code and can be compiled in different modes such that all simulation code can be removed from executables run on physical robots.

With the exception of differences in floating point accuracy on different processor architectures, the simulation framework allows completely deterministic replay of experiments on physical robots. The well-designed, modular architecture provides that any information that the robot reads from the environment or action that it carries out is abstracted through a clean interface. Thus, the majority of robot code can be left untouched while switching the implementation of these interfaces. Also, by serialising all sensor data obtained through these interfaces an experiment can be re-run in simulation by de-serialising this data. All sensor data is written to experiment to XML log files, with larger pieces of data such as camera images saved as binary data. Experiments involving complex, high-level decision-making can therefore be debugged at a later date, facilitating iterative deliverable-oriented development.

## 4.2.2 Distributed Computation for a Team of Low-Cost Robots

An advantage of the plug-in design adopted in this system, along with the clean design of interfaces between modules and separation of logic and data, is that a robot can serialise the input to a function, for example `Exploration.calculateProfit`, send this to an external agent, and have the output returned. Thus, robots can share processor load and that complex behaviours can be implemented even on robots with low-power processors.

The system presented here provides that robots can make use of dis-
tributed computation to implement any function that would otherwise require a lot of clock cycles. In particular, the functions to evaluate targets for \texttt{LOCAL\_AREA\_EXPLORATION} and \texttt{WIDE\_AREA\_EXPLORATION} are particularly expensive as they provide accurate path planning to each possible target. These functions enable robots to accurately estimate the most profitable target at all times and thus avoid wasting movements or processing of sensor data. However, on the ARM processors described in the appendix, these functions could take in the region of 20 seconds, effectively rendering the robots unusable for real-world applications.

The use of distributed computation dramatically reduces the computation cost on the robot. As can be seen in the snippet of an XML log file from an experiment on a physical robot in figure 4.1, the cost of implementing behaviours is minimal relative to the cost of processing sensor data (\texttt{updateState}) and performing movements (\texttt{processBehaviour}). This system has been verified to be completely deterministic in simulation and on physical robots. It can be switched off and on with flags passed to \texttt{make} and removes all unused code at pre-processor time, meaning that there is no increase in code-size.

### 4.2.3 Visual Obstacle Detection

A key capability of a mobile robot is to be able to move through an environment without unintentionally colliding with any artefacts within it, potentially damaging the robot itself or the environment. As the low-cost robots considered in this work are equipped with visual sensors alone, it follows that the capability to differentiate between free space and obstacles from captured images would have to be developed. As listed below, low-cost robots display a number of inherent characteristics that result in unique challenges when implementing a visual mapping system:

- Monocular vision only — minimising per-robot cost meant that stereo cameras that would facilitate calculation of range from each set of images could not be used, so that other means would be required to generate a map.
Figure 4.1: Snippet of XML log file from an experiment run on a physical robot. Here, nodes showing timing information from a single iteration are shown, with ticks meaning micro seconds. The use of cheap, off-the-shelf sensors and actuators is evident from the large number of clock ticks taken up by grabbing and processing image data (updateState) and performing a movement (processBehaviour). Functions that evaluate and implement robot behaviours such as explorationProfit only take a fraction of the time, while performing each of these functions on the robot itself would take upwards of 20 seconds.

- High image-capture latency — for the robots used in experimentation in this work, described in greater detail in the appendix, there was a huge latency, approximately 6 seconds, involved in capturing an image. This was caused by low bandwidth communication between processor and camera, slow memory write access and low processor frequency. The main implication here is that unless the robot is limited to moving extremely slowly, there are large intervals between captured images. This makes using image data for shape-from-motion impossible, and robustly tracking features to calculate range difficult.

- Low image resolution — as low-cost camera sensors were used, and because the processor would not be capable of processing high resolution images in real time, the images made available to the robots used in this work were limited to a resolution of 176 by 143 pixels. As such, the field of view was limited to approximately 30 degrees.

- High noise levels in images — along with being of low resolution, images
from the camera used were found to have high levels of noise and to be sensitive to lighting conditions. Samples from the set of training images are shown in figure 4.2.

Figure 4.2: Example images used to train appearance for obstacle detection; captured with a CMU Cam 2.

A consequence of these characteristics is that range information cannot be calculated from images but instead must be inferred from appearance. The appearance-based occupancy calculation approach developed to this end is described in the next section.

The robot vision system was developed in the context of the hardware setup as illustrated in figure 4.3. As described in the appendix, the robots used in this work are equipped with CMU Cam 2 cameras. The cameras have a field of view of approximately 30 degrees along the y axis. Therefore, it was decided to mount visual sensors as high as possible on the robot in order to increase the angle relative to the ground and facilitate detection of smaller objects or obstacles.

It was also determined that, during experimentation, robots would prioritise detection and avoidance of obstacles such that they would not sustain damage. Therefore, cameras were oriented such that nearby terrain would be visible. Thus, by being mounted at approximately 0.49 radians, the cameras could observe terrain at a distance of approximately 0.2 metres from the robot’s centre-of-gravity.
The robot code is structured such that the parameters describing different robots can be defined within the same source code. The appropriate parameters to use are then determined at pre-compile time based on a flag passed to `make`. Then further camera parameters are setup in the `Robot.init` function, e.g. the exact area visible to the camera, the minimum and maximum visible distances, the optimum distance from which to observe a target and matrices, etc. for projecting from image space to world space.

![Figure 4.3: Typical camera setup on robots used in experimentation. Here, R is the centre of the robot’s wheelbase. F is the focal point of the robot’s camera.](image)

**Appearance-Based Obstacle Detection**

The technique presented in this work treats occupancy calculation as an object classification. Building on the state of the art in feature detection for visual SLAM [192, 194, 110], this technique describes objects using hue, intensity and texture. Such features have been demonstrated to be suitable for discriminating between artefacts in an environment [201].

As with similar appearance-based object recognition techniques, the approach put forward here uses a database of classified images and a training algorithm to generate a model to describe the different object classes [205]. The technique is also applied to robot recognition, as explained in later chapters, but for occupancy calculation the two classes defined are obstacles and free space. To reduce the manual input required to train the model, all obstacles are identified as belonging to the same class, with grouping of images of each distinct obstacle left to the training algorithm.

Many state of the art object recognition techniques use descriptions
that capture the global appearance of an object, e.g. shape contexts [206], Karhunen-Loeve transforms [207] and Fisherfaces [208]. However, due to the limited field of view of the cameras in use and the close proximity to the ground, obstacles are typically partially occluded or indeed only a small part of them may visible in images captured by robots during experiments. As explained by Murase [207], while such approaches are well suited to conditions where the background can be controlled, these techniques will not be as effective when objects are partially occluded or when image backgrounds contain a lot of clutter.

Therefore, the approach presented here makes use of local features for object recognition, but whereas in the literature such features are typically tracked in order to localise the robot or calculate range, here they are used for obstacle detection, i.e. classification of image feature as either free or occupied terrain.

Instead of attempting to perform segmentation on an image to determine the exact boundaries between different objects, which can be difficult in unstructured environments with a lot of background clutter [209], this approach divides an image into a grid of 9 pixels by 6 pixels cells. These parameters were chosen as they roughly translate to a square area when mapped onto the ground. Also given the low resolution and high noise levels of the images available, this is the smallest artefact that could robustly identified. The corresponding areas in the environment are then roughly equal in size to cells in the occupancy grid.

Thus, the process of segmenting an image is simplified, as instead of searching for specific objects, the approach attempts to classify all areas of an image. The size of regions or cells in an image is small enough that distinctions between regions are excessively blurred, but large enough that texture and hue can be robustly calculated. Hence, the process of calculating occupancy involves calculating a description for each cell, comparing the descriptions to a model of the appearance of free space and of typical obstacles, creating a similarity map of image regions and calculating from this the occupancy of the corresponding area of the environment.

With the low-cost camera hardware used during experiments, it was found
that the appearance of the same objects would vary greatly based on the lighting conditions, as illustrated in figure 4.4. Observing the same object from a different orientation, at a different time of day, with a shadow cast over the object or with different artificial lights would change the appearance of the object in terms of hue, intensity and texture.

As such, obstacle detection was found to be greatly improved when the appearance model contained a representation of lighting conditions. Thus, comparing a region of an image to a trained model involves a weighted combination of a comparison with the region itself and with the image appearance.

Within the technique, each image region is described as a feature vector. A design goal was to include basic appearance features and allow the training system to determine how good each of these features is at distinguishing between objects. Thus, the task of recognition, which would be executed on the robots at run-time, could be kept simple, while complexity could be encapsulated in the training system which generates the appearance model offline. The most computationally efficient way of describing appearance distinct from brightness is to convert from RGB to a cylindrical colour space. Therefore, in this system the Hue-Saturation-Intensity (HSI) space [210] is used to model appearance, where intensity is the measure of brightness and saturation is the measure of colourfulness relative to brightness. Additionally, texture is modelled as variance in these features over the image region. Thus, in the robot vision system, each image cell is described as a vector of 6 floating point values.

The overall appearance of each image is described in terms of the typical appearance of the cells within it, along with a very rough representation of
how appearances are distributed across the image. As with the cell descriptions, a design goal was to keep the task of recognition, or calculation of similarity between an image and different classes in the model, as simple as possible. Therefore, each image is described as a vector containing the mean cell description and the variance over cell descriptions within the image. To this end, the appearance of each image is described as a vector or 12 floating point values in the system.

**Automated Training for an Appearance-Based Classifier**

The appearance-based object recognition system presented in this work includes a training application. The application will generate an appearance model for a set of object classes given a set of input training images. The user may then test the model against another set of images to determine accuracy. Once satisfied, the appearance model can be compiled into code to run on physical robots. The robot vision code used in testing and in physical experiments is therefore identical.

If an appearance model is already present, this will be used to estimate classification as free or occupied in training images and will generate source code that can be compiled to run the training step. If a model is not already present, the system will generate skeleton code for the user to manually classify the images. The images are then displayed to a user, allowing rapid classification.

Given a set of classified images, the training application first creates a description of image regions and of the overall appearance of images. As described above, each image region or cell $c$ is described as a vector of 6 values, based on a HSI representation of the image:

$$c = \begin{bmatrix} H & S & I & \sigma^2_H & \sigma^2_S & \sigma^2_I \end{bmatrix}$$ (4.1)

For each image, the overall appearance is described in terms of the average appearance across the set of image grid cells, e.g. the average variance in hue within image cells, $\bar{\sigma}^2_H$. A coarse representation of how appearance differs across the image is given by the variance of the image cell features, e.g.
the variance in intensity across image cells, $\sigma_I^2$. This is particularly useful in distinguishing between locations which are partially occluded by nearby obstacles or partially covered by shadows, and thus within which free and occupied areas have greatly different appearances from typical locations. The appearance of an image $i$ is thus represented as a vector of 12 features:

$$i = \begin{bmatrix} \bar{H} & \bar{S} & \bar{I} & \sigma_H^2 & \sigma_S^2 & \sigma_I^2 & \sigma_H^2 & \sigma_S^2 & \sigma_I^2 & \sigma_H^2 & \sigma_S^2 & \sigma_I^2 \end{bmatrix}$$  \hspace{1cm} (4.2)$$

The training algorithm first assigns images to a category according to their appearance. Images are grouped based on a K Nearest Neighbour like approach. As training is performed off-line, i.e. not during robot experiments, a brute-force approach is taken to ensure an accurate categorisation is achieved. Similarity measurements are calculated between all images. The similarity between two images is calculated as distance in 12-dimension space, with the distance value along each dimension normalised by the standard deviation along that dimension over all training images.

$$\text{sim}(a,b) = \sum_{f=1}^{12} \frac{|a_f - b_f|}{\sigma_f}$$  \hspace{1cm} (4.3)$$

The $n!$ similarity measurements between all training images are inserted into a sorted queue, iteratively popping the smallest values from the front.
Table 4.1: Accuracy of categorisation of training images for appearance-based obstacle detection.

<table>
<thead>
<tr>
<th></th>
<th>( \bar{x} )</th>
<th>( \sigma )</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance to classified group</td>
<td>3.97</td>
<td>1.82</td>
<td>1.23</td>
<td>10.02</td>
</tr>
<tr>
<td>Distance to closest other group</td>
<td>6.91</td>
<td>3.01</td>
<td>2.42</td>
<td>14.29</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.62</td>
<td>0.24</td>
<td>0.15</td>
<td>1.6</td>
</tr>
</tbody>
</table>

of the queue, and creating a group from the corresponding pair of images. The description of the group is calculated as the mean of its constituent images. If a group \( g \) has been created from images \( a \) and \( b \), then the queue is updated by replacing each remaining value involving either \( a \) or \( b \), with a new similarity against \( g \), e.g. a value for \( \text{sim}(a, x) \) is replaced with \( \text{sim}(g, x) \). As with the training of any classifier, it is intended that the scatter or average distances within a group will be minimised relative to the scatter between groups [208], but as more images are added to a group, the average distance between each image and the group description will typically increase. The error in a group is calculated as the average distance between each image and the group description. To counteract groups becoming too general, the distance is adjusted by the group error \( e \) by an error coefficient \( k \) when recalculating similarities. Thus, the similarity between two groups \( g \) and \( h \) is calculated based on their similarity and the scatter, \( e_g \) and \( e_h \) within the groups:

\[
\text{sim}(g, h) = \frac{1}{12} \sum_{f=1}^{12} \frac{|g_f - h_f|}{\sigma_f} + ke_g + ke_h
\]  

(4.4)

The process runs recursively in this manner until a desired number of groups has been achieved [211]. The training application outputs an analysis of the quality of the generated groups. Quality is estimated by calculating for each image the ratio of distance to its classified group against the distance to the nearest other group. Results for 103 training images for obstacle detection are displayed in table 4.1.

Given a set of images categorised by appearance, the training application generates a description for objects identified in each image group. As
explained earlier, each image region is described using raw appearance, i.e. hue, saturation, intensity and distributions of these values over the region. An interesting extension would be to evaluate spatial relationships between features when generating a model of object appearance, as in the technique demonstrated by Belongie et al. [206]. However, as the results obtained from analysing appearance alone have proved adequate, such extensions have been left for future work.

The training process generates a set of appearance descriptions for each classified object that occurs in an image group. For each object class, the set of corresponding image regions is first amassed and a description for each region generated. The region descriptions are then grouped to form the set of appearances that an object can take on from different poses.

As with grouping images described earlier, a brute-force approach is applied to object appearances. Similarities are calculated between each pair of regions and placed in a sorted queue. Here (lack of) similarity is calculated as distance between regions in 6-dimension space, with distance along each axis normalised by standard deviation along that axis across all regions for that category. Again, the technique avoids scatter within groups by calculating an error metric for each region group and adjusting the similarity value by an error coefficient.

Although a large number of iterations will be required to run the training code, the loops with large n iterations contain straightforward and only floating point code and are thus readily optimised by compilers. Also, potentially the most expensive step of generating groups and updating the similarity queue is greatly optimised by using a dictionary data structure with constant access time.

While the training phase is not real time, once a set of training images has been provided it can typically be run in a matter of minutes. By designing the system to carry out as much computation as possible offline, this enables the code that is run on robots to be optimised in terms of code-size and processor overhead. This is especially important given the small memory and low powered processors typical on low-cost robots. In actual experiments, the entire image processing step typically takes in the order of 100ms.
const int nUnoccupiedCellGroupsForImgGroup12 = 32;
const float unoccupiedCellGroupsForImgGroup12[32][6] = {
    { 130.171395f, 159.459812f, 133.558315f, 25.202128f, 56.638831f,
        35.931384f}, // Group 24, from 47 cells
    { 128.125001f, 150.754631f, 116.815172f, 32.841347f, 82.464905f,
        32.685097f}, // Group 25, from 52 cells
    { 130.453284f, 165.500842f, 158.613638f, 26.875569f, 64.253979f,
        44.273296f}, // Group 26, from 44 cells
    ...
}
const int nOccupiedCellGroupsForImgGroup12 = 32;
const float occupiedCellGroupsForImgGroup12[32][6] = {
    { 122.682660f, 101.889731f, 56.989899f, 67.515910f, 117.431820f,
        16.100000f}, // Group 28, from 22 cells
    { 73.221193f, 93.751029f, 57.884774f, 84.381947f, 102.373613f,
        18.800001f}, // Group 29, from 18 cells
    { 121.492755f, 96.835749f, 52.789856f, 91.915219f, 131.771741f,
        15.436957f}, // Group 30, from 23 cells
    ...
}

const float *unoccupGroupDescs[6];
const float *occupGroupDescs[6];

for (imgGroupIndex = 0; imgGroupIndex < N_IMAGE_GROUPS; ++imgGroupIndex)
{
    switch (imgGroupIndex)
    {
        case 0:
            unoccupGroupDescs = unoccupiedCellGroupsForImgGroup0;
            occupGroupDescs = occupiedCellGroupsForImgGroup0;
            break;
    ...

Figure 4.6: Auto-generated descriptions of appearances of obstacles and unoccupied areas for obstacle detection.
The training application also provides a testing feature, allowing the classification model to be tested against different data sets. Classification statistics are generated and the user may select to have the classifications presented graphically on screen for each image. Again, the actual obstacle detection code in the test application is identical to that executed on the robots in situated experiments.

For each image passed to the obstacle detection system, each cell is compared to the possible appearances of free space or obstacles from the \(k\) most similar lighting conditions in the trained model. The similarity between a cell \(c_t\) in a test image \(i_t\) and an object appearance \(o\) corresponding to an image group \(g\) in the trained model is calculated as

\[
sim(c_t, i_t) = w_c \cdot sim(c_t, o) + w_i \cdot sim(i_t, g)
\]

where the coefficients \(w_c\) and \(w_i\) dictate the degree to which the overall image appearance should effect the classification results. For the results presented in figure 4.7, weights of \(w_c = 1\) and \(w_i = 0.5\) were used. Determining the classification involves comparing the best score for an obstacle to the best score for free space. If the difference between these values is greater than a confidence value, then the cell is marked as occupied or free as appropriate, otherwise the cell is marked as not classified, and so is not incorporated into the robot’s map.

Figure 4.7 shows classification results for a set of test images of the environment in which a robot experiment is to be run. The appearance model had been trained with an equally sized set of images from the same environment, a sample of which is shown in figure 4.2, and, as expected, high classification accuracy has been achieved.

This occupancy calculation technique thus provided robust results, with accuracy in line with results achieved by state of the art classification techniques [205], although the technique presented here deals with much less complicated images and much less variance in test conditions. Nevertheless, the system presented in this work facilitates quick training and verification for deploying robots in a novel environment for experimentation, while the
exact recognition code used in offline verification can be automatically incorporated into the executable that is run on a robot.

Figure 4.7: Classification of occupied and free terrain in a set of test images using the offline training application.

4.2.4 Local-Area Exploration

In this framework, exploration is implemented by robots as a profit-centric behaviour, i.e. a robot does not implement adopt the behaviour in direct response to a stimulus in the environment, but rather upon evaluating its state and the state of the environment, the robot determines that adopting exploration will be the most profitable option.

For the robot to earn profit there must be an agreement in place with another agent or set of agents that specifies that revenue will be exchanged in return for a resource. As described in section 2.1.1, any entity that can be of use to the robot or that can be exchanged for revenue can be modelled
as a resource. In the case of exploration, the robot will need to expend
localisation certainty, battery power and time. From these resources, the
robot will aim to generate map data, which, through the agreement with
Map-Aggregator Agent (MAA), has been defined as a commodity. The robot
can then exchange this commodity for a predefined amount of revenue.

As described in section 3.2.2, the map aggregator agent implemented in
this framework broadcasts that it will exchange revenue for map data. The
utility that it associates with map data depends on how accurate the data
is. Here, accuracy refers to the estimation of the location of a section of
map data relative to a global coordination frame. As demonstrated in sec-
tion 4.2.3, the obstacle detection approach can estimate the location of arte-
facts with high accuracy relative to the robot’s location. However, as will be
discussed throughout this chapter, robots typically quickly accumulate large
errors while exploring while relying only on dead-reckoning for localisation.
Therefore, when processing an image to generate map data, a uniform error
value is encoded in each occupancy cell grid.

In the implementation of the exploration behaviour in the experiments
carried out during this work, the calculation of revenue from map data is
known by all agents and remains constant throughout the experiment. The
parameters of the function can be changed if the mission requires that a
more accurate map is required, or if speed should be prioritised over map
accuracy. Thus, for a section of map data submitted to the MAA consisting
of \( n \) occupancy grid cells, with an encoded localisation error of \( e \), where a
maximum permitted error of \( e_{\text{max}} \) is specified, the revenue for this section of
map data is calculated according to a map coefficient \( k_m \) as
\[
 k_m n \left( 1 - \frac{e}{e_{\text{max}}} \right).
\]

**Behaviour Implementation**

In order to be incorporated into the plug-in framework described in sec-
tion 2.2.2, exploration must be designed to implement the specified behaviour
interface. The interface prescribes a set of functions that the behaviour must
implement, with detailed specifications of the data types to be passed as ar-
arguments and returned. These functions for Local-Area Exploration (LEA)
are as follows:

- **Exploration.calculateProfit** — as this is not a behaviour adopted out of necessity, a robot should only adopt this behaviour if it determines that it will provide the best return in revenue for resources expended. Thus, when `behaviourControl.evaluateBehaviours` is executed, this function will look at the space of potential targets in order to determine the maximum ratio of `profit / cost` that it can achieve. The control module will then compare the ratio for all profit-centric behaviours and determine which to adopt. LEA considers the set of targets over its local map, where targets are areas of the environment roughly correspond to the area covered by a robot’s visual sensor. The motivation behind the treatment of exploration in a hierarchical manner is explained in Section 4.2.5. The value used to compare behaviours, the profit-cost ratio, or utility $u$, is calculated for exploration as $u = \frac{p_n}{r}$, where $p_n$ is net profit and $r$ is the resources that will be expended to achieve a particular target.

Net profit $p_n$ is calculated as $p_g - s$, where $p_g$ is the estimate gross profit, or total revenue that the robot will receive and $s$ is the amount of this revenue that is must allocate to other agents or other behaviours based on existing agreements. Here, gross is estimated based on the amount of map data generated $m$ and the magnitude of uncertainty in the robot’s pose estimate $e_a$ according to the criteria used by the Map Aggregator agent. Additionally, if multiple robots are in the vicinity of an unexplored target, then each will assume that the robot nearest the target will try to explore it. The likelihood of any robot achieving a target is roughly estimated based on the ratio of its distance from the target, and the distance of the closest other robot.

The resources $r$ that a LEA target may require are $e + b + t$. Here, $e$ models the error that the robot will accumulate in its location estimate. The error in a robot’s location estimate is modelled as a covariance matrix. The metric used to quantify this error is the area of the corresponding error ellipse. Thus, if the robot’s current location
can be represented with an error ellipse of area $e_a$, and the estimated error when a target has been achieved is $e'_a$, then the cost associated with this resource will be calculated based on the error coefficient $k_e$ as $k_e (e'_a - e_a)$. In the exploration behaviour presented in this work an appropriate value for the coefficient $k_e$ is calculated by analysing experiment logs. If it is assumed that the robot will hit the maximum permitted localisation error before other factors cause the experiment to end, then by plotting the change in gross profit $p_y$ over localisation error $e_a$, $\frac{dp_y}{de_a}$, integrating over the intervals $e_a, e_{\text{max}}$ and $e'_a, e_{\text{max}}$, the amount of profit that the robot loses out on by accumulating error can be roughly estimated.

The battery power resource $b$ models the work carried out in achieving a target relative to the total amount of work a robot can do before running out of power. For the robots used in experimentation in this work, the amount of power consumed by performing different actions and by remaining idle had been calculated during calibration. For a specific exploration target, as described in detail in the following section, the moves required to navigate to the appropriate destination along with the time required can be estimated.

The coefficient $k_b$ determining the value associated with battery power can be set based on the parameters of the mission, e.g. if it is estimated that the mission will end before battery power is depleted then a minimal cost can be associated with battery power. Conversely, if it is estimated that battery power will be critical, then $k_b$ can be calculated from experiment logs by plotting the average gross profit $p_y$ earned over loss in battery power.

The time resource $t$ required to explore a target can, as with battery power, be estimated accurately as described in the next section. Depending on the parameters of the mission, if it is preferable that robots perform tasks as quickly as possible, then a higher cost coefficient $k_t$ can be associated with time, while for a mission with no time constraints $k_t$ can be set to zero.
In terms of bandwidth, it is assumed that communication between the Map Aggregator and even a large swarm of robots will not exhaust the available bandwidth given that low-power Wi-Fi standards now provide bandwidth in megabytes. The energy cost associated with data communication is negligible when compared with that of the robot motors, i.e. approximately 100mW as opposed to 5W. Thus, it was not deemed necessary that this be added to the cost model.

Thus, by calculating these values, the profit ratios for all possible exploration target can be compared, and the optimal target returned to the robot.behaviourControl module as ProfitData and NavigationData structures. These define the profits and cost involved in exploring that target and the specifics of the target and how to get there respectively.

- **Exploration.adopt** — when it is determined that exploration will be the most profitable behaviour, the robot can call the appropriate adopt function. A table of function pointers is stored in the robot.database.behaviourData structure. For LEA, this function will push a value onto the behaviour stack identifying it as the current behaviour. Due to the modular design of behaviours within the structure, no further setup is required, as the robot-control module can simply look up the appropriate function for LEA to respond to all events.

- **Exploration.checkState** — this will be called at each iteration and when the robot has determined that it has reached the destination specified for the selected exploration target. As illustrated in figure 2.2, the robots movements are carried out in the function behaviourControl.implementBehaviour. By referencing the behaviour stack, this function will be able to retrieve the correct NavigationData structure for Local-Area Exploration. As explained in section 4.2.4, this structure may define that the path to the destination in question should be calculated using a simple hill climbing technique, or using an accurate Inverse Kinematics (IK) technique where the movements of the robot are constrained due to low cost, off-the-shelf hardware components.
For LEA, this function compares the area within range of the robot’s sensors from its current pose against the centre of the target area. If it is determined that the robot has explored the target, either by capturing map data, or by determining that the target is blocked by an obstacle, then the function will return success, thus prompting the control loop to call `handleSuccess`.

- **Exploration.handleSuccess** — this behaviour only considers one target at a time, not sequences of targets like other exploration techniques [8]. Thus, on determining that the current target has been achieved satisfactorily, the `LOCAL_AREA_EXPLORATION` pointer is popped from the behaviour stack, data is written to the log file for generation of stats later and `Available.adopt` is called.

- **Exploration.handleFailure** — typically, the robot may determine that it has failed when trying to reach an exploration target after sensing from its orientation sensor that a rotation movement differed from what was intended, thus bringing the robot to an incorrect destination pose. The behaviour may easily be adapted such that the robot retries to reach failed targets, but in the implementation used in experiments presented here, it was determined that a more flexible approach would be to simple revert to `AVAILABLE` and re-evaluate all behaviours.

The entire state diagram for Local-Area Exploration is shown in figure 4.8. As described in later sections, the `checkState` function may determine that the robot’s path to its destination has suddenly become blocked and that the `FOLLOW_PATH` behaviour should be adopted. Similarly, a collision with another robot may be imminent, in which case the `AVOID_COLLISION` behaviour should be called to manoeuvre away from the potentially dangerous location and ideally revert to `LOCAL_AREA_EXPLORATION` once this has been successfully completed.
Figure 4.8: State transition diagram for the Local-Area Exploration behaviour. Transitions to FOLLOW_PATH and AVOID_COLLISION are initiated from the checkState function.

**Accurate Planning for Motion-Constrained Low-Cost Robots**

When building low-cost mobile robots, off-the-shelf components may to be used. This will increase development speed over building the entire hardware platform from scratch, and may decrease the unit price per robot, as mass produces processors, cameras, motors, etc. are typically more cost-effective than custom build versions.

Off-the-shelf components may however come with limitations that constrain the resultant robot’s capabilities. In the case of the robots used in this research, as described in detail in appendix A, a major limitation that arose was the coarse granularity of moves that a robot could make. This is particularly important in the context of battery life and mission time, as performing a move will expend vastly more battery power and time than performing computations.

For the robots used in experiments in this work, movements were limited to separate forward movement and rotation. The actual distances that the robot could move and rotate were also constrained. Taking one particular robot, for which calibration details are included in the appendix, the shortest forward move that the robot could make was 58.01mm. Additionally, the robot could only rotate in steps of 0.836 radians, or 48 degrees. Bearing in mind that the wheelbase of the robot was 106mm in width, such limitations
have an effect on this ability of the robot to make precise manoeuvres over short distances.

An illustration of this problem is presented in figure 4.9. Here, the robot’s initial pose is rendered in dark, and the pose after subsequent moves is rendered in lighter colour. The point $p$ indicates the centre of the robot’s field of view. In this example scenario, the robot’s behaviour dictates that it must observe a target $t$ such that $p$ is within a specific leeway of $t$, depicted as the thin red circle around $t$.

![Figure 4.9: Example scenario where moving from the initial pose, in black, to a pose to observe the target $t$ requires a planned path given constraints on the robot's movements, i.e. the robot can only rotate or move forward in specific steps.](image)

Given such constraints as those imposed on the robot described above, it is clear that a simple hill climbing approach to arriving at an optimal pose from which to view the target will may be sub-optimal, potentially requiring a lot of moves to correct the robot’s pose.

To this end, the robot control system in this work treats the problem of path planning over short distances as an Inverse Kinematics (IK) problem. In robotics, IK is typically applied to the problem of determining the appropriate rotations around joints such that an end-effector is moved to a required position. Thus, IK is often applied to a problem where a set of rigid bodies are connected around joints to form a kinematic chain. Therefore, the length between joints is constant, but the rotation around joints can be changed. In this case however, the rotations are constrained while the distance between
rotations can be changed.

Given an initial robot pose and a target, the code to calculate the exact moves required is written using a typical Jacobian inverse technique. From its initial pose, the robot determines if it will be possible to rotate and make a specific number of move-rotation pairs. The kinematics module tries this for 1 move-rotation, 2 move-rotation pairs, etc. in order to find the smallest amount of moves that will achieve the desired final pose. Pseudo-code for calculating the exact moves to observe a target with a rotation and 2 move-rotation pairs is shown in figure 4.10.

Given a function \( f \) such that
\[
f : \mathbb{R}^m \rightarrow \mathbb{R}^n
\]
(i.e., \( f \) maps from \( m \) dimensions to \( n \) dimensions, the derivative of \( f \) at any point \( p \) may be given by the Jacobian matrix \( J \), where:
\[
Jf = \begin{pmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m}
\end{pmatrix}
\]

where the partial derivatives are calculated at \( p \) [212]. Thus, in the \( \text{ik} \) function in figure 4.10, a 2-D Jacobian can be constructed to describe the change in observedPoint.x and observedPoint.y as the 2 distances moved change. Given the offset between the observed point and the desired observed point, the inverse Jacobian can be applied to adjust the distances moved to reduce the error.

Any behaviour that requires that a target be observed given a specific leeway can thus flag in the NavigationData structure that an accurate path is required, and the kinematics module will then calculate the exact sequence of moves required.

Figure 4.11 shows a comparison between exploration using accurate path planning for each target, and a hill-climbing navigation approach (For this experiment, the latter approach was simply taken from behaviours that typically involve navigation over longer distances, e.g. \texttt{FOLLOW\_PATH} and
void ik(...)
{
  while (...)
  {
    errorForSequenceOfMoves = calculateError(...);
    if (errorForSequenceOfMoves < leeway)
    {
      return;
    }

    // Calculate Jacobian matrix in 2-dimension space that maps
    // (distance1, distance2) to (point.x, point.y)
    jacobian = calculateJacobian(...);
    jacobianInverse = calculateInverse(jacobian);
    applyJacobianInverseToDistances(jacobianInverse, ...);
  }
}
WIDEAREAEXPLORATION, and applied to LOCALAREAEXPLORATION). The results are taken from a simulated experiment with one robot carrying out exploration. The robots’ kinematics are modelled on the physical robot described in the appendix, i.e. the robot is limited to moving in separate rotations and forward movements, where rotations are constrained to jumps of approximately 47 degrees, while the minimum forward movement is approximately 51mm. The targets in question are LOCALAREAEXPLORATION targets. These correspond to areas of the environment, 0.28 by 0.28 in size. The target is accepted as reached when a sufficient proportion of the area, 0.9 chosen arbitrarily in this experiment, has been mapped.

The hill-climbing navigation approach attempts to minimise the distance between the target and a point at the centre of the robot’s field of view. This may lead to the robot frequently having to navigate away from the target in order to try to observe it again. As suggested in figure 4.12, the targets are typically within close range of the robot — the robot estimates the number of moves to reach a target at 1.46 on average. While this approach proves sufficient for a number of targets, for others the required amount of moves is many times the initial estimate, with the average number required in this case being 5.13.

For the targets approached with accurate path planning, as shown in figure 4.13, there are still some targets for which the planned sequence of moves does not work out. Such instances typically occur when the state of either the environment or the robot changes unexpectedly. For example, an obstacle may block the robot’s path, or after making a rotation, the robot may detect that the resulting orientation differs more than expected from the intended orientation.

4.2.5 Wide-Area Exploration

The task of exploration is divided into local-area exploration and wide-area exploration. As described in earlier sections, the LOCALAREAEXPLORATION behaviour assesses the profitability of exploring areas of terrain roughly equal to that covered by the robot’s sensors, i.e. areas to which the robot can navigate
(a) Simulated environment for single-robot exploration experiment.

(b) Map generated from exploration using accurate path planning.

(c) Map generated from exploration using hill-climbing navigation.

Figure 4.11: Simulated experiment comparing exploration using accurate path-planning and a hill-climbing approach. As described in section 2.2.4, the value in each occupancy grid cell denotes the localisation error and whether the cell is estimated to be occupied or free space. For reference, the maps generated by the robots are shown superimposed over the obstacles from the simulated environment. Unmapped cells have a value of 127, mapped occupied cells have values in the range 1..123, mapped unoccupied cells have values in the range 128..250 and cells with a value of 0 correspond to superimposed obstacles.
Figure 4.12: Number of moves required to reach LOCAL_AREA_EXPLORATION targets. The robot estimates the number of moves required when evaluating different behaviours, while the Kinematics module implements moves to achieve targets.
Figure 4.13: Number of estimated and actual moves required to map LOCAL_AREA_EXPLORATION targets using hill-climbing algorithm.
to immediately. The \texttt{WIDE\_AREA\_EXPLORATION} behaviour assesses the profitability of navigating to an area of the environment and from there assessing specific targets to explore.

The factors motivating this design are as itemised below:

- **Modularity** — As discussed in chapter 2.2.2, a core design principle of the architecture is that a robot’s actions are controlled using behaviours as much as possible. Instead of requiring a complex, overspecific, monolithic control schema to tackle a complex problem, it is preferable if the robot can tackle such a problem by encapsulating logic within behaviours that specify how to act in certain situations, while high-level goals can be considered independent of this complexity.

  Therefore, to ensure that the system is designed to be flexible and scalable, behaviours should be designed to be as coherent as possible, i.e. a robot is motivated to adopt a behaviour in response to a well define event or state, while the behaviour is concerned with achieving a specific goal.

- **Computational complexity** — The \texttt{LOCAL\_AREA\_EXPLORATION} behaviour involves the computationally expensive task of determining the actions required to achieve the necessary position and orientation, and checking the path along which the robot would move for collisions against already detected obstacles. Using the hardware detailed in the abstract, this takes in the order of $10^5$ micro seconds for a single target, while taking $10^2$ micro seconds on an Intel x86 machine.

  Thus, the size of the area over which targets are assessed has a large effect on the computation load on the robot. Adopting a hierarchical approach to target selection is thus necessary in order to allow the system to scale to larger environments.

- **Avoiding local maxima** — As described in section 4.2.4, the approach of maintaining a local map on each robot and considering exploration within the bounds of this map allows for reduced storage and computation overheads on the robots while simultaneously allowing them to
carry out accurate and safe path planning without being overly-reliant on other robots or agents to analyse the environment.

This reduced range means that robots' exploration strategies may be severely sub-optimal, i.e. while robots consider targets in their immediate vicinity that may offer little gain in revenue for a lot of expended resources, another area of the environment may offer better a better revenue to cost ratio.

The areas assessed during wide area exploration area aligned along the same grid as local area exploration targets, and the dimensions are a multiple of those of exploration targets. In the environment representation used in the robot system presented in this work, the exploration targets are 28cm squared, while wide area exploration targets are made up of a grid of 6 x 6 such targets, thus 168cm squared.

Determining an optimal scale at which to analyse the environment may offer improved returns in terms of exploration efficiency (i.e. area mapped relative to resources expended). However, for the implementation discussed here it was determined that considering WIDE_AREA_EXPLORATION targets corresponding in size to robots' local maps.

The areas are thus aligned along the same grid as local area exploration targets, and the dimensions are a multiple of those of exploration targets. In the environment representation used in the robot system presented in this work, the exploration targets are 28cm squared, while wide area exploration targets are made up of a grid of 6 x 6 such targets, thus 168cm squared. This approach offered a number of implementation benefits:

- Analysing the environment — One important advantage of such an approach is that any data gained from processing of local area targets can be used to determine characteristics of the wide area targets in which they are present. In the environment representation presented in this work, the Map Aggregator Agent (MAA) processes each local area target to determine information on obstacles and on how much unexplored terrain is within the area. For each wide area target the
amount of occupied and unexplored terrain is calculated from this information.

- Calculating revenue — The calculateProfit section later in this chapter describes how revenue for wide_area_exploration is calculated based on local_area_exploration revenue. When robots exchange map data for revenue with the MAA, they do so by compressing and sending their local map. Thus, having a wide_area_exploration target that corresponds to a local map means that revenue for the target will be calculated from a single payment from the MAA.

**Behaviour Implementation**

As with local_area_exploration as described in section 4.3, in order to be easily incorporated into the plug-in behaviour framework this behaviour must also implement a specific interface. This allows the higher-level control modules to determine the utility of adopting wide_area_exploration, and allows the logic controlling how events are handled to be encapsulated within the behaviour.

- WideAreaExploration.calculateProfit —

For this behaviour, targets are adopted with the intention that they will bring the robot to a location from which it can explore terrain and obtain revenue from the Map Aggregator Agent. Thus, in a sense, these targets model a more long-term view of robot profitability relative to local-area exploration targets. In this respect, the approach described here has similarities to that developed by Zlot et al. [8], where robots coordinate with each other such that each plans to explore a set of exploration targets in sequence. Here, Zlot demonstrated a substantial increase in performance over greedy selection of targets. Similarly, in a framework presented by Dias et al. [152], robots could coordinate by specifying their preference for a set of paths that each would follow.

The approach presented here differs from such systems, however, as exploration targets are considered over a small area only, i.e. the robot’s local map, so long sequences of local-area targets cannot be considered.
This means that the tasks of arriving at a potentially profitable area and obtaining profit are separated into two behaviours.

However, it was a design goal in this system that, aside from necessary behaviours such as AVOID_COLLISION, each behaviour would only be adopted on the expectation of revenue. Therefore, if robots choose to adopt behaviours without considering their utility, the efficiency of the system may be compromised.

Therefore, the behaviour system allows behaviours themselves to act as agents, such that interactions between agents can be modelled in a profit-centric manner. This approach bears similarities to the agent system described by Vig et al. [9], where each resource was controlled by an individual agent, with which agents modelling tasks would interact.

The interaction between WIDE_AREA_EXPLORATION and LOCAL_AREA_EXPLORATION has been implemented in a straightforward manner so that minimal coordination between the two would be required. The behaviours do not arrive at a prior agreement for exchanging revenue. Rather, when a robot adopts a wide-area exploration target that brings the robot to a new location, the behaviour control system imposes the obligation that local-area targets explored within this area must attribute a proportion of revenue to the initial wide-area exploration target.

Evaluating the profitability of a wide-area target thus involves estimating the cost required to get to the target area and the revenue generating by subsequent local-area targets. As with local-area targets, the utility $u$ of a target is calculated as $\frac{p_n}{r}$. As with local-area targets as described in section 4.2.4, the resources $r$ required are calculated in terms of error accumulated, power and time. Net profit, $p_n$, is calculated as gross profit minus expenditure $p_g - s$. Here gross profit equates to revenue obtained from local-area targets. To estimate this value, the robot must determine how many exploration targets will likely be subsequently adopted and the combined gross profit of these
Figure 4.14: Proportion of robots’ local maps explored per iteration. Each local map contains 7056 occupancy cells.
targets. It is assumed that the wide-area target will then receive an income corresponding to a proportion of each.

For each wide-area target, these values are estimated based on statistics from previous experiments. Figure 4.14 illustrates the typical trend while updating a robot’s local map to centre on its location as it explores an environment. As local maps overlap (so that the robot’s location is always within its local map), new local maps typically have 0.2 of their area already explored. Then as the proportion increases within each local map, a robot will typically have to expend more resources to map new targets, thus the profit ratio for exploration targets decreases. Figure 4.15 demonstrates this trend using data from a simulated experiment with two robots exploring independently.

![Figure 4.15: Profit ratio for local-area exploration targets relative to the current proportion of the robot’s local map explored.](image)

The gross profit for subsequent local-area targets is calculated using an iterative algorithm as described in figure 4.16. For the sake of clarity,
man y of the details of the code are omitted here, but the actual code employs the same algorithm.

The profit ratio that a robot can expect to achieve for local-area targets can be estimated based on the state of the environment and of the local map. The relevant parameters are as follows:

- Error — When evaluating the initial wide-area target, the robot will have formed an accurate estimation of the magnitude of error in its localisation estimate upon arriving at the target area. This value can be updated upon calculating the number of moves required to achieve each local-target.

- Resources — As illustrated in figure 4.18, a rough estimate of the moves required for each local-area target can be obtained from the proportion explored in the local map. The simulation framework discussed in section 4.2.1 allows results from both real and simulated experiments to be analysed to obtain values for generating these estimates. The robot code then contains a set of values for different experimental setups, with the appropriate values selected at pre-processor time (in order to cut down on code size).

- Area mapped — Again, analysis from experimental results provides coefficients for estimating the area of terrain mapped for each local-area target. Figure 4.17 shows results from a simulated experiment. Here, the values correspond to the estimate the robot makes when adopting each local-area target.

It is important that wide-area exploration adopts an appropriate value for the proportion of revenue that local-area targets will share with it. As described in section 4.2.4, the expenditure $s$ will determine the net profit $p_n$ of each subsequent local-area target. If the proportion is too low, then the target may not be profitable. If the proportion is too high, then the robot may determine that the local-area targets are not profitable and move to explore another area.
totalRevenue = 0;
navData = calculateMovesToLocalMap(localMap);
localisationError = navData.errorAtDest;
do {
    revenue = estimateLocalAreaExplGross(proportionLocalMapMapped, localisationError);
    totalRevenue += revenue;

    nMoves = estimateLocalAreaExplMoves(proportionLocalMapMapped);
    localisationError = updateLocalisationError(nMoves);
    proportionLocalMapMapped = updateLocalMapMapped(proportionLocalMapMapped, nMoves);
} while (proportionLocalMapMapped < localMapThreshold);

wideAreaExplProfit = calcWideAreaExplProfit(totalRevenue);
wideAreaExplCost = calcMoveCost(navData);

wideAreaExplRatio = wideAreaExplProfit / wideAreaExplCost;

Figure 4.16: Pseudo-code for calculating gross profit for a WIDE_AREA_EXPLORATION target.
In the implementation described here, the proportion of a revenue that a local-area target loses as expenditure is defined as a constant value, \( k_s \). The value was calculated by evaluating statistics from experiments to compare the resources expended by wide-area targets and subsequent local-area targets. It would be an interesting direction for further research to consider a mechanism that allows \texttt{WIDE\_AREA\_EXPLORATION} to determine an optimum proportion to maximise its own profit.

![Estimated area mapped per target](image)

**Figure 4.17:** Estimated area mapped for \texttt{LOCAL\_AREA\_EXPLORATION} targets relative to the current proportion of the robot’s local map explored.

- \texttt{WideAreaExploration.adopt} — As with other behaviours, the control module can look up this function from a pointer table maintained in the \texttt{BehaviourData} structure. Upon adopting this behaviour, a robot will update its behaviour stack with an appropriate identifier. Electing to explore a particular target may imply an additional change of state for the robot. If currently in a coalition, then a robot is limited to exploring target areas within a certain range. If a target outside this range is
Figure 4.18: Estimated moves required for \texttt{LOCAL\_AREA\_EXPLORATION} targets relative to the current proportion of the robot’s local map explored.
deemed more profitable, even without the potential profit increase that the coalition affords, then the robot should leave the coalition, and is obliged to communicate this to other robots via the Coalition Arbiter Agent.

As revenue for this behaviour, as discussed in the previous point, is calculated based on local-area targets, this behaviour must initialise an ExplorationSession structure within the WideAreaExploration structure in the robot's database to ensure that revenue will be accredited as appropriate.

- **WideAreaExploration.checkState** — As robots considered in this work are constrained in terms of the number of operations they can perform, either due to time constraints on the mission or resource constraints, it is important that moves are not made when they will not be profitable. As the path traversed to a new local map may involve a large number of discrete moves a robot, it is therefore prudent that the robot evaluate the status of the target area en-route. A robot will therefore check the state of the target area to determine that it has not been explored by another robot. Also, as the path may be continuously updated to navigate around newly-discovered obstacles, the profitability of continuing to the target area should be assessed.

- **WideAreaExploration.handleSuccess** — Update the ExplorationSession structure to define parameters for attributing revenue to WIDE_AREA_EXPLORATION for having performed this task. Subsequent exploration targets within this area are obliged to accredit a portion of revenue according to pre-defined rules governing how the two behaviours operate together.

This is implemented by registering two event handlers with the behaviour control system. Event handlers may be added at various points throughout the system, and are implemented by simply executing a list of function pointers in sequence. The control system's readProfit function will usually simply allocate revenue received to the appro-
priate behaviour and target. By registering a handleProfitAllocated handler, WIDE_AREA_EXPLORATION can ensure that the correct proportion of revenue allocated to local-area targets are instead allocated to it. A handleLocalMapChanged is also registered to remove this handler (and itself) when the robot moves to a new local map.

- **WideAreaExploration.handleFailure** — If a robot determines that a target area cannot be reached, e.g. it is completely surrounded by obstacles, it should communicate this to other robots. The Map Aggregator Agent described in section 3.2.2 provides a condensed representation for recording this, requiring a single bit is used per target area.

Thus, WIDE_AREA_EXPLORATION provides robots with a mechanism to effectively select exploration targets over a large environment, while not demanding an increase in computation load or storage. The behaviour is designed such that it can be removed from a robot control system with a few quick changes. While adding some complexity to the system, in terms of agreements between behaviours and event handlers, the behaviour may still be added and removed safely without affecting the rest of the system. Adoption of the behaviour is motivated by profit, thus enabling the robot to balance it with other behaviours as resources or the state of the environment change. Results from experiments utilising this behaviour are shown in section 4.3.

### 4.3 Exploration Results

This section discusses results from simulated experiments using the framework described in section 4.2.1. Simulations were run with accurate models of robot forward kinematics and navigation error based on the calibration of the robot platforms detailed in the appendix.

The experiments detailed in this section validate the map representation employed and the Map Aggregator agent described in section 3.2.2, while also demonstrating the flexibility of the behaviour architecture detailed in section 4.2.1. While not explicitly cooperating, the robots share state information, including current location, behaviour and task. It is evident from
experiments using multiple robots that this allows the task of exploration to be effectively parallelised. The implementation of robot control using a profit-centric framework is shown to support adaptability of robot behaviour to the requirements of the mission. The human controller may prioritise accuracy, speed, etc. while parameters controlling interactions between behaviours can be adjusted to affect behaviour patterns.

4.3.1 The Effect of Market Values on Robot Behaviour

Figure 4.19 demonstrates the effect of changing the required map accuracy when running an experiment with a single robot to carry out exploration. Here, the maximum allowed error magnitude for a robot submitting map data is set to 10, 40 and 80. As stated in section 3.2.2 the error magnitude metric used in these experiments is calculated based on the area of the error ellipse for $1\sigma$ for the covariance matrix representing the robot’s location uncertainty. Specifically the value of $\text{area}/\pi$ is used to give an intuitive representation of the size of the ellipse. Units used here are measured in occupancy grid cells, i.e. 0.02m. Thus, a location estimate with an error magnitude of 10 will expect the estimate to be within 0.5m of the actual location, based on the 68-95-99.7 rule.

The simulation framework injects zero mean Gaussian error into robot movements according to the error model generated during calibration, while the random number generator is seeded with time. As can be seen in figure 4.20, although subject to random variances, error in a robot’s location estimate is accumulated linearly initially. Table 4.2 shows parameters of the resulting maps generated. When high accuracy is required in the generated map, a single robot equipped only with dead-reckoning for localisation will quickly accumulate error causing it to be unusable.

Table 4.3 illustrates the effects of changing the value attributed by robots to internal resources. In this case the monetary value attached to battery power is changed across experiments. Here, one unit of currency equivalent to that received for one cell of the occupancy grid with 0 localisation error submitted to the Map Aggregator agent. Due to the accurate path planning
(a) Simulated environment for single-robot exploration.

(b) Maximum error magnitude is set to 10.

(c) Maximum error magnitude is set to 40.

(d) Map generated where maximum error magnitude is set to 80.

Figure 4.19: Simulated experiment comparing maps generated with varying maximum permitted localisation error for map data submitted to the Map Aggregator agent. Here the greater variance in shade visible in latter images are due to the greater range of error encoded in the occupancy grid.
Table 4.2: Experiment results with varying maximum permitted error.

<table>
<thead>
<tr>
<th>Max permitted error</th>
<th>Avg map error</th>
<th>N robot moves</th>
<th>Area mapped</th>
<th>Avg loc error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.1917</td>
<td>148</td>
<td>0.0485</td>
<td>19.8496</td>
</tr>
<tr>
<td>40</td>
<td>18.7551</td>
<td>475</td>
<td>0.1614</td>
<td>46.6049</td>
</tr>
<tr>
<td>80</td>
<td>31.7684</td>
<td>805</td>
<td>0.2498</td>
<td>44.0172</td>
</tr>
</tbody>
</table>

Figure 4.20: Distance between estimated and actual robot location. Error is injected in simulated experiments using the Box-Muller technique to generate normally distributed random values according to calibration results from physical robots.
algorithm employed for exploration, there is little variance in the amount of map data accumulated. As robots always calculate the most efficient move in terms of the return in profit, the manner in which LOCAL_AREA_EXPLORATION targets are approached does not alter.

While robots were able to obtain a similar amount of map data with a similar accuracy, it is notable that increasing the value of battery power caused the robots to reach the maximum permitted localisation error in fewer moves. This may be explained by the increased adoption of WIDE_AREA_EXPLORATION. As this behaviour does not itself focus on gaining map data, but is attributed profit by LOCAL_AREA_EXPLORATION, it will be greedily adopted when the robot determines that more profit may be gained for less exhaustion of resources.

A similar assumption regarding the efficiency of the LOCAL_AREA_EXPLORATION planning mechanism may be made based on figure 4.21. Here, the robot attributes different monetary values to the certainty in its location estimate, where certainty is measured based on the area of the error ellipse corresponding to its covariance matrix. Although the amount and accuracy of map data obtained is similar for all values, it is evident in the generated maps that when robots attempt to conserve localisation accuracy, more nearby exploration targets will be adopted, while where other resources such as time or battery power are prioritised, robots will explore further through the environment.

Figure 4.23 shows the effects of varying the parameters controlling how WIDE_AREA_EXPLORATION and LOCAL_AREA_EXPLORATION interact on a robot’s exploration performance. A notable side effect of this interaction is the

<table>
<thead>
<tr>
<th>Battery power value</th>
<th>Avg map error</th>
<th>N robot moves</th>
<th>Area mapped</th>
<th>N wide-area targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>14.3681</td>
<td>383</td>
<td>0.1353</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>13.6740</td>
<td>353</td>
<td>0.1381</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>14.9600</td>
<td>302</td>
<td>0.1390</td>
<td>6</td>
</tr>
<tr>
<td>120</td>
<td>14.1755</td>
<td>286</td>
<td>0.1385</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>13.5230</td>
<td>254</td>
<td>0.1113</td>
<td>9</td>
</tr>
<tr>
<td>400</td>
<td>15.7579</td>
<td>267</td>
<td>0.1290</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4.3: Experiment results with varying battery power. The prioritisation of battery power is reflected by adjusting the associated monetary value.
(a) Cost of accumulating a unit of error magnitude is set to 50.

(b) Cost of accumulating a unit of error magnitude is set to 200.

(c) Cost of accumulating a unit of error magnitude is set to 300.

(d) Cost of accumulating a unit of error magnitude is set to 400.

Figure 4.21: Simulated experiment comparing maps generated with varying values associated with localisation accuracy. Robots use this value to calculate the cost of accumulating error when considering the profitability of actions.
Table 4.4: Experiment results with varying profit allocated to \texttt{WIDE\_AREA\_EXPLORATION} targets.

<table>
<thead>
<tr>
<th>Proportion of profit</th>
<th>Avg map error</th>
<th>N robot moves</th>
<th>Area mapped</th>
<th>N wide-area targets</th>
<th>N local maps</th>
<th>Avg local map area explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10.0765</td>
<td>235</td>
<td>0.0939</td>
<td>1</td>
<td>10</td>
<td>0.4178</td>
</tr>
<tr>
<td>0.3</td>
<td>9.4242</td>
<td>244</td>
<td>0.0912</td>
<td>2</td>
<td>13</td>
<td>0.3351</td>
</tr>
<tr>
<td>0.5</td>
<td>10.0522</td>
<td>187</td>
<td>0.0840</td>
<td>8</td>
<td>22</td>
<td>0.1653</td>
</tr>
<tr>
<td>0.8</td>
<td>10.0784</td>
<td>164</td>
<td>0.0871</td>
<td>10</td>
<td>24</td>
<td>0.1554</td>
</tr>
</tbody>
</table>

amount of local maps a robot submits to the Map Aggregator, and the proportion of each local map that has been explored. As described in section sec:widearea, the \texttt{WIDE\_AREA\_EXPLORATION} behaviour facilitates the exploring of \texttt{LOCAL\_AREA\_EXPLORATION} targets in a new area of the environment, with the imposition that a proportion of revenue for any targets in that area be accredited to it.

\texttt{WIDE\_AREA\_EXPLORATION} targets are typically adopted when a robot has explored its immediate vicinity, including the adjacent local map areas, while other areas of the environment contain large sections of unmapped terrain. The proportion of profit attributed to these targets may be adjusted to influence how quickly a robot will cease exploration in one environment and move to the next. This may be useful, for example, where a comprehensive map of the environment is not required, but rather as broad an area of the environment should be mapped as quickly as possible.

Table 4.4 illustrates the effect on the generated map when the proportion is altered. These values are taken from a set of simulated experiments with one robot exploring a novel environment. The duration of these experiments was limited by the maximum localisation error with which the robot was permitted to operate.

As expected, the proportion of \texttt{WIDE\_AREA\_EXPLORATION} targets selected increases linearly with the increase in profit. As a consequence, the robot mapped a larger of local maps as the revenue proportion was increased, though a smaller proportion of each local map submitted to the Map Aggregator was explored. As a result, a slight decrease in the overall area of
Figure 4.22: Local maps submitted to the Map Aggregator with varying proportions of map revenue attributed to WIDE_AREA_EXPLORATION targets. In the map structure used in these experiments, there are 7056 occupancy cells in each local map.
Table 4.5: Experiment results with varying n robots exploring independently.

<table>
<thead>
<tr>
<th>N robots</th>
<th>Avg map error</th>
<th>N moves</th>
<th>Area mapped</th>
<th>N local maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.7333</td>
<td>210</td>
<td>0.0920</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>9.5465</td>
<td>451</td>
<td>0.1735</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>9.3991</td>
<td>875</td>
<td>0.3268</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>9.0722</td>
<td>1231</td>
<td>0.4744</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>7.671</td>
<td>1427</td>
<td>0.5066</td>
<td>83</td>
</tr>
</tbody>
</table>

the environment explored is incurred, as the robot spends more iterations navigating to wide-area targets instead of gathering map data. The average accuracy of map data collected is not affected by the change in behaviour. As moves to each target are specifically planned, the robot will gather map data uniformly as it accumulates localisation error.

As illustrated in figure 4.22, a lower number of WIDE_AREA_EXPLORATION targets will result in more efficient use of local maps, which will result in lower overheads related to processing and communicating maps to the Map Aggregator agent.

4.3.2 Exploration with Multiple Robots

Figure 4.24 shows the map of a simulated environment generated when different n robots are deployed. While there is no coordination between the group, robots will indirectly be repelled from one another when considering exploration targets based on the heuristic described in section 4.2.4, i.e. when calculating the utility of an exploration target, robots will consider the probability of achieving the target given the proximity of other robots to it. Thus, as can be seen in the resultant maps, robots immediately disperse through the environment, resulting in minimal overlap between paths traversed by different robots.

Table 4.5 shows further information on the maps generated. While most parameters increase linearly with n robots, it can be seen that there are slight improvements in the accuracy of the map generated. This occurs because a greater proportion of map data is generated earlier on in experiments with larger n robots. As robots explore their immediate vicinity, they move
(a) 0.1 of local-area revenue is attributed to the corresponding wide-area target.

(b) 0.3 of local-area revenue is attributed to the corresponding wide-area target.

(c) 0.5 of local-area revenue is attributed to the corresponding wide-area target.

(d) 0.8 of local-area revenue is attributed to the corresponding wide-area target.

Figure 4.23: Experiments demonstrating varying levels of profit attributed to WIDE_AREA_EXPLORATION targets.
Table 4.6: Exploration efficiency normalised over n robots.

<table>
<thead>
<tr>
<th>N robots</th>
<th>Area mapped per robot</th>
<th>Avg area mapped per local map</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0920</td>
<td>0.3312</td>
</tr>
<tr>
<td>2</td>
<td>0.0867</td>
<td>0.2231</td>
</tr>
<tr>
<td>4</td>
<td>0.0817</td>
<td>0.2220</td>
</tr>
<tr>
<td>6</td>
<td>0.0790</td>
<td>0.2162</td>
</tr>
<tr>
<td>8</td>
<td>0.0633</td>
<td>0.2197</td>
</tr>
</tbody>
</table>

outward to different local maps. When this occurs there is some overlap between robots' paths, resulting in a slight drop in the proportion of map data generated per move.

This effect is illustrated in table 4.6. There is a slight drop in the exploration efficiency of each robot when the number of robots is increased. Additionally, the efficiency with which map data is sent to the Map Aggregator is reduced, as robots have to navigate further to reach unexplored terrain, resulting in many local maps being submitted with a low proportion of new map data.

4.3.3 Experiments with Physical Robots

Experiments were run with physical robots, as depicted in figure 4.25. These experiments were run in order to validate the robot control system developed in this work and the novel exploration techniques required to deploy low-cost, off-the-shelf robots. It was desired that the following contributions in particular be validated:

- Behaviour control framework — Allow robots to evaluate their surrounding and autonomously determine what behaviours to employ and what actions to take.

- Combining map data via an external agent — Robustly communicate map data to an external agent such that the storage and computational load of maintaining and analysing the map is not placed on the robots themselves.
(a) Exploration using 1 robot.  
(b) Exploration using 2 robots.  
(c) Exploration using 4 robots.  
(d) Exploration using 8 robots.  

Figure 4.24: Maps generated in simulated experiments with multiple robots. Here, robots explore in INDEPENDENT mode, with no collaboration.
• Visual mapping with low-cost cameras — Provide robots with a mechanism to accurately differentiate between occupied and free space and to translate this data into an occupancy map.

• Market framework — Motivate robots to act in exchange for profit and allow the formation of coalitions between robots in an autonomous manner.

The experiment in figure 4.25 was set up to employ two robots to perform collaborative exploration. Two robots are set up with known positions and communicate their state via Bluetooth to a Map Aggregator agent running on a laptop. The robots immediately form a coalition, with one robot electing to perform WIDE_AREA_EXPLORATION followed by CLOSE_LOOP to perform cooperative

Figure 4.25: Experiment setup with 2 physical robots. The experiment in question was run in COLLABORATION mode, with both robots connected to a Map Aggregator and Coalition Arbiter agent running on a laptop.
localisation for the removal of accumulated error from the map data it has gathered.

The sequence of images shown in figure 4.26 demonstrates the robot’s ability to calculate occupancy and its employment of \texttt{WIDE\_AREA\_EXPLORATION}, \texttt{FOLLOW\_PATH} behaviours, including an updating of the robot’s path as an obstacle is encountered.

The compasses used in this research, as detailed in the appendix, provided accurate measurements within areas of the test environment. The measurement errors recorded matched those expected and were in line with those reported in similar research, e.g. recent work by Mao et al. [213] on localisation using similar low-cost robots. While the results in experiments verified the findings in calibration, it was found that electrical interference in the environment chosen for experiments caused severe inaccuracies in the readings as robots moved over larger distances.

Figure 4.27 shows a snippet from the log file from the experiment shown in figure 4.25. While iterations 0 to 14 show expected orientation measurements, where differences between expected and actual orientation concur with the odometry errors found during calibration, iteration 15 shows a sudden jump in 2.2477 radians, or 128.7865 degrees.

To deal with this limitation, it was necessary to implement an alternative \texttt{SensorProcessing} module where compass readings were ignored and uncertainty in orientation was instead accumulated as the robots moved. As orientation error from movement had already been calibrated and modeled in the simulation framework, it was possible to simply run a set of simulated experiments in randomly generated environments to update constant values used in each robot’s config file, e.g. the values mapping the proportion of unexplored terrain in a local map to the average number of \texttt{LOCAL\_AREA\_EXPLORATION} targets adopted within that map.

### 4.4 Summary of Contributions

Section 4.2.1 presented a unique simulation and robot control framework. The framework allows robot control code for miniature robots (which fre-
Figure 4.26: Images from an experiment with a physical robot carrying out \texttt{WIDE\_AREA\_EXPLORATION}. The robot adopts the \texttt{FOLLOW\_PATH} behaviour to navigate to another local map, and updates its path dynamically upon observing an obstacle.
Figure 4.27: Snippet of log file from the experiment depicted in figure 4.25 illustrating the problems encountered with the low-cost compass modules used for global orientation measurements. While iterations 0 to 14 suggest that the compass is operating within the expected accuracy margins and correcting estimated orientation from dead-reckoning, iteration 15 shows a sudden jump after a forward movement, presumably caused by environmental interference.
sequently have ARM-based processors) to be run in a simulation with accurate modelling or robot movements and sensor data. The framework greatly improves the ease of development on such robots by allowing experiments run on physical robots to be replayed in simulation in order to debug issues and evaluate performance. Importantly, when code is compiled to run on physical robots, all functionality not required by the robots is completely removed, meaning that it can be used on robots with severely limited storage capacity.

Section 4.1.2 describes a novel robot vision system for use with low resolution images with high levels of noise. The system, while straightforward in its design, has the advantages of imposing very low computation overheads on robots, being able to robustly determine occupancy from monocular visual data and offering a very quick setup time. The system incorporates an offline system for setting up training images, training and verifying the recognition model and exporting the model and recognition code for inclusion in the executable to run on the actual robots.

Section 4.2.4 describes a novel exploration technique developed for low-cost robots with monocular visual sensors. While such sensors can potentially provide rich information for robots, allowing them to recognise objects in the environments and be deployed for further tasks such as search-and-rescue or box-pushing, they result in a severely limited range for robots, in particular when image quality is poor and recognition of obstacles at longer distances is not robust. Whereas the problem of constrained movements is not typically addressed in research with wheeled robots, it is an issue with low-cost platforms and also with humanoid robots. This technique allows efficient exploration given these constraints by calculating exact paths to exploration targets, thus allowing power and time to be conserved. Further behaviours described in the remaining sections facilitate this exploration approach within the market framework described in chapter 3.
Chapter 5

Distributed Collaborative Exploration for Low-Cost Robots

The previous chapter demonstrated that deploying multiple robots may decrease the time required to explore an environment. However, when robots operate independently without making use of information provided by other robots, either implicitly or explicitly, then there is no gain in the accuracy of the map generated. Thus, if more robots are added to the group for such tasks, the area may be mapped more quickly, but this may be immaterial if the generated map is too inaccurate to be of use. This is particularly problematic given the low-cost robots considered in this research. Such robots are categorised by inaccurate actuators and low-quality sensors with low-resolution and high noise levels.

An important open research question is thus how such robots can be deployed such that the quality of the work done is improved. This chapter will explain how behaviours implemented as part of the robot architecture described in chapter 2 can achieve improved mapping accuracy. The behaviours prompt robots to form coalitions to ensure collaboration during the exploration of regions in the environment, in which robots use each other as artificial landmarks to remove localisation errors accumulated while exploring and to correct errors in map data collected.

This works within the same market framework covered in section 3.2.1,
thus allowing robots to determine when it is appropriate to cooperate in a
distributed, flexible manner. As described in section 3.2.1, the framework
developed here supports the formation of coalitions between robots, while
the negotiation overhead is reduced by using a Coalition Arbiter agent to
hold auctions and maintain the state of coalitions between robots.

In the remainder of this chapter, section 5.1.1 describes related approaches
in the literature such as the state of the art in multi-robot localisation, includ-
ing research on cooperative localisation with low-cost robots. Section 5.2.1
explains the technical details of the relative localisation method presented
in this work, while sections 5.2.2 and 5.2.3 describe the main behaviours
designed to form coalitions to improve mapping accuracy. Results are then
discussed in section 5.3.

5.1 Related Work

5.1.1 Cooperative Localisation in Multi-Robot Teams

In recent years, cooperative localisation has become increasingly prevalent in
the literature and has been demonstrated as an effective approach to state
estimation in environments where GPS is not available. Such environments
include underwater environments, indoors, underground and other planets
[84]. Techniques vary from those where accuracy of state estimation is pri-
oritised and some robots are kept stationary to act as artificial landmarks to
those where efficiency is prioritised and all robots move at the same time.

An early cooperative localisation approach presented by Kurazume et al.
[144] involves dividing a team of robots into two groups, where one group re-
ains stationary, acting as artificial landmarks, while the other group moves.
The technique provides much lower error accumulation as opposed to odome-
try, and has been employed for various tasks including mapping and floor
cleaning.

Rekleitis et al. [1] demonstrate a related approach for simultaneous locali-
sation and mapping. Here, two robots are used with one remaining stationary
while the other moves, maintaining visual contact at all times. Robots nav-
igate by constructing a geometric model of the environment to determine suitable points at which to swap roles. Robots used in experiments were equipped with range and visual sensors, and were fitted with patterns to facilitate estimation of relative position and orientation.

An alternative set of approaches in the literature allows the entire group of robots to move and exchange relative localisation measurements within the group while moving. Giguere et al. [84] present a technique where bearing-only measurements between robots are used. Robots are equipped with LED lights mounted on dark backgrounds to act as salient landmarks and omnidirectional cameras for detecting each other. Montesano et al. [214] demonstrate the use of a particle filter and an EKF to track bearing measurements between a pair of robots and show how the combined filter increases robustness.

Maxim et al. [215] use range measurements between robots to estimate relative location by trilateration. The technique has been used to control outdoor robots moving in formation. Das et al. [216] describe a distributed approach to navigating in formation. Here, a group of robots would elect a leader around which the other robots would adopt a network formation for exchanging location estimates.

In a technique presented by Roumeliotis et al. [145, 5, 217] and more recently Mourikis [4] a group of robots navigating to a target exchange relative location estimates with each other. An EKF is used to combine all relative position estimates, while orientation is ignored and assumed to be available using global measurements. An upper bound on the accumulation of uncertainty is presented, while it is shown in experiments that errors in robots’ position estimates were consistently below the theoretical limits. Experiments demonstrate that the accumulation of positioning uncertainty does not depend on the topography of the robot group or on the accuracy of relative position estimates, but on the accuracy of the robots’ proprioceptive position estimates and the number of robots in the group.

A study by Rekleitis et al. [218] evaluates the efficiency and accuracy of measuring relative orient, range, position and full pose between robots for cooperative localisation, and demonstrates that measuring the full relative
pose provides the most effective reduction in error accumulation. Related to these approaches, Kosmatopoulos et al. [219] present a simultaneous co-operative localisation, mapping and target tracking system, which combines the location estimates of multiple robots with the system modelling the environment. The approach has been demonstrated in simulations with two autonomous air vehicles.

With specific regard to small, low-cost robots, Mao et al. [213] illustrate the difficulties encountered when robots with inaccurate self-localisation and sensors are used. The technique presented in this work demonstrates relative localisation for the task of robot-docking, i.e. one robot navigating to a position beside another robot.

Grabowski and Khosla [2] present an approach employing similarly small Millibot robots and describe a cooperative localisation technique with Maximum-Likelihood estimation. In this approach, each robot moves in turn and emits a signal that can be used by other robots to calculate relative range. Rothermich et al. [3] utilise cooperative localisation in a SLAM approach developed for use with a robotic swarm. Here, robots use a simple control schema which directs them to navigate or act as landmarks depending on their proximity to other robots and to the state of the environment around them. Robots communicate their desire to move to other robots, with cooperation emerging as robots competing for the same task will acquiesce to the robot with the highest stated desire. The technique was demonstrated using iRobot’s Swarmbot platform [220]. The robots do not directly sense the environment themselves, but infer free space from visibility to other robots.
5.2 Behaviour-Based Collaborative Exploration for Low-Cost Robots

5.2.1 Relative Localisation with Low-Cost Visual Sensors

A motivation factor in the use of low-cost robots is the availability of inexpensive sensors that are capable of providing detailed information about the environment and can be used to overcome the inaccuracy of typical actuators and self-localisation mechanisms.

The specific problem considered in this work is the use of visual sensor data to offset the rapid growth in localisation error. While a range of cooperative localisation techniques in the research using range measurements has been presented, it was decided to use visual sensors here as the availability of colour information meant that robots could unambiguously distinguish between one another. Thus, in contrast to cooperative localisation techniques where robots are strictly coordinated or assumed to maintain a strict formation [3, 2], this system allows robots to operate in a truly autonomous, unstructured fashion.

In this work, the problem of detecting robots is somewhat simplified by having distinct colours on each robot, a practice common in the literature [218, 84]. In addition, robots are aware of the exact dimension of other robots, allowing for robust calculation of relative location and orientation.

As with the occupancy detection technique described in section 4.2.3, the system developed for detection and relative localisation of other robots was designed to be as lightweight as possible in terms of both storage and computational demands on the robots.

Common visual feature detection techniques in the literature are outlined in section 4.1.2. The recognition approach here is based on the same functionality used to distinguish between occupied and free space as described in that chapter. While having the advantages of saving on code size and development time, this technique provides robust detection and differentiation between robots.
Table 5.1: Classification of image cells against a model of robot appearance in a set of test images. The grid of cells in each test image is classified using the training application and then compared to an appearance model for the set of colours assigned to robots.

<table>
<thead>
<tr>
<th>Colour index</th>
<th>N test cells</th>
<th>N incorrect</th>
<th>Proportion incorrect</th>
<th>Avg dist to actual colour</th>
<th>Avg best dist to other colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3778</td>
<td>565</td>
<td>0.149550</td>
<td>0.999998</td>
<td>1.583324</td>
</tr>
<tr>
<td>3</td>
<td>3778</td>
<td>98</td>
<td>0.026508</td>
<td>1.000000</td>
<td>3.410256</td>
</tr>
<tr>
<td>5</td>
<td>9031</td>
<td>45</td>
<td>0.004983</td>
<td>1.000001</td>
<td>3.124477</td>
</tr>
<tr>
<td>6</td>
<td>2355</td>
<td>3</td>
<td>0.001274</td>
<td>0.999999</td>
<td>3.584034</td>
</tr>
<tr>
<td>9</td>
<td>267</td>
<td>4</td>
<td>0.014981</td>
<td>1.000000</td>
<td>2.781323</td>
</tr>
<tr>
<td>11</td>
<td>1510</td>
<td>23</td>
<td>0.015232</td>
<td>1.000000</td>
<td>2.772248</td>
</tr>
<tr>
<td>12</td>
<td>604</td>
<td>44</td>
<td>0.072848</td>
<td>1.000000</td>
<td>2.148747</td>
</tr>
<tr>
<td>13</td>
<td>1163</td>
<td>53</td>
<td>0.045572</td>
<td>1.000000</td>
<td>2.416676</td>
</tr>
<tr>
<td>14</td>
<td>2523</td>
<td>2</td>
<td>0.000793</td>
<td>0.999998</td>
<td>2.718664</td>
</tr>
</tbody>
</table>

As part of this work, a training application has been developed to facilitate the generation of an appearance model for each robot. As with the obstacle detection model, colour and texture information are extracted from a grid of cells in each training image, and a K-Nearest-Neighbour-like approach creates a set of descriptions of the colours attached to each robot. Again, as with the obstacle detection system, this allows initial classification of training images to make the setup process as simple as possible. The human user can review and correct the initial classification before generating and testing the appearance model. The generated model is output as source code that can be immediately compiled into the executable to be deployed onto a physical robot.

Output from the appearance training application is shown in figure 5.1. As can be seen, robust differentiation between the colours is achieved with basic hue, saturation, intensity and texture features.

Upon capturing an image, the robot divides the image into a grid of cells and calculates a description of each. For each colour description, the similarity to each grid cell is calculated, creating a grid of scores for each cell. From this, blobs are created by joining adjacent cells of sufficient similarity.
Figure 5.1: Results from test system performing blob detection and relative localisation using a set of test images from known relative positions.
with sufficiently good scores. Blobs are then filled outwards to match the edge of each face visible on each robot as closely as possible.

Given a blob of a specific colour, the corresponding RobotFace is identified, i.e. the ID of the visible robot and the side on which this colour is visible. As robots broadcast orientation information via the Coalition Arbiter agent, this can be used as an initial estimate for the expected slopes of the visible edges of each face. As each face is known to have straight edges, a Hough transform is used to fit lines to blob edges, with the initial slope estimate used to cut down the search space and reduce computational overhead on the robots.

Where edges of faces are detected with sufficient robustness, corresponding vectors are translated to world space to provide estimates for the face position and orientation. The training system provides an output of localisation error which is used to generate a covariance for each estimate. A trend observed from these estimates is that horizontal edges typically provide estimates with low error along the normal to the face, while vertical edges provide low lateral error relative to the robot’s orientation. Thus, combining covariances from more than one edge can provide a more accurate localisation estimate. Pseudo-code for this algorithm is presented in figure 5.2.

Relative localisation errors for a sample set of test images are shown in figure 5.3. While relative localisation accuracy given an image of a robot varies based on the number of edges detectable and where in the image the robot is visible, these results demonstrate the typical accuracy that can be expected across a set of random relative locations. As an example, for the physical robot run in the experiments described in section 4.3.3, the overall uncertainty in relative location estimates of its partner robot was calculated by the test system as $\begin{bmatrix} 457.8740 & 112.8287 \\ 112.8287 & 536.4766 \end{bmatrix}$. To obtain these results, a grid had been laid out in the environment and images of robots taken at known relative locations. The test system was then run given a set of structures, each containing a camera image and corresponding relative position information.
FacePose* estimateRobotFacePose(RobotFace *robotFace, Blob *blob, ...)
{
    facePose = initFacePose();

    // Detect top, bottom, left, right edges of blobs
    edges = calculateVisibleEdges(blob);

    for (i = 0; i < 4; ++i)
    {
        if (edges[i].visible)
        {
            edgeInImageSpace = fitHoughLineToEdge(edges[i]);
            edgeLoc = projectEdgeToWorld(edgeInImageSpace);
            edgeCov = estimateErrorForEdge(i, edges[i]);
            facePose = updateCovForFace(i, edgeLoc, edgeCov);
        }
    }
    return facePose;
}

colourScores = calcAppearanceScores(imageGridCells);
for (colourId = 0; colourId < nColours; ++colourId)
{
    robotPose = initRobotPose();

    initialBlobs = groupImageGridCells(colourScores);
    blobs = fillColourBlobs(initialBlobs);

    for (i = 0; i < nBlobs; ++i)
    {
        robotFace = identifyRobotFaces(blobs[i]);
        robotFacePose = estimateRobotFacePose(robotFace, blobs[i]);
        robotPose = updateRobotPose(robotPose, robotFacePose, robotFace);
    }
}

Figure 5.2: Pseudo-code for estimating the relative pose of another robot. Each physical robot has 3 distinct colours. The ObjectRecognition module contains a description of each and can map from a recognised colour to a particular face on a particular robot.
Figure 5.3: Relative localisation errors given a set of images of a robot from known relative poses.
5.2.2 Supervision

The Supervision behaviour is adopted by robots in order to act as artificial landmarks for other exploring robots. This enables other robots to explore the surrounding area and periodically use the landmark to remove error accumulated along their paths. As stated in related work [75], and as backed up by findings in experiments carried out in this work, if the error in robots' localisation estimates is normally distributed, then the error accumulated over a path traversed while closing a loop may be assumed to have been accumulated uniformly over steps in that path.

Collaborative Exploration Coalitions

As with WIDE_AREA_EXPLORATION targets described in section 4.2.5, this behaviour does not itself allow robots to obtain revenue. Instead, it facilitates the generation of revenue by other behaviours, for which it will receive a specific proportion. In this case, the CLOSE_LOOP behaviour allows robots to improve the accuracy of map data submitted to the Map Aggregator Agent, for which it receives revenue.

In order to avoid wasted actions by robots, it is necessary for robots implementing CLOSE_LOOP to be certain that a landmark robot will be available at a specific point. Therefore, it is necessary that an agreement is made between a robot implementing SUPERVISION and one or more robots implementing CLOSE_LOOP that the supervisor robot, which has a known localisation certainty, will remain at the centre of a supervision-area in return for a proportion of revenue from corrected map data.

To this end, adopting Supervision requires that a robot first forms a coalition with one or more robots. The initial proposal to form the coalition includes information as described in figure 5.4. Namely, it describes how much revenue a potential partner can expect to gain by joining the coalition, and how much revenue it will have to attribute to the supervisor.

As discussed in section 3.2.1, the arbiter agent can be furnished with instructions for accepting bids on a proposal and create the resultant coalition without requiring further negotiation between robots. For the experiments
typedef struct {
    int supervisorId;
    Pose supervisorPose; /* Contains accuracy of location estimation */
    Point coalitionRegion;
    time expiryTime;
    float supervisorCost;
} Proposal;

Figure 5.4: A proposal to form a coalition for collaborative exploration posted on the Coalition Arbiter Agent by a prospective supervisor.

described in this work, only a first_price_sealed_bid auction has been implemented. The auction ends either when all robots have viewed the proposal or when a specified duration expires.

For the coalition system implemented in the experiments carried out in this work, bids placed by potential partners will state the proportion of revenue gained by improving map accuracy through cooperative localisation that they are prepared to allocate to the supervisor robot. While it would be an interesting study to determine the effects of different bidding mechanisms on exploration performance, all bidding robots use the same function to calculate bids based on the estimated efficiency with which they partake in the experiment. Efficiency here is calculated in terms of the resources that a robot would have to expend in order to perform map adjustment, explained in section 5.2.3, i.e. \( \frac{p_{ma}}{r_e} \), where \( p_{ma} \) is the estimated gross profit obtainable by adjusting map data and \( r_e \) the resources that the robot would have to expend. The bidding mechanism then simply calculates the ratio \( \frac{r_s}{r_e} \), i.e. the ratio of resources the proposing robot, or supervisor, would have to expend, \( r_s \), to its own resources. Here the supervisor states the estimated resources it will expend over the course of a coalition as the supervisorCost, which is communicated as part of the Proposal structure as illustrated in figure 5.4.

The Supervision behaviour is evaluated alongside other profit-centric behaviours in the behaviourControl.evaluateBehaviours step shown in figure 2.2. When a group of robots is deployed in independent mode, i.e. with no explicit cooperation between robots, then this function simply calls the
calculateProfit of each profit-centric behaviour module present and adopts the behaviour with the highest utility. When the robots are compiled in COLLABORATION mode, however, the algorithm is as depicted in figure 5.5.

**Behaviour Implementation**

As with independent profit-centric behaviours discussed in chapter 4, all logic for the SUPERVISION behaviour is encapsulated within a plug-in module. In this case, the module contains additional functions that are used by the robot control module for the formation of coalitions. The makeProposal and considerProposals functions are implemented within this module and are called from behaviourControl.evaluateBehaviours, as depicted in figure 5.5. Minimal logic is contained in the function itself, as it only compares the profit ratio of all possible behaviours, though it controls the sequence in which proposals and bids can be made. Independent behaviours are first evaluated, with the most profitable independent behaviour then used as a reserve when evaluating the utility of adopting SUPERVISION and considering proposals from other robots. When a robot determines that a bid to join a coalition has been successful, the AVAILABLE behaviour is adopted such that the robot can then re-consider potential targets given its new state.

The implementation of the functions defined by the behaviour interface are as follows:

- **Supervision.calculateProfit** — This behaviour does not enable robots to directly generate revenue but rather to act in a manner that is beneficial to other robots. As the robot can only receive revenue through agreements with other robots, it follows that when evaluating the profitability of this behaviour, a robot will estimate the profitability of a partner robot and from this value calculate the amount of revenue that it itself will be accredited.

As stated in section 5.2.2, the coalition system presented here uses the same heuristic across all robots for calculating bids, thus ensuring that robots are allocated to coalitions based on their suitability, not on the effectiveness of their bidding approach. Bids are calculated as the
Figure 5.5: Evaluation of behaviours in \textsc{collaboration} mode. The estimated profit ratios of adopting independent behaviours, making a proposal to form a coalition and bidding on proposals from other robots are compared.
ratio of supervisor resources to explorer resources, $\frac{r_s}{r_e}$. When evaluating supervision robots make the rough initial estimate that a winning bid will have a ratio of 0.5, i.e. the partner robot will allocate 0.5 of revenue from the close-loop behaviour.

Each supervision area corresponds to a square area of 252 occupancy grid cells in length, i.e. 3 adjacent local maps or 5.04m. As with the environment grids for independent exploration listed in table 2.1, the grid of supervision areas is updated by the Map Aggregator agent. It is aligned to the local map grid such that information about each supervision area can be inferred from the local maps within it.

The utility, or profit ratio, for each supervision target is calculated as shown in figure 5.6. Here, the supervisor simultaneously calculates the estimated profit for potential partners as well as the estimated profit for itself. Potential partners will typically explore $n$ local maps within the supervision area, depending on the exploration parameters specified for the mission, before adopting close-loop to correct errors accumulated in map data.

Within each local map, the local_area_exploration moves are calculated as in section 4.2.5. This gives the number of moves which are assumed to have corresponding map data and thus will allow close-loop to generate revenue from them. The total number of moves that a partner is expected to make is then used to estimate how long the supervisor robot will remain as a stationary landmark, which will involve a cost in terms of time and power expended. Further details on close-loop profit is given in section 5.2.3.

- Supervision.adopt — Upon creating a new coalition, the Coalition Arbiter agent will update all affected robots, i.e. those just allocated to the coalition and those whose coalitions have been ended as a consequence. This means that a robot adopting supervision does not have to synchronise with any other robots and can navigate towards the appropriate supervision area. As with any newly adopted behaviour, the isAtDestination flag for this is initially set to false.
partnerMoves = 0;
localAreaExplMoves = 0;
navData = calculateMovesToSupArea(supArea);
localMapsToExplore = getBestLocalMaps(supArea);

for (i = 0; i < nLocalMapsToExplore; ++i)
{
    localMap = localMapsToExplore[i]
    proportionLocalMapMapped = getProportionMapped (localMap);
    partnerMoves += estimateWideAreaExplMoves(proportionSupAreaMapped);

    do
    {
        nMoves = estimateLocalAreaExplMoves(proportionLocalMapMapped);
        partnerMoves += nMoves;
        localAreaExplMoves += nMoves;

        proportionLocalMapMapped = updateLocalMapMapped(proportionLocalMapMapped, nMoves);
        proportionSupAreaMapped = updateSupAreaMapped(proportionSupAreaMapped, nMoves);
    } while (proportionLocalMapMapped < localMapThreshold);
}

nMoves = estimateCloseLoopMoves(proportionSupAreaMapped);
partnerMoves += nMoves;

closeLoopProfit = estCloseLoopProfit(localAreaExplMoves, navData.errorAtDest);
supProfit = closeLoopProfit * 0.5f;
supCost = calcMoveCost(navData) + calcSupervisionCost(partnerMoves);
supProfitRatio = supProfit / supCost;

Figure 5.6: Pseudo-code for calculating gross profit for a SUPERVISION target.
Supervision.checkState — A supervisor robot’s partners are free to form another coalition at any time. When this occurs the Coalition Arbiter will clear the old coalition structures. Thus, in each iteration of the robot’s control loop, before evaluating profit-centric behaviours, a supervisor will check the state of coalitions on the Coalition Arbiter to determine if its partners have left, in which case the supervisor will adopt available and evaluate behaviours as usual.

Supervision.handleSuccess — The RobotStateData structure maintained by each robot and communicated to the communication hub agent (in the implementation discussed here, this is hosted on the Map Aggregator agent) contains a flag isAtDestination to indicate to other robots that this robot has successfully arrived at the destination specified in the NavigationData for its current behaviour. In the case of SUPERVISION, this flag is used by partner robots to indicate that it is not possible to use the supervisor as an artificial landmark for correcting errors in map data.

Supervision.handleFailure — If it is determined that a robot cannot reach the destination specified by the supervision target, then the robot must leave the coalition and alert its partner robots. The CommunicationCore.writeLeaveCoalition function allows the robot to send a request to the Coalition Arbiter agent asking it to update the set of live coalitions as appropriate and set flags to alert affected robots.

5.2.3 Loop Closing

A central technique in the collaborative exploration system presented in this work is the use of robots as artificial landmarks. This allows a group of low-cost robots to generate a more accurate map over a larger area than would be possible if the same group of robots were to operate independently.

As outlined in section 5.1.1, there are a number of accounts of cooperative localisation in the literature, where relative location estimates are exchanged between multiple robots in order to improve localisation accuracy.
when performing various tasks. In the context of these accounts, the main contributions of the work presented here are the utilisation of extremely low-cost robots and the employment of a market framework to allow flexible, distributed coordination of robots.

The robots considered in this work have an extremely short sensor range, in the region of 0.5m, in addition to limited computational and storage capabilities. These limitations add to the complexity of exploring an environment, as a behaviour-based control architecture is required to allow robots to act effectively while responding to changes in the state of the group or the environment. Thus, using a central coordination system with such robots would compromise the effectiveness of their behaviour-based control schemas and would need to be overly complex to facilitate the various exploration tasks. To this end, the system presented here encapsulates all processes required to coordinate robots within behaviour modules, which robots utilise in a profit-driven manner.

Section 5.2.2 outlines the SUPERVISION behaviour implemented as part of the market-based control system that motivates robots to act as artificial landmarks for other robots. In this section, the complementary behaviour that motivates robots to use such landmarks to correct errors in map data will be presented. The CLOSE_LOOP behaviour prompts robots to assess the utility of navigating to a supervisor robot to update its location estimate and thereby adjust the estimates made along the path it has traversed, in a manner similar to the loop-closing techniques described in section 5.1.1. This behaviour implements the interface required by the plug-in architecture described in section 2.2.2 and thus can be updated or removed from the robot control system with minimal code changes.

The remainder of this section will describe the map adjustment technique and provide details on the behaviour implementation, while results from experimentation are presented in section 5.3.
Removing Error from Map Data

The map adjustment technique utilised within this behaviour is inspired by the loop-closing approaches presented in [75, 71], as discussed in section 5.1.1. The technique makes use of the observation in this work that, upon correcting a large error in a robot’s location estimate, it will be assumed that the error has been accumulated uniformly over the moves made by the robot from an initial known position.

To verify this observation, the close_loop behaviour was implemented to prompt robots to attempt to trace a loop to a position from which their location can be accurately estimated and a post-processing application developed to analyse logs in order to determine the accuracy of the map adjustments carried out.

For the purpose of training a model of the accuracy of the map adjustment technique, the naïve assumption was initially made that a sequence of robot poses adjusted with completely accuracy upon closing a loop. Thus, if a robot starts a mission with a known position, i.e. the offset between the robot’s estimate location and its actual location at time 0, \( o_0 \) is \((0, 0)\), and the robot then determines that the offset after \( n \) moves, \( o_n \), is \((x, y)\), then the offset at each step \( i \) along the sequence can be determined with complete certainty as \((x \frac{i}{n}, y \frac{i}{n})\). Thus, the simulated experiment run with these assumptions would discard initial estimates and allow the adjusted estimates to be compared to the robot’s actual locations.

![Figure 5.7: Calculating the error in estimating a robot’s location at iteration \( n - 1 \) based on the location at iteration \( n \). The offset to the actual location is translated to the coordinate frame aligned along the vector between the two locations.](image-url)
Table 5.2: Errors in location estimates in sequences of robot locations with known starting position. Errors are taken from simulated experiments with 2 robots implementing collaborative exploration.

<table>
<thead>
<tr>
<th></th>
<th>$\bar{x}$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead-reckoning error</td>
<td>17.99874605</td>
<td>8.218613963</td>
</tr>
<tr>
<td>Adjusted error</td>
<td>9.647808699</td>
<td>3.711778781</td>
</tr>
</tbody>
</table>

Given experiment logs written to XML, a post-processing algorithm was provided to analyse the error in each position along a set of such sequences. The algorithm assumes that the robot’s location at the last step in a sequence has been determined by estimating the relative location of a landmark, and that the certainty of the robot’s estimate at that point, as discussed in section 5.2.1, can be modelled as a covariance matrix $\text{cov}_n$. It is also assumed that the offset from the estimated to the actual location, $o_n$, is known exactly. The algorithm proceeds by calculating the change in offset from each location $l_n$ to the previous location $l_{n-1}$, as illustrated in figure 5.7. For each pair of locations, the difference in offset, i.e. $(x_{n-1} - x_n, y_{n-1} - y_n)$, is translated from global coordinates to the coordinate frame aligned along the vector from $l_{n-1}$ to $l_n$.

Thus, as the process of adjusting the sequence of locations calculates $l_{n-1}$ from $l_n$, this algorithm outputs the errors from such a sequence. The covariance of the errors is then calculated, which the application outputs as source code to be included in the robot control executable. As with all parameters that are specific to the robot hardware in use, the appropriate parameters are selected at pre-compile time based on arguments to make.

Given the expected accuracy of adjustments made to a sequence of locations, the algorithm to perform a loop-close then updates each location by combining the initial and adjusted estimate according to the magnitude of error in each and updates the resultant covariance accordingly.

In simulation, using calibration models from the robots described in the appendix, the technique was found to give a sizeable improvement in accuracy over locations estimated from dead-reckoning alone. For example, for sequences of robot locations from a known starting location, typically
over 100 location estimates in length, locations estimated by dead-reckoning alone would typically have an average offset of 17.99cm, while the offset for locations estimated using the map adjustment technique was 9.64cm.

An example of loop-closing performed in a simulated experiment is shown in figure 5.8. While in a coalition, map data is submitted to the Map Aggregator agent, but marked as preliminary. When a robot carries out a loop-close and submits map data for the same sequence marked as updated, the Map Aggregator is instructed to wipe the preliminary map data. The submitting robot is obliged to relinquish all revenue earned for the preliminary data before earning revenue for the adjusted data.

Figure 5.8: Example of map-adjustment using collaborative exploration. Two robots form a coalition to explore an area and then remove error accumulated in the collected data.

Reconnaissance

When attempting to achieve visual contact with a supervisor robot, an explorer will frequently determine that it has navigated successfully to its destination pose but its supervisor is not visible due to error accumulated in its location estimate. If this occurs, the robot may be left with no supervisor and a large accumulated error rendering it incapable of exploring further.
Table 5.3: Maps generated using independent exploration. As explained in section 3.2.2 the map error metric used here is calculated based on the area of the error ellipse for $1\sigma$ for the robot’s location estimate covariance. The area mapped is specified as a proportion of the entire environment.

<table>
<thead>
<tr>
<th>N robots</th>
<th>Avg map error</th>
<th>Area mapped</th>
<th>N moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9.4741</td>
<td>0.1764</td>
<td>475</td>
</tr>
<tr>
<td>4</td>
<td>9.3491</td>
<td>0.3432</td>
<td>848</td>
</tr>
<tr>
<td>8</td>
<td>9.1655</td>
<td>0.6440</td>
<td>1693</td>
</tr>
</tbody>
</table>

Therefore, in this event the close_loop behaviour’s handleFailure function will instruct the robot that it must switch to the reconnaissance behaviour.

This behaviour will prompt the robot to follow a path around the supervisor’s location until visual contact with it has been established. At each point on the path the robot performs a full rotation. The radius of the first path corresponds approximately to the robot’s sensor range, and points along the path are approximately twice this distance from each other. Thus, the robot achieves complete coverage of the area, ensuring that it will eventually achieve visual contact and perform cooperative localisation. Upon completion, the behaviour is popped off the behaviour stack and close_loop is resumed.

5.3 Collaborative Exploration Results

This section presents results from experiments implementing collaborative exploration, along with a comparison to maps generated using independent exploration. Maps are compared to those generated with independent exploration to demonstrate the gains in accuracy and area mapped that collaboration can offer.

Figures 5.9, 5.10 and 5.11 compare maps generated in independent and collaboration mode over the same number of iterations for 2, 4 and 8 robots respectively. That is, collaboration experiments are limited to the number of iterations achieved in independent experiments before robots are rendered inoperable due to accumulated localisation error.
Table 5.4: Maps generated using collaborative exploration. Experiments were limited to the number of iterations achieved in the corresponding independent exploration experiment listed in table 5.3.

<table>
<thead>
<tr>
<th>N robots</th>
<th>Avg map error</th>
<th>Area mapped</th>
<th>N moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.4066</td>
<td>0.1066</td>
<td>221</td>
</tr>
<tr>
<td>4</td>
<td>3.8797</td>
<td>0.1656</td>
<td>434</td>
</tr>
<tr>
<td>8</td>
<td>4.0723</td>
<td>0.3587</td>
<td>946</td>
</tr>
</tbody>
</table>

As with the independent experiments described in section 4.3, error magnitude is calculated based on the area of the $1\sigma$ error ellipse of the robot's location estimate, specifically the value of $area/\pi$ is used. For clarity's sake, the experiments described here measure distances in terms of occupancy grid cells. When applied to real environments, these cells correspond to an area of $4\text{cm}^2$. The simulated environment used in these experiments is 504 by 504 map cells, thus 101,6064m$^2$.

Due to the use of robots as stationary artificial landmarks, it is expected that the area mapped per iteration will be less when collaborating than when operating independently. However, the area mapped per robot move is comparable in both modes. Thus, if experiments are limited by robot battery power and not by time, then collaboration will not impose any reduction in the area mapped.

In these experiments, the maximum permitted localisation uncertainty when submitting map data is 20 map units$^2$. As explained in section 3.2.2 this value corresponds to $area/\pi$ for the $1\sigma$ error ellipse for the robot's location estimate covariance.

As illustrated in table 5.3, when exploring in independent mode, the average uncertainty in generated maps is uniformly half the maximum permitted uncertainty. This occurs because the robots do not exhaust potential exploration targets before accumulating the maximum localisation error. The proportion of the environment mapped also grows linearly with $n$ robots. The local_area_exploration behaviour repels robots in close proximity from attempting to explore the same targets, while wide_area_exploration efficiently disperses robots through the environment.
Figure 5.9: Comparison of maps generated with 2 robots using independent and collaborative exploration. Here, both experiments are limited to 293 iterations. Collaborative exploration facilitates the formation of coalitions between robots that allows them to reduce the error in generated maps, but at the cost of the rate at which map data can be generated. Parameters of the generated maps are given in tables 5.3 and 5.4.
Figure 5.10: Comparison of maps generated with 4 robots using independent and collaborative exploration. Both experiments are run for 270 iterations. With collaborative exploration, 0.1656 of the environment is explored with an average error magnitude of 3.8797. With independent exploration, a larger proportion of the environment is explored, 0.3432, but with on average 3 times the error in generated map data.
Figure 5.11: Comparison of maps generated with 8 robots using independent and collaborative exploration over 279 iterations. Here, the use of collaboration gives a reduction in the average error magnitude in map data from 9.1655 to 4.0723.
Table 5.4 shows results where robots employ collaborative exploration to form coalitions while mapping. As the number of iterations in each experiment was limited to that achieved in the corresponding independent experiment, an expected drop in area mapped is observed. At any time during an experiment with \( n \) robots, there may be between 0 and \( n/2 \) robots acting as stationary landmarks. This is reflected in the total number of moves made by all deployed robots, with the area mapped per move in line with that obtained through independent exploration. The maps generated show a dramatic increase in accuracy, however. This is caused both by the reduced number of moves made over the course of the experiment and the periodic correction of accumulated error through cooperative localisation. Figures 5.12 and 5.13 show accumulated error in robots' location estimates in independent and collaboration experiments, with distances given in occupancy grid units. The offsets reported in figure 5.13 correspond to the robots' estimated locations as they explore, not the corrected estimates after performing loop-closing. Thus, it can be seen that error is accumulated at an equal rate in both cases, but with the robots being able to dramatically reduce this error using cooperative localisation.

While the experiments discussed thus far compare independent and collaborative exploration over the same number of iterations, this does not give a fair representation of the increased exploration range that collaboration provides. Figures 5.14, 5.15 and 5.16 compare maps generated with independent and collaborative exploration where experiments are terminated when all robots have accumulated sufficient localisation uncertainty such that they can no longer perform tasks of any utility.

The parameters of the generated results are shown in table 5.5. By comparing these values to table 5.3 it can be seen that collaboration prolongs the duration over which robots can explore. The area explored for each \( n \) robots surpasses that explored using independent exploration, while the average uncertainty encoded in the generated maps is reduced by between 23 and 26 percent.

Figure 5.17 shows the errors in the location estimates from loop-closures in the experiment depicted in figure 5.14. In the experiment, 2 robots perform-
Figure 5.12: Errors in location estimates in an experiment with 2 robots running in INDEPENDENT mode. Distances between actual and estimated location are given in occupancy grid units.

Table 5.5: Maps generated using collaborative exploration, corresponding to figures 5.14, 5.15 and 5.16. Experiments were not limited to a number of iterations, but rather terminated when all robots had accumulated localisation uncertainty surpassing a specified threshold.

<table>
<thead>
<tr>
<th>N robots</th>
<th>Avg map error</th>
<th>Area mapped</th>
<th>N moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7.1735</td>
<td>0.2475</td>
<td>765</td>
</tr>
<tr>
<td>4</td>
<td>7.2107</td>
<td>0.4251</td>
<td>1322</td>
</tr>
<tr>
<td>8</td>
<td>6.7680</td>
<td>0.6654</td>
<td>2290</td>
</tr>
</tbody>
</table>
Robot localisation error with collaborative exploration

Figure 5.13: Errors in location estimates in an experiment with 2 robots running in collaboration mode. Here, error distances are given between robots’ initial location estimates and their actual locations. Accumulated error is periodically removed using cooperative localisation. Using the map adjustment technique described in section 5.2.3, the error in the actual map data gathered between loop-closes is further reduced.

Table 5.6: Loop-closes in collaborative exploration experiments. The reduction in average error reduction evident for 8 robots is a result of the exhaustion of contiguous areas of unexplored terrain. As the robots approach complete exploration of the environment, exploring robots are forced to target smaller areas of unexplored terrain over shorter distances.
Figure 5.14: Comparison of maps generated with 2 robots using independent and collaborative exploration. In both cases, the experiment is run until all robots have accumulated localisation uncertainty surpassing the stated threshold.

Figure 5.15: Comparison of maps generated with 4 robots using independent and collaborative exploration.
(a) Map generated using independent exploration.

(b) Map generated using collaborative exploration.

Figure 5.16: Comparison of maps generated with 8 robots using independent and collaborative exploration.

Using collaborative exploration generate a map covering 0.24 of the simulated environment. While this is 40% more than 2 robots using independent exploration could cover, collaboration also provides a reduction in uncertainty in the generated map. An interesting trend noticeable in this data is that the error in adjusted scans does not grow unbounded as the experiment progresses. Figure 5.18 shows the reduction in errors in map scans relative to the estimates before loop-closing. As odometry error by its nature is unpredictable, it can occur that error is accumulated in a particularly non-linear fashion, resulting in an actual increase in error in adjusted map scans. Loop 4 in this figure increases the offset from the actual location of a number of scans. However, the typical trend evident from this figure is that loop-closing provides effective removal of localisation error as it is accumulated by exploring robots.
Figure 5.17: Errors in adjusted map scans after loop-closing performed in a collaborative exploration experiment with 2 robots.
Figure 5.18: Reduction in errors in map scans adjusted by loop closing. The reductions compare the values shown in figure 5.17 to the original offsets.
5.3.1 Experiments with Physical Robots

As described in section 4.3.3, many technical issues arose when experimenting with physical robots over larger areas due to the use of low-cost off-the-shelf components. While cooperative localization itself had been thoroughly tested and verified, as illustrated in section 5.2.1, the interference with global orientation measurements meant that the large error accumulated by robots when navigating made utilisation of collaborative exploration over large areas impossible. While experiments demonstrating mapping of areas in the region of $100m^2$ were carried out in simulation, physical robots were limited to smaller areas, with robots typically able to performing loop closing only once before accumulating error greater than the permitted threshold and becoming immobilised. Figure 5.19 shows snapshots from an experiment with one robots performing LOCAL_AREA_EXPLORATION while in a coalition with a supervisor robot, and adopting CLOSE_LOOP to establish visual contact with the supervisor in order to update its location estimate and remove error from map data it has gathered.

5.4 Summary of Contributions

In this chapter, a novel visual cooperative localisation approach was presented in section 5.2.1. This approach allows accurate relative range and orientation measurements to be made between robots using low-quality, low-resolution images, while the appearance-based training and testing system means the technique can be easily applied to recognition of different objects.

Sections 5.2.2 and 5.2.3 describe robot behaviours that facilitate a distributed, robust collaborative exploration technique. The technique allows low-cost robots to cooperate in order to greatly increase their ability to explore an environment in terms of the quality of the map generated and the size of the area which can be explored, while allowing robots to operate in a completely distributed and autonomous manner. Additionally, the basis within a market-framework allows the technique to be employed in a flexible manner alongside other tasks.
Figure 5.19: Images from an experiment with physical robots carrying out LOCAL_AREA_EXPLORATION and CLOSE_LOOP while in a coalition.
Section 5.3 presents results from experiments employing collaborative exploration. Efficiency of mapping and scale and accuracy are compared for collaborative and independent exploration. It is shown that employing collaboration can provide benefits in all these respects, with the use of sensor data through cooperative localisation effectively compensating for the accumulation of localisation error.
Chapter 6

Conclusion

This work has examined a market-based multi-robot exploration framework. The framework is aimed at enabling low-cost robots with poor sensors, odometry and processing power to generate an accurate map of a large novel environment, while also being adaptable to additional tasks. This work provides a number of contributions in multi-robot exploration, which are concluded in the remainder of this chapter.

6.1 A Lightweight, Extensible Behaviour-Based Robot Control Architecture

The robot control architecture has been demonstrated in simulation and in real-world experiments on robots with low-powered embedded processors. It has been demonstrated on different processor architectures, and can be built and run on further platforms with minimal or no code changes.

A number of behaviours performing independent and collaborative exploration have been implemented via the plug-in behaviour interface. Experiments have demonstrated that robots can switch between behaviours as appropriate, while a behaviour stack coherently maintains robot state at all times.

The modular design means functionality interfacing with sensors or actuators can be adjusted to work on different physical platforms. Experiments
have been run on three physical robots with two distinct robot platforms, while sensor and movement calibration settings can be cleanly plugged-in for each robot and selected at pre-compile time for optimum efficiency. Comprehensive logging of experiments (where verbosity can additionally be adjusted as required) allows quick analysis of performance. Applications to analyse log files and present statistics are provided as part of the framework, as can be seen in the graphs presented throughout this thesis.

A simulation framework built around the robot control code has been demonstrated. Code run in simulated frameworks differs from that run on physical robots only in terms of communication, performing movements and capturing sensor data, and as such has been shown to support replaying of physical experiments in order to debug problems and optimise behaviours. Additionally, the clean interface presented by functions in behaviour modules means that computationally expensive operations can be sent to other agents and the results returned, allowing low-cost robot to be demonstrate advanced capabilities while still remaining autonomous.

6.2 Exploration Technique for Low-Cost Robots

Experiments, both simulated and real-world have demonstrated an exploration technique allowing low-cost robots with off-the-shelf sensors and actuators to explore and map a novel environment.

The technique includes accurate path calculation for motion-constrained robots which increases efficiency when exploring in terms of the area mapped per movement made and facilitates accurate navigation necessary for cooperative localisation.

Generating an occupancy map from low-cost monocular visual sensors has been demonstrated. The appearance-based image processing technique provides robust results for detecting obstacles and calculating the relative location of visible robots. A user-friendly application is provided to enable quick training and verification of appearance descriptions for new environ-
ments or objects of interest. This allows quick integration of generated models into robot code, where output is generated as source code which can be immediately compiled and run on robots. The mapping technique supports consistent pose estimation, allowing accuracy in map data to be improved when cooperative localisation is performed.

Generation of a consistent map from multiple robots has been demonstrated through a Map Aggregator agent. This allows each robot to maintain an up-to-date representation of the environment at all times and avoids duplicated effort across robots by carrying out analysis on map data and generating information for path planning.

6.3 A Market-Framework for Collaborative Exploration with Autonomous Robots

While the robot control architecture allows efficient exploration with individual robots, it also supports collaboration between robots. Efficient formation of coalitions between robots has been demonstrated using a Coalition Arbiter agent. This accepts instructions and rules from robots wishing to initiate coalitions, thus allowing robots to collaborate without any centralised control. The communication overhead on robots is reduced by having each robot only communicate with the Coalition Arbiter.

Collaborative exploration has been implemented through behaviour modules, and thus can be swapped out of the robot control code or updated as required. This technique allows robots that would otherwise be prone to inaccurate localisation to remove accumulated error and thus explore larger areas. While the requirement that some robots remain stationary means that the rate at which map data is generated by the whole team is reduced, the rate per robot is in line with independent exploration. Collaborative exploration has been shown to result in greater accuracy and greater area mapped in exploration experiments.
6.4 Publications


Appendix A

Low-Cost Robot Hardware

As part of this research 3 physical robots were built from off-the-shelf components. These robots were used to determine the accuracy of dead-reckoning and the quality of sensor data that could be achieved with low-cost robots. Determining dead-reckoning accuracy was vital in developing a simulation framework to accurately model real-world exploration experiments. These robots were also used to run physical experiments in the environment depicted in section 4.3.3, but due to severe interference experience in compass measurements were not able to fully verify the collaborative exploration technique.

The robots used in testing were built using the following hardware components:

- Gumstix Waysmall Computer-on-module — This comes with an Intel-/Marvell PXA255 processor [135] and a Bluetooth class 1 receiver with approximately 10m range and 80kbit/s application data throughput. In testing, robots were setup to communicate over a piconet with one master and up to 3 slaves. Here the master was run on a PC running the Map Aggregator agent.

- Rogue robotics ATR base [133] — As described in section 4.2.4 the motors imposed severe constraints on the granularity of movements that could be carried out. The minimum rotations that could be made were typically in the order of 40 degrees, while the minimum forward
movement that could be made were in the order of 50mm. Calibration results for one such robot are shown in figure B.2.

- CMU Cam 2 monocular camera — These cameras are capable of providing images at resolutions of 352 by 286 or 176 by 143. However, camera images are grabbed via a 115,200 baud serial connection, i.e. 14,400 bytes/s. As an RGB image of 167 by 143 pixels is 75,504 bytes in size, grabbing an image takes 5.2 seconds. As it is necessary to avoid motion blur, the robots only grab images when stationary. In order to avoid obstacles and properly plan actions the robot control loop must grab and image, process it, determine actions and then perform these actions. While the control system can make use of some of this time by communicating with the Map Aggregator or Coalition Arbiter agent, this still adds a large latency to each iteration.

- Graupner 2500mAh 2s 20C LiPo batteries — Supporting 15C continuous discharge (37.5A) and 20C (50A) burst discharge.

- Honeywell HMC6352 Digital Compass — While the heading accuracy of these modules was stated at 3.5 degrees in the documentation, during calibration it was found to be 4.1 degrees. Additionally, as shown in log output from physical experiments in section 4.3.3, arbitrarily large changes in the orientation measurement were experienced when moving between different areas of the environment.
Appendix B

Robot Calibration

To expedite the calibration process a set of Python scripts were written and sit next to the robot and simulation c / c++ code. A scripting language such as Python is suited to file access and manipulation of arrays of data. As this code did not have to run in real time performance was not a concern and all scripts are capable of converting input data to output in less than a minute. To calibrate robot motion, robots were positioned on a demarked flat surface and sets of movements carried out. Two points were marked either side of each robot and for each movement the start and end position of two points were recorded. From this, the centre-of-gravity position and orientation could easily be calculated.

Due to the limited sensor range available to the robots, it was determined that keeping the maximum move distance within this range would be prudent — allowing robots to avoid collisions with other robots or with obstacles. Sets of movements were carried out in steps from 0 to the maximum move distance with the position recorded at each point and written into a Python dictionary as illustrated in figure B.1. For each move, the delta position and orientation relative to the starting position is then calculated by the script, and variance in each direction and covariance calculated. Sets of moves were made for 0, 50, 100, 50ms, etc. Sets of rotations movements were also made allowing calculation or orientation changes and position offset.

Figure B.2 shows example output from the calibration scripts. Input
class RobotMotionData:
    def __init__(self):
        ...

    self.fwdMoves = {
        0: {  # Engine burst of 0 ms
            'accumd': True,
            'locs': [
                ((0, 66), (0, -65)),
                ((37, 66), (39, -65)),
                ((81, 67), (86, -65)),
                ((126, 67), (133, -64)),
                ((177, 70), (183, -61)),
                ((219, 73), (230, -58)),
                ((262, 76), (277, -54)),
            ],
        },
        {  # Engine burst of 10 ms
            'accumd': True,
            'locs': [
                ((0, 66), (0, -65)),
                ((52, 66), (54, -65)),
                ((103, 67), (109, -64)),
                ((151, 68), (165, -63)),
                ((209, 71), (220, -60)),
                ((267, 74), (276, -56)),
            ],
        },
        ...,
    ],
    50: {  # Engine burst of 50 ms
        {  # Engine burst of 10 ms
            'accumd': True,
            'locs': [
                ((0, 66), (0, -65)),
                ((72, 67), (75, -64)),
                ((132, 68), (149, -63)),
                ((214, 70), (223, -60)),
            ],
        },
        ...
    }

Figure B.1: Example raw calibration data. Moves are made for a particular engine burst, i.e. how long the motors are turned on for. This allows mapping of engine burst to distance in addition to calculation of corresponding error.
measurements are processed in python and then written to a .h file. The calibration script additionally updates the build makefiles such that this file is included for a particular robot, e.g. a condition is written into the makefile so that this .h will be included when the instruction make robot 2 is given to build the robot control executable for robot platform number 2.
```c
#if ROBOT_PLATFORM == 2
#define ROT_LEFT_DELTA_ORIENT 0.828660059139694f
#define ROT_LEFT_DELTA_FWD_WORLD 1.038863566825077f
#define ROT_LEFT_DELTA_LAT_WORLD 1.83134279207705f
#define ROT_LEFT_VAR_ORIENT 0.004111054075077f
#define ROT_LEFT_VAR_FWD_WORLD 16.446537677340324f
#define ROT_LEFT_VAR_LAT_WORLD 2.631394285915838f
#define ROT_LEFT_COVAR_WORLD 3.769846262466532f
#define ROT_RIGHT_DELTA_ORIENT -0.844227778449122f
#define ROT_RIGHT_DELTA_FWD_WORLD -3.466562388591270f
#define ROT_RIGHT_DELTA_LAT_WORLD 1.353442303175682f
#define ROT_RIGHT_VAR_ORIENT 0.001488083948824f
#define ROT_RIGHT_VAR_FWD_WORLD 1.322516921503631f
#define ROT_RIGHT_VAR_LAT_WORLD 0.455641616827991f
#define ROT_RIGHT_COVAR_WORLD 0.397285147314804f
#define ROT_AVG_DELTA_ORIENT 0.836443918794408f
#define N_ROTATIONS_FOR_180 5
#define N_ROTATIONS_IN_TOTAL (1 + 2 * N_ROTATIONS_FOR_180)
#define BASE_MOVE_DELTA_ORIENT 0.001926266265142f
#define BASE_MOVE_DELTA_FWD_WORLD 58.007307886548077f
#define BASE_MOVE_DELTA_LAT_WORLD 1.0373398258521078f
#define BASE_MOVE_VAR_ORIENT 0.000095927126446f
#define BASE_MOVE_VAR_FWD_WORLD 6.58728676231638f
#define BASE_MOVE_VAR_LAT_WORLD -0.268971069608106f
#define MOVE_US_DELTA_ORIENT -0.000000083391470f
#define MOVE_US_DELTA_FWD_WORLD 0.000337470627025f
#define MOVE_US_DELTA_LAT_WORLD 0.000011529341924f
#define MOVE_US_VAR_ORIENT 0.000000002895523f
#define MOVE_US_VAR_FWD_WORLD 0.000084228961570f
#define MOVE_US_VAR_LAT_WORLD 0.00009243601086f
#define MOVE_US_COVAR_WORLD 0.000010679439444f
#endif
```

Figure B.2: Example calibration results for a physical robot.
Appendix C

Simulation of Robot Motion Error

Given the representation of the probability distribution of a robot’s state as a covariance matrix, error in the robot’s state estimation can be simulated using the Box Muller method. This method allows the generation of normally distributed values from independent uniformly distributed random values. The standard normal distribution is described by the function

\[
\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy
\]  

(C.1)

If \( U_1, U_2 \) are independent random variables from a uniform distribution in the interval \((0, 1)\), then \((X_1, X_2)\) as written below will be independent values from a zero mean normal distribution with unit variance.

\[
X_1 = (-2 \ln U_1)^{\frac{1}{2}} \cos 2\pi U_2 
\]  

(C.2)

\[
X_2 = (-2 \ln U_1)^{\frac{1}{2}} \sin 2\pi U_2 
\]  

(C.3)

These can be rearranged as

\[
U_1 = e^{-\frac{(X_1^2 + X_2^2)}{2}} 
\]  

(C.4)

\[
U_2 = -\frac{1}{2\pi} \arctan \frac{X_2}{X_1}
\]  

(C.5)
The angle given by $\arctan \frac{X_2}{X_1}$ is uniformly distributed over the interval $(0, 2\pi)$. The square of the length of the radius vector $X_1^2 + X_2^2$ has a Chi squared distribution, i.e. the sum of the squares of a set of independent normal random variables, and from the equation above it is equal to $-2 \ln U$. From the above equations the joint density of $X_1, X_2$ can be written as follows to verify that they are independently distributed

\[
f(X_1, X_2) = \frac{1}{2\pi} e^{-\frac{(X_1+X_2)^2}{2}} = \frac{1}{2\pi} e^{-\frac{X_1^2}{2}} \cdot \frac{1}{2\pi} e^{-\frac{X_2^2}{2}} = f(X_1) f(X_2) \quad (C.6)
\]
Bibliography


