DEVELOPMENT OF AN ARTIFICIAL INTELLIGENCE SYSTEM FOR THE INSTRUCTION AND CONTROL OF COOPERATING MOBILE ROBOTS

BY

PRANEEL CHAND

A thesis

submitted to the Victoria University of Wellington

in fulfilment of the requirements for the degree of

Doctor of Philosophy

Victoria University of Wellington

(2011)

Abstract

This thesis focuses on the development of an artificial intelligence system for a heterogeneous ensemble of mobile robots. Many robots in the ensemble may have limited processing, communication, sensing, and/or actuation capabilities. This means that each robot may not be able to execute all tasks that are input to the system. A hierarchical system is proposed to permit robots with superior processing and communication abilities to assign tasks and coordinate the less computationally able robots. The limited processing robots may also utilise the resources of superior robots during task execution. Effective task allocation and coordination should result in efficient execution of a global task. Many existing approaches to robot task allocation assume expert knowledge for task specification. This is not ideal if a non-expert human user wants to modify the task requirements.

A novel reduced human user input task allocation and feedback coordination technique for limited capability mobile robots is developed and implemented. Unlike existing approaches, the presented method focuses on expressing tasks and robots in terms of processing, communication, sensing, and actuation physical resources. This has the potential to allow non-expert human users to specify tasks to the team of robots. Fuzzy inference systems are utilised to simplify detailed robot information for comparison with simple human user inputs that represent task resource requirements. Like many existing task allocation methods, a greedy algorithm is employed to select robots. This can result in suboptimal task allocation. In addition to this, the non-expert user's task specifications might be erroneous in some instances. Hence, a feedback coordination component monitors robot performance during task execution.

In this thesis, a customised multi-robot mapping and exploration task is utilised as a model task to test the effectiveness of the developed task allocation and feedback coordination strategy. Extensive simulation experiments with various robot team configurations are executed in environments of varying sizes and obstacle densities to assess the performance of the technique. Task allocation is able to identify suitable robots and is robust to selection weight variation. The task allocation process is subjective to fuzzy membership function parameters which may vary for different

users. Feedback coordination is robust to variation in weights and thresholds for failure detection. This permits the correction of suboptimal allocations arising from greedy task allocation, incorrect initial task specifications or unexpected failures. By being robust within the tested limits, weights and thresholds can be intuitively selected. However, other parameters such as ideal achievement data can be difficult to accurately characterise in some instances.

A hierarchical hybrid deliberative-reactive navigation system for memory constrained heterogeneous robots to navigate obstructed environments is developed. Deliberative control is developed using a modified version of the A* algorithm and a rectangular occupancy grid map. A novel two-tiered path planner executes on limited memory mobile robots utilising the memory of a computationally powerful robot to enable navigation beyond localised regions of a large environment. Reactive control is developed using a modified dynamic window approach and a polar histogram technique to remove the need for periodic path planning.

A range of simulation experiments in different sized environments is conducted to assess the performance of the two-tiered path planning strategy. The path planner is able to achieve superior or comparable execution times to non-memory constrained path planning when small sized local maps are employed in large global environments. Performance of hybrid deliberative-reactive navigation is assessed in a range of simulated environments and is also validated on a real robot. The developed reactive control system outperforms the dynamic window method.

Acknowledgements

I wish to express my sincere gratitude to the many people who have helped me during this project. First, I would like to thank my supervisor Professor Dale Carnegie for his invaluable advice and encouragement. I am also very grateful for his time and effort in reviewing papers and thesis chapters.

A number of past graduate students are deserving of thanks for fruitful discussions, assistance and support. In particular, I would like to thank Dr. Christopher Lee-Johnson for providing feedback, contributing ides and collaborating on a conference publication. Other students who have assisted me are Jason Edwards, Luke Cawley, David Williamson and Thomas Roehr. I also appreciate the technical assistance provided by Scott Forbes and Bruce Rhodes.

I am also grateful to my mum for proof reading draft chapters of this thesis and continuously pushing me to get this thesis submitted as quickly as possible. Mohammed Faizal's assistance in compiling the thesis chapters into a single document is also appreciated. I am also grateful to the University of the South Pacific for funding to attend the oral examination.

Finally, I wish to thank the New Zealand Vice-Chancellors Committee (NZVCC) for awarding me a Commonwealth PhD Scholarship to undertake the research presented in this thesis.

Table of Contents

List of Figures

List of Tables

1 Introduction

Cooperative robotic behaviour independent of human intervention is an active area of mobile robotic research. Ideally, a human should only provide the initial command to a team of robots that then decide for themselves how to execute the given task.

There are several advantages to cooperative behaviour in multi-robot systems. By working in parallel, multiple robots can increase efficiency and reduce the time required to complete a task. Reliability is increased by introducing redundancy when using a team of robots, while cost can be reduced due to the use of smaller simplistic machine designs. Application specific design and manufacturing costs can be reduced by fabricating semi-generic robots. New complex tasks can be introduced to a team of robots which are difficult for a single robot to achieve.

Multi-robot systems can be homogeneous or heterogeneous. Homogeneous systems consist of robots with identical hardware and software elements. This provides good robustness to individual robot failure. However, the robots are required to be generalists that can perform any type of task given to team. It can be expensive to manufacture a team of generalists. Hence, many conventional homogeneous systems are limited to simplistic robot designs that employ detailed human user inputs for control.

Similar to homogeneous systems, heterogeneous systems can be robust to individual robot failure. But, heterogeneous systems comprise robots with non-identical hardware and software elements. Hence, the thesis presented here is that a heterogeneous ensemble of mobile robots can be hierarchically organised with task feedback control, which significantly reduces the need for human user input.

Three broad categories of multi-robot system applications are transportation, sensing, and foraging. Object transportation involves multiple robots transporting objects from one location to another and has been exhibited in robot soccer teams [1-3] as well as in object pushing [4, 5] and object lifting and carrying [6, 7]. Cooperative sensing develops a group robotic system for localisation, map building, and exploration [8-

11]. In foraging, groups of robots must locate and move objects scattered in an environment to a storage location [12-14].

A variety of mobile robots are currently under development at Victoria University of Wellington (VUW). Amongst these is a pair of functionally equivalent tricycle robots for investigating cooperative behaviour such as object transportation or sensing [15]. Another two robots, Rubble-bot [16] and Tank [17], are being developed as part of an urban search and rescue (USAR) multi-robot application [18].

The VUW USAR hierarchical heterogeneous multi-robot system has three categories of robots: grandmothers, mothers, and daughters. At the top of the hierarchy, the grandmothers are physically the largest and most computationally powerful. Grandmother robots are generally employed to manage the operation of a group task. They achieve this by monitoring and coordinating lower tiered robots (mothers and daughters). Mothers and daughters are smaller in size and less computationally powerful. They are also limited in their sensing and actuation abilities. The smaller size of the mother and daughter robots enables them to be deployed for searching the environment as worker robots.

Coordinating a team of mobile robots such as the VUW USAR system usually involves implementing task allocation and coordination mechanisms. Task allocation mechanisms address the question: “which robot should execute which task?” [19] Coordination mechanisms enable the actions performed by each robot to take into consideration the actions of the other robots in the team resulting in coherent team operation [20]. Recently research in multi-robot systems has also addressed coalition formation, the organising of multiple robots into temporary subgroups to accomplish an assigned task that would otherwise be impossible to complete [13, 21].

In certain multi-robot applications, such as exploration (section 2.9), predefined task allocation and coordination may not always work as desired. This is due to the inability to model all aspects of a robot’s interactions with the environment prior to task execution. Task allocation may also fail if tasks are incorrectly specified. Robots with limited capabilities also present the challenge of using resources effectively to achieve the objectives of the group task.

Allocating tasks to robots in a team like the VUW USAR system requires a strategy that takes into account the physical capabilities (i.e. resources) of the different robots. Generally, the resources present on a robot may be classified into four broad categories: processing, communication, sensing and actuation.

Most of the existing task allocation methods require expert knowledge to specify tasks to a team of robots. None of the methods reviewed in section 2.7 attempt to specify tasks in terms of the four broad physical capability categories. Specifying tasks in terms of these physical capability categories may have the potential to allow non-expert users to intuitively select task requirements.

After initial task allocation, robots may not perform as desired due to the inability to fully model all interactions with the environment accurately. Additionally, existing task allocation algorithms (section 2.7) employ heuristic greedy methods to select robots which can be suboptimal. It may also be possible for a user to inaccurately specify tasks when using the four broad physical capability categories. These problems need to be mitigated by employing a feedback mechanism that monitors task execution. Task execution can be classified into four broad categories: planning, communication, sensing and actuation. Hence, a feedback mechanism may be designed in terms of these four categories to detect and correct abnormal performance. Consequently, a group task should be executed with increased efficiency.

Using limited capability robots also presents challenges in task execution. In exploration tasks, robots are required to navigate beyond localised regions of an environment. Memory constrained robots in a heterogeneous system require a hybrid of deliberative and reactive control to achieve this. Deliberative control should be able to provide a path to travel beyond the localised region of the environment. A limited memory robot may be able to utilise the resources of computationally powerful robots to achieve this. Reactive control must facilitate collision avoidance and modify the path of travel when obstacles are encountered during movement.

1.1 Objectives

In this thesis, the aim is to develop an artificial intelligence system to instruct and control a heterogeneous ensemble of mobile robots. Motivated by the VUW USAR hierarchical system that is under development, the robots will have a variety of processing, communication, sensing and actuation resources. Many of the robots will have limited processing, sensing, and actuation capabilities.

One of the specific objectives is to develop a task allocation strategy based on physical robot capabilities. Tasks and robots will need to be specified in terms of the four broad physical capabilities (processing, communication, sensing and actuation). Specifying tasks in terms of the four major physical capabilities should allow non-expert human users to use the task allocation strategy.

Existing task allocation methods employ heuristic greedy techniques which can often be suboptimal. Developing an optimal mapping of tasks to robots is NP-hard [22]. Therefore, another major objective is to develop a feedback coordination strategy that monitors robots during task execution. Task execution performance will be characterised in the broad categories of planning, communication, sensing and actuation. By detecting and correcting failures, the feedback system should facilitate improved group task execution.

A conceptual diagram of the task allocation and feedback coordination mechanism (also motivated by the VUW USAR system) is shown in Figure 1.1. There are three levels of control. At the highest level (level A) a remote base station computer is used to specify the group task (management and worker tasks) and robots. Depending on the management task requirements and robot capabilities, the remote computer identifies manager robot(s) (level B). The manager robots are delegated the responsibilities of global data maintenance, worker task devolution and performance monitoring. After manager task devolution, the remote computer is no longer required by the robot team since task management is essentially transferred to the manager robot(s).

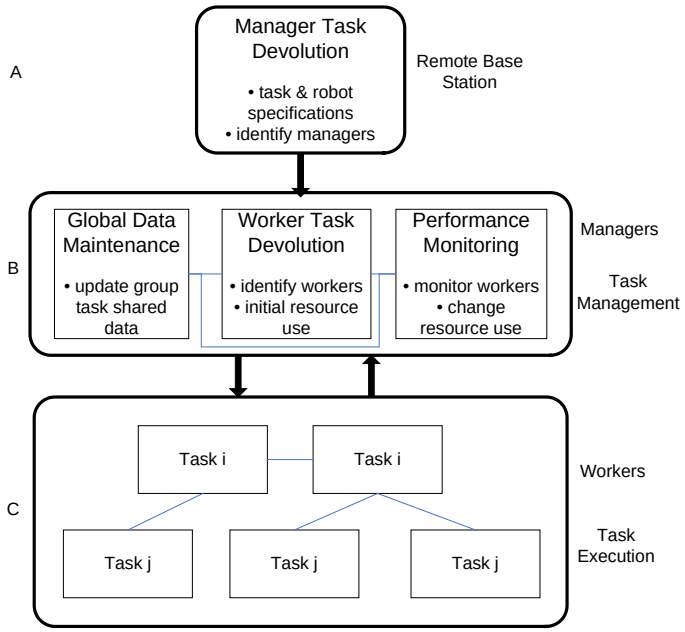


Figure 1.1: Overview of the multi-robot task allocation and coordination mechanism.

At the third level of control (level C), worker robots are responsible for executing the objectives of the group task. Worker robots are selected by the manager robot(s) during execution of a worker task devolution process. Depending on the nature of worker tasks to be allocated, the worker robots are assigned a position in a predefined hierarchy. Worker robots executing some tasks (e.g. Task *i*) could be supervising robots executing other tasks (e.g. Task *j*). Following the worker task devolution process, the worker robots perform their tasks and the manager robots monitor and direct their performance using a feedback coordination mechanism.

To evaluate the developed task allocation and feedback coordination strategies, a suitable multi-robot task needs to be implemented. A customised multi-robot map-building and exploration task is developed for this objective.

In some situations, the limited memory worker robots may be required to perform global path planning (deliberative control) to navigate beyond localised regions of the global world. This can be problematic if the limited memory robots are unable to store the entire map in their local memory. The methods reviewed in section 2.5 cannot be applied to the multi-robot application presented in this thesis. Thus another objective is to explore a new approach to global path planning for limited memory robots.

To successfully navigate obstructed environments, limited memory heterogeneous robotics requires a hybrid of deliberative and reactive control. Hence, a navigation system that combines the benefits of reactive and deliberative control for heterogeneous mobile robots is also developed in this thesis.

1.2 Thesis Outline

- Chapter 2 – A review of relevant literature is presented. Topics of interest in single robot and multiple robot control, such as control architectures, navigation systems, task allocation and coordination techniques, and fault-tolerance are reviewed. A review of multi-robot map building and exploration strategies is also included.

- Chapter 3 – The hybrid generic navigation system employed by the heterogeneous mobile robots is presented. Its design is based on the A* algorithm, a polar histogram and a modified dynamic window approach. Simulation experiments with three heterogeneous robots in a range of environments are conducted. Initial hardware experiments demonstrate the navigation system working in the real world.

- Chapter 4 – A two-tiered path planning technique to permit global path using limited (processing and memory) capability mobile robots is presented. Using a two-tiered A* algorithm that executes entirely on the limited capability robots, a set of local maps describing the global map is searched for a global path. Planning time, data communication and path length are evaluated for various combinations of local and global maps.

- Chapter 5 – Details of the proposed task allocation and feedback coordination technique for limited capability mobile robots are presented. The task allocation component employs Fuzzy Inference Systems (FISs) to simplify human user input at the task specification stage. FISs are also employed in the primary (manager) and secondary (worker) task devolution processes. Feedback coordination executes periodically to detect and correct three forms of robot failure: poor performance, partial failure and complete failure. An

exploration task (defined in chapter 6) is employed as a specific example to demonstrate the technique.

- Chapter 6 – The customised multi-robot map building and exploration task is detailed. A global environment is divided into local environments for limited capability mobile robots to explore. Planner and explorer tasks are executed by the limited capability robots to complete the exploration of a global world. Explorer robots utilise the navigation system presented in chapter 3 and planner robots plan global paths using the technique presented in chapter 4. Exploration of global worlds with relatively flat terrain containing sections of boggy terrain is considered.

- Chapter 7 – Experimental results of utilising the task allocation mechanism for the customised multi-robot map building and exploration task are presented. The influence of weight variation on ranking and selecting candidate robots is evaluated for five sets of eight robots.

- Chapter 8 – Results obtained from executing the multi-robot mapping and exploration task without any feedback is compared with task execution employing feedback coordination. Each type of feedback (task score, poor performance, partial failure and complete failure) is tuned and a complete system with tuned parameters is also evaluated.

- Chapter 9 – A summary of the contributions and publications arising from this research is presented as well as a discussion of future work.

2 Background and Related Work

2.1 Overview

The development of a hierarchical heterogeneous multi-robot system incorporates several areas of both single robot and multiple robot control. Figure 2.1 illustrates the relevant literature areas that have been reviewed. The chapter begins by discussing topics related to single robot control. These areas include single robot control architectures (section 2.2), hybrid navigation systems (section 2.3), reactive motion control and obstacle avoidance (section2.4) and memory constrained path planning (section 2.5).

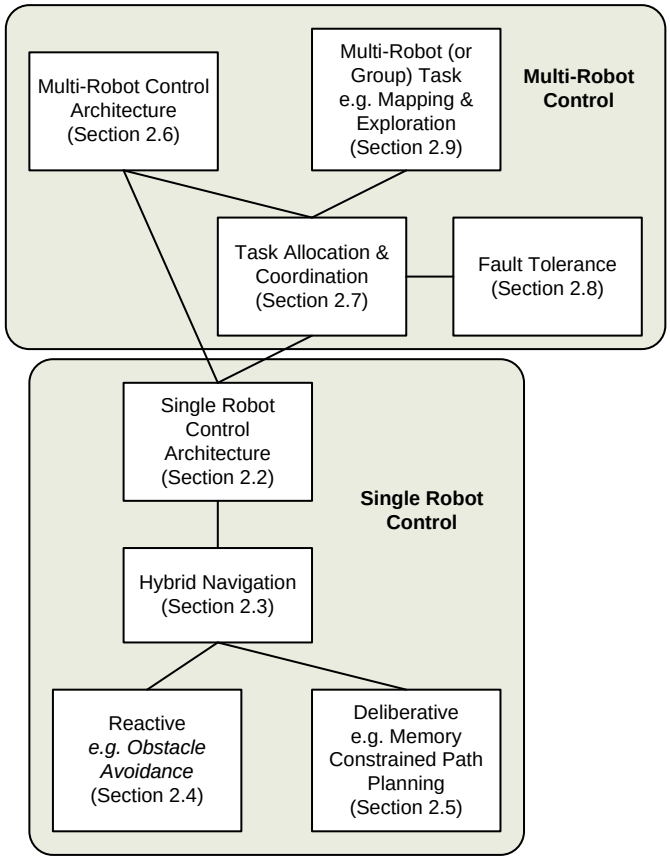


Figure 2.1: Block diagram of relevant literature areas reviewed.

Next, the chapter proceeds to topics in multi-robot control. Relevant areas reviewed include multi-robot control architectures (section 2.6), task allocation and coordination (section 2.7), and fault tolerance (section 2.8). The primary objective of a multi-robot system is to perform a multi-robot (or group) task. This thesis utilises a

multi-robot exploration and map building task as an example application. Hence, a review of multi-robot map building and exploration is included (section 2.9).

2.2 Single Robot Control Architectures

A robot's control architecture provides the framework in which a variety of control algorithms are implemented to enable appropriate functionality. There are three main categories of robot control architectures: reactive, deliberative, and hybrid systems.

Two popular reactive control architectures for mobile robots include the subsumption architecture [23] and the motor schema architecture [24]. In the subsumption architecture, reactive systems are structured from the bottom up using layered sets of rules. Inspired by the biological sciences, the motor schema architecture provides distributed and parallel primitive behaviours that are coordinated to produce an intelligent robot.

Behaviour-based systems, such as those reviewed by Arkin [25], can also be classified under reactive control architectures. However, Mataric [26] and Mataric and Michaud [27] have argued that there is a distinction between reactive and behaviour-based systems. Examples of behaviour-based systems include Control Architecture for Multirobot Planetary OUTposts (CAMPOUT) [7], Architecture for Behaviour-Based Agents (ABBA) [28], and Distributed Architecture for Mobile Navigation (DAMN) [29].

Generally, there is a tight coupling between sensing and actuation in reactive and behaviour-based systems. They feature real-time response, less dependence on complete world models and reduced predictive capabilities when compared with deliberative systems. However, tasks that require explicit world representations and high level intelligence can be difficult to implement in reactive and behaviour-based systems.

Before the development of reactive and behaviour-based architectures, deliberative reasoning methods were extensively used in robotics research [25]. Deliberative architectures [30, 31] integrate world knowledge into a robot's control by maintaining an explicit world representation. World knowledge is used to reason about the robot's

actions by optimising its performance relative to the world representation. The explicit world representation enables deliberative systems to solve certain types of problems better than reactive systems. However, systems relying solely on deliberative control are rare because they are generally too slow to cope with real world dynamic environments. World knowledge maintenance and optimal action selection computational overheads are the main causes of latency.

Hybrid architectures [32-35] have become popular in recent years as they combine the benefits of reactive control and deliberative control. According to Arkin [25] and Mataric and Michaud [27], reactive and deliberative control can be complementary. The deliberative component can guide the reactive component to avoid local minima situations. In turn, the reactive component can assist the deliberative component by making subtle changes to plans during execution. Managing the interactions between the reactive and deliberative components is a key challenge in hybrid architecture design.

The Autonomous Robot Architecture (AuRA) [32] was one of the first implemented hybrid architectures. AuRA uses motor schemas for reactive control and a traditional AI spatial planner for deliberative control. In the Planner-Reactor architecture [33], a planning mechanism is used to continuously modify a reactive controller according to some high level objectives. The three level Atlantis architecture was developed by Gat [34] and tested on the Mars rover Robby. In Atlantis, the deliberator and sequencer levels acknowledge failures and adapt the reactive control level accordingly. Layered behaviours link sensing to actuation via a local perceptual space in the Saphira architecture [35]. Saphira's Procedural Reasoning System (PRS) controls the activation and deactivation of the layered behaviours.

Based on the benefits of hybrid systems, a hierarchical hybrid approach [36, 37] is utilised to control individual robots in this thesis (chapter 3).

2.3 Hybrid Navigation Systems

Robot navigation systems are analogous to robot control architectures. A range of algorithms to facilitate successful navigation to a goal location are implemented in

navigation systems. Part of this thesis focuses on the development of a navigation system that can be employed on memory constrained heterogeneous robots. In particular, heterogeneity refers to robots with varying size, shape, drive type and sensor quantities for this thesis. A high degree of flexibility is needed for successful navigation in known and unknown environments. Hence, a hybrid navigation system that employs both reactive and deliberative control is favoured. A hierarchical hybrid navigation system can achieve the benefits of deliberative and reactive control.

The dynamic window approach [38] is a popular reactive collision avoidance technique. A weighted combination of goal-directedness, obstacle clearance and linear velocity determines the suitability of angular and linear velocity pairs in most implementations. It has been adapted, merged with other reactive techniques and combined with deliberative navigation techniques to produce a range of navigation systems.

A convergent dynamic window approach has been developed by Ogren and Leonard [39]. This technique combines the dynamic window approach with exact robot navigation using artificial potential functions. A framework suggested by Primbs *et al.* [40] combines the complementary properties of the two navigation strategies. Convergence is mathematically proved and a simulation example has been presented for circular-shaped robots only.

The nearness diagram method and dynamic window approach have been combined to produce reactive collision avoidance in dense and cluttered environments [41]. A direction for the current situation is computed from seven situations in the nearness diagram module. This direction is tracked using the dynamic window approach [42]. The framework has been implemented on a rectangular non-holonomic robot. Limited experiments indicate the robot travelled at average speeds of 0.318 m/sec to 0.48 m/sec through a cluttered environment.

A navigation strategy that integrates the dynamic window approach, elastic band and NF1 [43] path planning has been implemented for a differential drive tour guide robot [42]. In this implementation, the dynamic window method has been modified to use look-up tables for collision prediction. Collision prediction employs time to collision instead of distance to collision. The NF1 planner generates new plans that are adapted

by the elastic band as the robot moves through the environment. This implementation is hybrid deliberative-reactive and it is unclear how the modifications made to the dynamic window method function in a reactive system.

The A* path planning algorithm and dynamic window method have been integrated for indoor mobile robot navigation [44]. In this approach, the velocity space search advantages of the dynamic window method are combined with the local minimum free search characteristics of the A* algorithm. Global path planning is performed at every control cycle for navigation in unknown environments. A circular differential drive robot in a simulated environment is used to verify the method. This navigation system is extended in [45] where the A* algorithm is replaced with the focused D* algorithm [46] to achieve real-time control in dynamic environments.

A globalized version of the dynamic window approach similar to [44] is presented in [47]. Here, the dynamic window approach is used with an NF1 path planner instead of the A* algorithm. The framework permits goal-directed reactive motion in unknown environments and has been demonstrated on a circular holonomic robot.

Robot navigation comprises a model stage and planning stage in [48]. A reduced dynamic window (dynamic line) method accounts for robot shape and dynamics in the model stage. In the planning stage, an NF1 path planner generates paths with a curvature dependent velocity profile. The navigation system has been tested on two rectangular differential drive robots. A disadvantage of this method is that it is not discussed how the model stage copes if planning fails.

In [49], real-time obstacle avoidance based on the dynamic window approach is achieved via robot-specific look-up tables. A wave front expansion algorithm is used to generate intermediate way points for global navigation. The system has been tested on a circular synchrodrive robot and a forklift tricycle robot.

The obstacle-restriction method for robot obstacle avoidance in difficult environments is a two stage navigation technique [50]. A sub-goal selector stage identifies free space and selects alternative sub-goals if necessary. In the motion computation stage, motion towards the target direction is determined while avoiding collisions. The motion computation stage, like the dynamic window method, accounts for motion

constraints and steering direction motion. This method has been demonstrated on differential drive rectangular robots where reversing was not required.

A rudimentary hybrid navigation system that employs an A* path planner and the dynamic window method has been developed by Lee-Johnson [51]. This rudimentary system only supports differential drive robots with circular shapes. Hence, a major shortfall is a lack of support for robots with varying drive types and shapes. A pre-generated grid map with fixed binary occupancy data is employed for path planning. Hence, the system does not have map updating capabilities. Additionally, the path planner does not account for the size of the robot and varying occupancy probabilities. Lee-Johnson has independently further developed the rudimentary navigation system into an emotion-based control system [52].

The methods described above are not well suited for heterogeneous robots that may have limited sensing and processing capabilities. Additionally, the reactive control component is highly dependent on the deliberative component such that regular (periodic) path re-planning is required in unexplored obstructed environments. Hence, additional reactive control techniques and memory constrained path planning strategies are reviewed in section 2.4 and section 2.5, respectively.

2.4 Reactive Motion Control and Obstacle Avoidance

Reactive control for point-to-point motion and local obstacle avoidance is an important component of many autonomous mobile robots. Arras et al. [48] present a brief overview of various obstacle avoidance techniques. Some of these techniques include: potential field approaches [53-55], the elastic band concept [56], nearness diagram concept [57], vector field histogram concept [58], curvature velocity approach [59] and dynamic window approach [38].

Many obstacle avoidance approaches are restricted to a simplified robot shape (mainly circular) [48]. With the exception of [36, 37, 48, 49, 60, 61] all reviewed techniques assume a circular shaped robot. However, a circular shaped robot may not be ideal for some tasks. For instance, tricycle robots can provide good stability and manoeuvrability for object manipulation [15]. Approximating a non-circular robot

with a circle often results in highly conservative obstacle avoidance behaviour. The ego-kinodynamic space [62] is a complex framework that abstracts vehicle constraints from reactive control methods allowing robots with different shapes to use common obstacle avoidance approaches.

Many obstacle avoidance techniques can be broadly classified into directional methods [53-55, 58] and velocity space approaches [38, 59]. These methods are of interest because they deduce a motion command from the current sensor readings by applying a single rule. These techniques have been improved. Search-based look-ahead verification has been added to the vector field histogram concept [63]. The lane curvature approach [64] overcomes problems arising from the curvature velocity assumption that the robot always moves on circular arcs. A globalized version of the dynamic window approach is presented in [47]. The dynamic window approach is used with an NF1 path planner in [49], where real-time capability is achieved via a robot-specific look-up table. In [48] a reduced dynamic window (dynamic line) is used with an NF1 path planner to avoid the use of look-up tables.

Directional methods generally have limited ability to account for kinematics and dynamics of a robot. As a result, they are generally unsuitable for high speed navigation. On the other hand, velocity space approaches account for kinematics and dynamics but can be overly conservative.

The methods presented in this section have not been tested on a variety of robot shapes and drive types. Additionally, they do not explicitly combine directional methods and velocity space techniques. A hybrid of directional methods and velocity space approaches is presented in [36, 37] to mitigate their weaknesses. Chapter 3 details this hybrid approach to reactive collision avoidance.

2.5 Memory Constrained Path Planning

Microcontroller based mobile robots often have limited memory capacity and hence cannot store large volumes of environment data. In certain applications, these memory constrained robots may need to perform path planning to navigate beyond their local

region [65]. However, several reviews of path planning strategies do not outline any memory efficient implementations that enable these robots to perform such a function.

In this thesis, a rectangular occupancy grid [66] is used to represent the robot's environment. This has been favoured over more complex decompositions [67] or roadmap methods [68], [69] because of its simplicity and usability in a range of environments. Occupancy grids can be searched using algorithms such as A* [70], Dynamic A* (D*) [46], spreading activation [26] or wavefront propagation [43]. The A* algorithm has been selected for its ease of implementation.

The A* algorithm [70] is a best-first heuristic search algorithm that is often used for path planning on mobile robots that maintain node based maps [36], [43]. While A* is an efficient algorithm that utilises heuristics to guide the path planning process, its scalability is limited by the size of the map it has to search, just like any other path planning algorithm. Multi-tiered path planning strategies that do not utilise A* have previously been investigated primarily to facilitate faster plan generation rather than addressing memory limitations [71-74]. A review of multi-tiered and memory efficient implementations of A* based path planning is considered in the context of this thesis.

Warren [75] has developed a path planner using a modified A* method. The method involves using trial vectors that span several cells to perform a "loose" search in a fine grid, sacrificing path optimality in favour of speed. However, the issue of memory requirements for map storage and path planning variables is not addressed.

Zhao et al. [76] consider implementing a navigation system on mobile robots with limited on-board memory and communication bandwidth by using cache memories and auxiliary memory. The auxiliary memory and cache memories are physically (locally) available on the robot. A modified A* heuristic-search algorithm is used to plan global paths in a node-based map. For local planning, a potential field method searches a grid map for a path. The effect of various local and global cache policies on data access and planning times is investigated. Portions of the local map cached vary in size depending on robot velocity and a desired sample interval. Global cache data varies based on the environment type. Overall, this method addresses limited memory planning by installing additional memory on a robot.

Agent-centred search (also known as real-time or local search) is a technique for interleaving the planning and execution of paths. This type of search trades off planning and execution cost and may lead to memory savings. Learning real-time A* (LRTA*) [77] uses state values in a local search space to navigate towards a goal incrementally. Several other agent-centred methods similar to LRTA* are also discussed in [77].

Additional agent-centred search methods unlike LRTA* include [78] that employs rapidly-exploring random trees and [79] where a complex neural network and evolution strategy trades-off computation and memory search times. Agent-centred methods may require a robot to revisit a local search space while traversing towards the goal, especially if it has limited sensing abilities. Also, execution costs can be significantly higher than planning costs for mobile robots thus the local search space needs to be carefully selected.

Razavian and Sun [80] have developed an adaptive path planning algorithm that consists of a primary path stage and a refined path stage. Their adaptive algorithm's performance is compared to the A* algorithm. The performance metric used is the ratio of cells in the final path to the number of cells searched in determining the path. While their method reduces the number of cells searched and may save memory in path finding, the approach will not necessarily always produce a near optimal path. The method also implicitly assumes that the entire grid map is locally available on the robot.

Holte et al. [81] use a STAR abstraction technique to implement a hierarchical A* search algorithm. Path planning using their technique essentially involves searching for a path in an abstracted search space followed by using the abstract path to guide the search in the original search space. The emphasis of their work was to reduce path planning time and analysis of memory usage is not considered.

Game programmers have also developed hierarchical variants of the A* algorithm for path planning [82]. These methods employ a macro level search to produce a coarse resolution path followed by micro level search between coarse path nodes to refine the path. Schneider et al. [83] take a similar approach to address the issue of real-time

non-holonomic path planning in huge terrain datasets. These approaches are mainly used for faster plan generation and do not address memory issues involved in storing large grid maps.

The framed-quadtree approach [84] uses a grid based representation that lumps free space into single cells to save memory. This approach has been used with the D* algorithm [46] to plan paths for mobile robots in sparse environments [85]. However, as shown in [85] the memory requirements for this method of path planning increase with obstacle density and eventually exceeds that used by a regular grid representation. Similarly, the execution time increases with obstacle density to eventually exceed that of a regular grid. In addition to this, a framed-quadtree approach presents difficulties in planning paths to avoid other robot territories and obstacles simultaneously.

The memory requirements for storing variables associated with the A* search can be reduced by implementing linear space variants of A* such as iterative deepening A* (IDA*) [86] and recursive best-first search (RBFS) [87]. However, these methods have been shown to perform poorly in grid maps [88].

Alternatively, memory can be saved by not storing the closed list when searching for a path using A*. This has been demonstrated in [88] where a divide-and-conquer technique is employed to plan paths while avoiding duplicate search. Optimal paths can be produced using this method. However, this method is unable to produce partial paths that can be useful in mobile robot navigation. Zhou and Hansen [89] implemented a similar sparse-memory A* algorithm that only stores a small part of the closed list to save memory. These techniques are not well suited for a limited memory implementation where the entire map cannot be stored locally. This is due to essential recursive calls to the algorithm that may introduce significant communication overheads.

Since the methods reviewed in this section are not directly suitable for the heterogeneous limited memory robots and multi-robot application presented in this thesis, an alternative strategy is proposed (chapter 4). This alternative technique is a two-tiered global path planning method based on the A* algorithm.

2.6 Multiple Robot Control Architectures

The multi-robot or group architecture provides the infrastructure upon which collective behaviours are implemented and determines the capabilities and limitations of a multi-robot system [90]. One of the key features of a group architecture for mobile robots is whether the system is centralised or decentralised. Centralised architectures are characterised by a single control agent whereas decentralised architectures allow multiple control agents. In pure centralised architectures, all slave robots are completely dependent on a central master robot for commands and control. If communication is lost at any point in time then the slave robots fail to function. The decentralised architecture has been the dominant group architecture since it has several inherent advantages over centralised structures. Two types of decentralised architectures include hierarchical architectures and distributed architectures [90].

Hierarchical architectures are locally centralised and can consist of multiple master control agents. Subordinate robots are independent in carrying out tasks to achieve certain goals but they communicate with a master or host that has a global view of operations and assigns goals to the agents. The subordinate robots can remain functional with intermittent loss of communication. CEBOT, a hierarchical architecture, consists of a group robotic system that is dynamically reconfigurable, has been simulated [91]. The GOFER architecture, which uses a central task planning and scheduling system, was used to study distributed problem solving by multiple robots in an indoor environment using traditional AI techniques and was successfully used with three physical robots [90]. Cooperative behaviour based on fuzzy logic optimized by micro genetic algorithms for fixed obstacle and multiple robot avoidance in a centrally managed robot system has been simulated in [92].

Hierarchical architectures have also been implemented in cooperative robot soccer teams where autonomous soccer robots are linked to a host computer system [2]. The use of two cooperative robots operating in a master/slave configuration to facilitate localization and mapping has also been studied [93].

Distributed architecture implementations remove the need for a master or host. All robots have equal control and are largely autonomous in their decisional process,

relying only on critical information from other robots. Robots in distributed systems can function autonomously when communication is partially or completely lost. A practical application based on a distributed architecture is map building for exploration in an unknown environment using two cooperative robots [8]. In this application the robots share perceptual information, but maintain separate maps and make independent decisions which leads to the system being robust to the loss of communication between them as well as to the loss of a robot. A distributed system carrying out a box pushing task using explicit communication for coordination has been shown to perform more effectively than a single robot or two non-communicating robots [4].

A cooperative box pushing mission by two heterogeneous robots has been achieved using a fully distributed system at both the individual robot and team levels based on the ALLIANCE architecture [12]. This architecture has also been implemented on a physical robot team performing a laboratory version of a hazardous waste cleanup. The ALLIANCE architecture has the advantage of using adaptive actions to achieve fault tolerant control within small to medium sized teams of heterogeneous robots. The ABBA architecture [28], which is designed for distributed cooperative planning, has been utilized in the implementation of a cooperative cleaning task with two autonomous mobile robots. This implementation has also shown the advantage of robustness in the face of failures. CAMPOUT [7], a distributed control architecture for tightly coupled coordination of multiple robot systems has been developed at the Jet Propulsion Laboratory. It has been applied to ongoing physical experiments involving the exploration of cliff faces and the deployment of extended payloads.

A fully distributed architecture is cost expensive to implement if there are many robots in the system. All robots will need to have sufficient processing abilities to plan and execute tasks independently. However, in this thesis, many of the robots will have limited processing, sensing, and actuation capabilities. Hence, a decentralised hierarchical architecture is proposed for the multi-robot system (section 1.1). Manager robots act as master control agents. Subordinate worker robots are independent in carrying out tasks to achieve certain goals, but they communicate with the mangers to receive their goals and tasks.

2.7 Task Allocation and Coordination

Central to the success of many multi-robot systems is the ability of the individual robots to cooperate and coordinate their activities. This can result in advantages such as increased efficiency in performing tasks and robustness to failure of individual robots. Coordinating a team of mobile robots usually involves implementing task allocation and coordination mechanisms. Various methods for coordination and task allocation in multi-robot systems have been discussed in [19, 20, 94]. Whereas [20] focuses on coordination, [19, 94] address task allocation.

Of the classifications based on coordination identified in [20], the weakly centralised systems [95-97] are of particular interest since they can be utilised in hierarchical heterogeneous systems. In these systems, a leader robot is selected dynamically during task execution based on the situation of the team and the environment.

A weakly centralised approach is proposed in [95] where the robots start auctions and bid to become the team leader. The proposed method intends to take into account the physical capabilities of robots but has not been fully implemented. A fixed set of robots and fixed task assignments are employed in experiments. In [96] the robots are heterogeneous and a leader is selected based on specific sensing or actuation capabilities. A pair of robots (a supervisor and pusher) is utilised to accomplish a box pushing task. Experiments with larger teams of robots are neither simulated nor physically implemented. Market based strategies (described below) are employed for dynamic selection in [97]. Pockets of centralisation improve task allocation optimality when compared to purely distributed methods. Revenue and cost functions need to be defined but these are not expressed in terms of physical robot capabilities. A task domain specific clustering algorithm is required to process bids and select a leader robot in a timely manner.

In [94] a taxonomy has been developed for the multi-robot task allocation problem, differentiating robots as either single-task (ST) or multi-task (MT), tasks as either single-robot (SR) or multi-robot (MR), and assignment types as either instantaneous (IA) or time-extended (TA). Representative approaches to multi-robot task allocation [9, 12, 98-102] are analysed in [94] based on the developed taxonomy. In these approaches, a set of indivisible tasks is distributed amongst a team of robots such that

each robot executes an individual task. It has been shown that developing an optimal mapping of tasks to robots is NP-hard [22]. Hence, existing approaches employ heuristic greedy methods to achieve this mapping, leading to suboptimal solutions.

ALLIANCE [12] and BLE [102] are examples of behaviour-based approaches to multi-robot task allocation. Behaviour sets typically represent tasks in these approaches and action selection mechanisms are utilised to enable or disable behaviours representing tasks. ALLIANCE uses motivational behaviours to monitor and dynamically reallocate tasks thus achieving fault tolerance and adaptive behaviour. In the BLE system, each robot has a corresponding behaviour that is capable of executing each task. The robots select a task to execute by continuously broadcasting locally computed eligibilities followed by determining the most eligible task using a greedy algorithm. A behaviour-based approach to multi-robot task allocation that uses the concept of "vacancy chains" [103] is presented in [104]. Vacancy chains are social structures capable of resource redistribution. In [104], the resources to be redistributed are the tasks requiring allocation or reallocation. The approach is demonstrated in groups of homogeneous robots where "vacancy chains" emerge through reinforcement learning.

Market-based task allocation methods [9, 97, 98, 100, 101, 105] have also been widely utilised in multi-robot systems. In these approaches, the global task consists of a set of tasks that the robots bid and negotiate on. Costs and revenues are associated to the tasks, and robots can trade the tasks trying to maximise their revenue. An auctioning mechanism utilises a task to revenue/cost mapping function to greedily assign tasks to the highest bidders. TraderBots [106] is a market-based approach that has the ability to allocate resources and roles in addition to tasks. This is achieved via a RoboTrader module that executes on each robot. Complex tasks comprising sets of atomic tasks with some constraints placed on them can be represented as task trees in market-based techniques [107]. Robots can bid on combinations of task tree nodes resulting in a complex task being assigned to one or more robots. A drawback of market-based methods is that it can be difficult to express revenue and cost functions in terms of robot physical capabilities such as processing, communication, sensing and actuation. Additionally, the reviewed market based methods assume that a robot already meets the physical capability requirements of a task when it makes a bid.

Dynamic role assignment [99] assigns roles to each robot in the team. The behaviour of robots is modelled using a continuous state space representation. A hybrid automaton [18] models robot behaviour together with the roles, role assignments and associated variables. Each role is a control mode of the hybrid automaton. During the execution of a cooperative task, robots within a team can dynamically exchange roles in a synchronised manner adapting to changes in the environment. Specialised dynamic role assignment methods have been used for robotic soccer [3, 108] where the robots dynamically switch between roles such as attacker and defender or master and supporter. These approaches do not address the issue of robots autonomously determining contributions to the solution based upon processing, communication, sensing and actuation capabilities. Additionally, the robot team is known *a priori* in dynamic role assignment.

Burgard et al. [11] address task allocation and coordination in multi-robot exploration. For each robot, they trade-off the utility and cost of potential target points for exploration. In this manner, each robot is assigned a target point for exploring. A more recent example using a similar technique for coordination is [109]. The coordination strategies of [11, 109] are not explicitly based on the computational and physical resources that each robot possesses and assumes that each robot is capable of solving the exploration problem. Additionally, robot malfunctions are not addressed in these implementations.

Teamwork models have been developed that provide mechanisms for agents to form teams to accomplish a common goal [110, 111]. A general model of teamwork, STEAM (Shell for TEAMwork) is presented in [111]. It facilitates monitoring of team performance and allows team reorganisation. STEAM is based on a hybrid of joint intentions theory [112] and SharedPlans theory [113]. Joint intentions theory relies on high level conditions that are based on robots having mutual goals (joint persistent goals). SharedPlans is not based on a joint team mental attitude, unlike joint intentions. Instead, individual agents intend that a collaborator is able to produce a solution for a required action. STEAM has been applied to the simulated tasks of helicopter attack and transport formation and RoboCup synthetic soccer.

Another general model based on joint intentions theory is the joint responsibility GRATE (Generic Rules and Agent model Testbed Environment) system [110]. This system involves satisfying defined preconditions before collaboration can begin in addition to generating plans for agent behaviour during collaboration. Hence, it is able to provide an explicit model of cooperative problem solving for industrial applications. GRATE agents have two components; a cooperation layer and a domain level system. The GRATE system has been applied to the real-world domain of electricity transportation management.

STEAM and GRATE provide high-level models for problem solving and role assignment. However, the contributions and requirements of physical robot capabilities (such as processing, communication, sensing and actuation) are not addressed in these approaches.

Parker and Tang [21] consider the problem of single-task robots performing multi-robot tasks in the development of heterogeneous robot coalitions that solve single multi-robot tasks. They use an approach called ASyMTRe (Automated Synthesis of Multirobot Task solutions through software Reconfiguration) to generate multi-robot coalitions using complete information. The approach employs perceptual and motor control schemas to encode robot capabilities. Connections between schemas across multiple robots are determined and evaluated to find a task solution. Hence, ASyMTRe is primarily suited to reactive or behaviour-based systems. The approach is demonstrated on multi-robot transportation tasks that require robots to share sensor and effector capabilities. Due to its dependence on schemas, ASyMTRe may be difficult to apply to systems that do not employ schemas or are not behaviour-based.

Vig and Adams [13] identify issues that arise while attempting to use multi-agent coalition formation algorithms for multi-robot systems. Their work also addresses the problem of single-task robots performing multi-robot tasks. They develop a multi-robot coalition formation algorithm using an adaptation of Shehory and Kraus' [114] algorithm for multi-agent coalitions. Shehory and Kraus' algorithm is a distributed task allocation strategy that assigns a task to a group of agents. It consists of greedy distributed set-partitioning and set-covering algorithms. Their algorithm considers cases where tasks have a precedence order and is applied to a Blocks World domain

problem where blocks are moved from an initial configuration to a final configuration using multiple agents. A drawback of Shehory and Kraus' algorithm is that no auxiliary mechanisms are employed to monitor the potentially suboptimal coalitions during task execution. Subsequently, unexpected failures during task execution are also not accounted for.

By addressing the issues of computation versus communication, task format, and coalition imbalance, Vig and Adams develop their multi-robot coalition algorithm. The first stage of the algorithm involves distributively calculating initial coalition values for all possible coalitions while a second stage involves robots agreeing on coalitions and forming them. Iteratively, the algorithm is able to form multiple coalitions, hence assigning multiple robots to multiple tasks. Coalition sizes of two to five robots have been demonstrated for box pushing, cleanup and sentry duty tasks. Vig and Adams multi-robot coalition formation method also suffers from the drawbacks of Shehory and Kraus' algorithm.

Vig and Adams [115] also developed RACHNA (Robot Allocation through Coalitions using Heterogeneous Non-Cooperative Agents), a market-based multi-robot task allocation scheme. A multi-unit combinatorial auction is employed to allocate resources (robots) to the best (optimal) set of tasks through overall utility maximisation. This reverse auction strategy is achievable due to the inherent redundancy in sensor and actuator capabilities of robots. Service agents and task agents are required to facilitate the bidding process of the auction. Tasks are classified in three categories (urgent, standard and non-preemtable) to achieve successful coalition formation. Preliminary simulations show that RACHNA is able to outperform global greedy (best task first) algorithms in terms of overall utility. A disadvantage of RACHNA is that the formed teams are highly dependent on the initial utilities assigned to tasks. Task utility usually incorporates a balance between quality (revenue) and cost. It can be difficult to quantify the quality of task execution prior to coalition formation. Additionally, no performance monitoring system is employed during task execution.

An area of task allocation and coordination that has not been addressed in the literature is the ability to specify tasks in a generic format such that non-expert human users can adapt and utilise multi-robot systems. This requires a reduction of human

user inputs to the system. The requirements of physical robot capabilities (such as processing, communication, sensing and actuation) can be specified as graded inputs by non-expert human users. Specifying task requirements explicitly in terms of physical robot capabilities and utilising this information to determine the contributions of robots towards a global task has not been addressed in the reported approaches. Chapter 5 presents a task allocation strategy that utilises reduced human user inputs and has the potential to be utilised by non-expert human users.

A potential problem faced after task allocation is the failure of robots during task execution. This may be due to hardware failures, interactions with the environment, or incorrect task-robot matching. Some form of feedback is required to address this issue.

2.8 Fault Tolerance

Fault tolerance in multi-robot systems can be viewed as a specialised form of multi-robot coordination. In fault tolerant multi-robot systems performance metrics and monitors are used to detect robot failures. The team responds to individual robot failures by dynamically reselecting or reassigning the tasks of the failed robot.

Parker [22] has implemented performance monitors for behaviour sets in the L-ALLIANCE architecture. Task completion time is used as the performance metric in this architecture. Hence, the performance monitor on each robot keeps track of task completion times of any robots performing similar tasks to it. Task reallocation is effected via a learning process that updates the control parameters of the behaviour sets associated with the task that a robot executes. A drawback of L-ALLIANCE is that it is tailored for behaviour based systems making it potentially difficult to apply to non-behaviour based architectures.

Kannan and Parker [116] have developed task execution success and failure metrics to investigate the influence of fault tolerance on overall system performance. In their implementation, the robots are required to perform a number of tasks and each robot-task pair contributes towards the overall performance. The overall performance is the difference in the reward gained from successfully executed tasks (success metric) and

the punishment for unsuccessful task execution (failure metric). Experimental data are used to quantify the performance of a large-scale locate-and-protect mission involving a large team of heterogeneous robots. A drawback of this approach is that performance is only determined after task execution completes and not during task execution.

An extension to Kannan and Parker's work on fault tolerance [117] measures the effectiveness of fault tolerance in box pushing and deployment tasks. Fault tolerance in these tasks is tested using predefined and adaptive causal model methods. A case-based reasoning approach is utilised in the implementation of the adaptive causal model. Their results indicate that adaptive models provide more effective fault tolerance than predefined models. However, the implementation of causal model methods can be cumbersome when there are many robots, fault nodes, and fault combinations. Additionally, causal model methods need to be tailored for the task that the robots execute and the environment that they operate in.

Tolerance to sensor failures in a small team of distributed robots has been investigated in [118]. This research extends the sensing fault tolerance capability of the Sensor Fusion Effects (SFX-EH) architecture [119] to multiple robots. Sensor failures are diagnosed by allowing the robots to share knowledge of state of their sensors and task execution via communication. Tasks are redistributed when robots become inoperable. The distributed nature of the multi-robot system allows robots with failed sensors to attempt recovery by accessing sensory information from other robots in the team. A target search scenario using two robots demonstrates the usefulness of the fault tolerance mechanism. The main drawback of this implementation is that it only addresses sensing failures. Also, the implementation uses cameras and it would be difficult to apply this technique to limited capability robots that do not employ vision sensors.

None of the reviewed fault tolerance methods monitor the four broad categories of robot hardware resources (processing, communication, sensing and actuation) explicitly. Such an approach is required if tasks are specified to robots in terms of these resource categories. It is envisioned that this will enable the detection and correction of various types of hardware failures and failures due to poor interaction with the environment (possibly due to incorrect task-robot matching). A feedback

coordination mechanism that achieves fault tolerance by monitoring the four broad categories of robot resources is presented in chapter 5.

2.9 Multi-Robot Map Building and Exploration

In a multi-robot map building or exploration task, a team of mobile robots is utilised to construct a map of an unknown environment in a minimum period of time. Several techniques for multi-robot map building and exploration have been proposed. Generally, these techniques can be classified into either hierarchical [11, 120, 121] or distributed [8-10, 122, 123] approaches similar to the multi-robot control architecture classification of section 2.6.

Some research in multi-robot exploration and mapping (e.g. simultaneous localisation and mapping (SLAM)) [67] has focussed on issues relating to sensor data and robot position uncertainty. SLAM methods employ alternating localisation and mapping steps. Sensor data is used to improve the position of a robot based on the current map in the localisation stage. In the mapping stage, the improved robot position and sensor data updates the map data. The SLAM problem can be “solved” by spending money on sophisticated sensing and processing based robots. This is the general approach taken in many implementations [67, 124-126]. However, practical mapping with inexpensive robots must handle limitations in sensing, computing and memory. In such a scenario, navigation aids such as beacons or GPS (DGPS) can be employed to correct imprecise localisation.

This thesis considers the issue of implementing efficient exploration strategies for multiple robots. The central question in implementing an efficient exploration strategy is: “Given the current location of a robot, what is the best place or area to move to in an environment that consists of unexplored space?” Many exploration strategies are based on the distance to traverse to unexplored space and the expected information gain from the unexplored space [8] [120] [11] [9] [10] [127] [121]. Other strategies take advantage of robot capabilities to select robots for special situations [122] [123] .

Singh and Fujimura [122] address cooperation in heterogeneous multi-robot exploration. Heterogeneity arises due to varying robot sizes. During exploration, the

robots identify “tunnels” to unexplored regions. If a robot is too large to pass through a tunnel, a delegation process informs other robots and the tunnel is assigned to a smaller robot. The robots use a north-south horizontal sweeping heuristic to explore the environment. Robots utilising this approach do not take into account the allocations of other robots. Hence, there can be interference between robots or overlapping of areas explored by the robots.

Yamauchi [8] presents a decentralised strategy for multi-robot exploration where robots share perceptual information but maintain separate grid maps. Frontier based exploration directs the robots independently to areas that are likely to provide new information. Experimental results are presented for a pair of homogeneous robots. The minimal coordination arising from independent navigation can cause robots to waste time exploring areas previously covered by other robots. Also, the performance of the team is not quantitatively analysed.

A centralised approach to merge maps and explicitly coordinate robots has been implemented by Simmons [120]. A relatively low cost algorithm attempts to produce good results by maximising overall utility such that overlapping information is minimised. Local map data are forwarded by each robot to the central mapper unit to create a consistent global map. Experiments were carried out using a team of three robots. In environments with minimal obstacles, this approach can produce significant interference between multiple robots.

Burgard et al. [11] carried out extensive experiments to validate Simmons’ approach for teams of up to twenty robots in three types of environments. The results indicate that the coordinated approach significantly outperforms an uncoordinated system. Burgard et al. also extends the technique to robot teams with limited communication range. Limited communication experimental results indicate that a communication range of 30% of the diameter of the environment produces similar performance to exploration with unlimited communication. The authors remark that their research on multi-robot exploration and mapping can be extended to investigate scenarios in which robots may malfunction or break.

Zlot et al. [9] applied a market based approach to multi-robot exploration. In this approach each robot generates a set of goal points that are organised into a tour.

Exploration is achieved by subsequently refining the tours through continuous inter-robot negotiation. Experimental results suggest that selecting goal points using a greedy strategy yields poor performance compared to random and quadtree methods. The authors remark that incorporating time-based cost information could improve exploration efficiency and that their approach does not handle robot losses explicitly.

Stroupe et al. [10] present the Move Value Estimation for Robot Teams (MVERT) approach for directing the movement of robots mapping objects in their environment. In this approach, each robot maximises a value function with the objective of reducing uncertainty in target measurements. Hence, multiple robots are initially drawn towards uncertain targets when they are found. As a target's uncertainty reduces, the robots disperse to specialise in observing other targets. Overlapping exploration by robots is a necessity in this approach to improve observation accuracy. Experimental results indicate that MVERT can initially reduce uncertainty much faster than a non-cooperative approach. However, the eventual improvement in value is between three and thirteen percent.

A new cost function for multi-robot exploration has been proposed [127]. The new cost function includes a value for robot separation expressed in terms of the distance between assigned frontier cells. This facilitates maximum robot separation in a reduced period of time during exploration. For simplicity, a greedy algorithm is employed to assign frontier locations. Experiments are carried out to compare the performance of the new cost function with Yamauchi [128] and Burgard et al. [129]. Yamauchi [128] employs a similar cost function to Yamauchi [8] and Burgard et al. [129] uses a cost function similar to Simmons [120]. Results from exploration in a tunnel-type environment with no obstacles suggest that the new cost function yields lower exploration times.

Diosdado [123] developed the BEhavioural ROle DEcentralised (BERODE) architecture to address limited communication in multi-robot exploration and mapping. A minimum spanning tree (MST) control network maintains connectivity between the team of robots. During exploration the MST control network is updated to improve the signal quality of the network. The robots use behavioural roles to balance the tasks of exploration and network maintenance to improve the efficiency of

the team. It is assumed that robots are selected *a priori* for the exploration task. Diosdado remarks that robots can only explore a limited size environment using this approach due to computational costs associated with map updates. The performance could be improved using local maps.

An exploration strategy that balances two priorities has been proposed [121]. One priority is to explore unknown regions of an environment, while the other is to maintain a small amount of overlap with explored areas to improve localisation. A complex utility function is optimised by a randomised sampling scheme to achieve this balance. Simulations and experiments with real robots are presented. The authors argue that their strategy reduces robot proximity to objects thus reducing sensing blindness or short-sightedness.

All of the map building and exploration approaches reviewed assume that the robots have sufficient memory to locally store the entire map of the global area to be explored. However, this may not always be the case, especially if a robot uses a microcontroller as its main processor. Hence, an alternative method for map building and exploration is required (chapter 6). Additionally, using a feedback coordination strategy (chapter 5) scenarios where robots malfunction or break-down are also investigated.

2.10 Summary

Several areas of single robot and multiple robot control have been reviewed. From the relevant literature reviewed in single robot control, hybrid control architectures have been identified as the best option since they combine the benefits of reactive and deliberative control. Consequently, a hybrid system is the best option for navigation. Previously implemented hybrid navigation systems are not well suited for heterogeneous robots that may have limited sensing and processing capabilities.

A method that does not need periodic planning like the approaches reviewed is required for navigation in unexplored obstructed environments utilising limited memory robots. Existing approaches to reactive control have not attempted to explicitly combine the benefits of directional methods and velocity space techniques

such that dependence on periodic planning can be removed. Additionally, the memory constrained path planning methods reviewed for deliberative control are not suitable for the limited memory robots in the hierarchical multi-robot system outlined in chapter 1. Chapter 3 and chapter 4 present methods for hybrid navigation and memory constrained path planning that can be applied to a hierarchical heterogeneous multi-robot system comprising limited capability robots.

In the area of multi-robot control, justification for the selection of a decentralised hierarchical architecture for the heterogeneous robots has been presented. While a lot of research has been carried out in the area of task allocation and coordination, there has been no work attempting to reduce human user input for task specification such that non-expert users can adapt and utilise multi-robot systems. Specifying tasks in terms of physical robot capabilities (such as processing, communication, sensing and actuation) may enable non-expert users to specify tasks. There is always the possibility that tasks specified by non-expert users can be erroneous. Hence, some form of feedback to achieve fault tolerance is needed. Existing methods for fault tolerance cannot be utilised as they do not attempt to monitor the four broad physical capability categories of robots. Chapter 5 presents a task allocation and feedback coordination strategy that permits reduced human user input for task specification and detects abnormalities during task execution.

Existing approaches to multi-robot mapping and exploration assume that robots have sufficient memory to locally store the entire map of the global area. However, this may not always be possible in a hierarchical heterogeneous system that consists of limited memory robots. A customised multi-robot mapping and exploration task that exploits the benefits of hierarchical heterogeneous systems to permit exploration with limited capability robots is presented in chapter 6.

3 Basic Robot Navigation System

3.1 Navigation System Overview

A hierarchical hybrid system has been selected to implement basic robot navigation. This allows the navigation system to exhibit the benefits of both reactive and deliberative control. A rudimentary version of Lee-Johnson's navigation system [51] has been adapted for use. There were several bugs and shortfalls in this initial navigation system (highlighted in section 2.3) that have been addressed independently of Lee-Johnson [51].

The generic control architecture used by the mobile robots in this project is illustrated in Figure 3.1. It consists of a number of modules for sensing (information extraction and sensor fusion), modelling (localisation, environment map), planning (multi-robot coordination interface, task planning and decomposition, deliberative control) and acting (reactive control, low-level motion control/execution). The hierarchy of the modules provides an indication of the temporal decomposition of control. Modules on the left and right represent perception/representation and action/planning respectively. The indicated update rates are employed for the tricycle robots Itchy and Scratchy. These frequencies can be adjusted for other robots.

3.2 Environment Representation and Path Planning

3.2.1 Environment Representation

As outlined in section 2.5, a rectangular occupancy grid has been selected for representing a robot's environment because of its simplicity and usability in a range of environments. An occupancy grid map is generated by dividing the environment into discrete cells and assigning unit interval values to represent occupancy probability. Figure 3.2 illustrates a 30 cm resolution occupancy grid map for a 38.4 m by 38.4 m environment. Darker shaded cells represent higher occupancy probability.

While navigating towards a goal location, a robot can update occupancy probability data using Bayes' rule [8, 130]. To ensure that Bayes' rule updates probabilities correctly, occupancy probability data are restricted to a range of [0.05,0.95]. Updating occupancy probability data is further detailed in section 6.2.2.

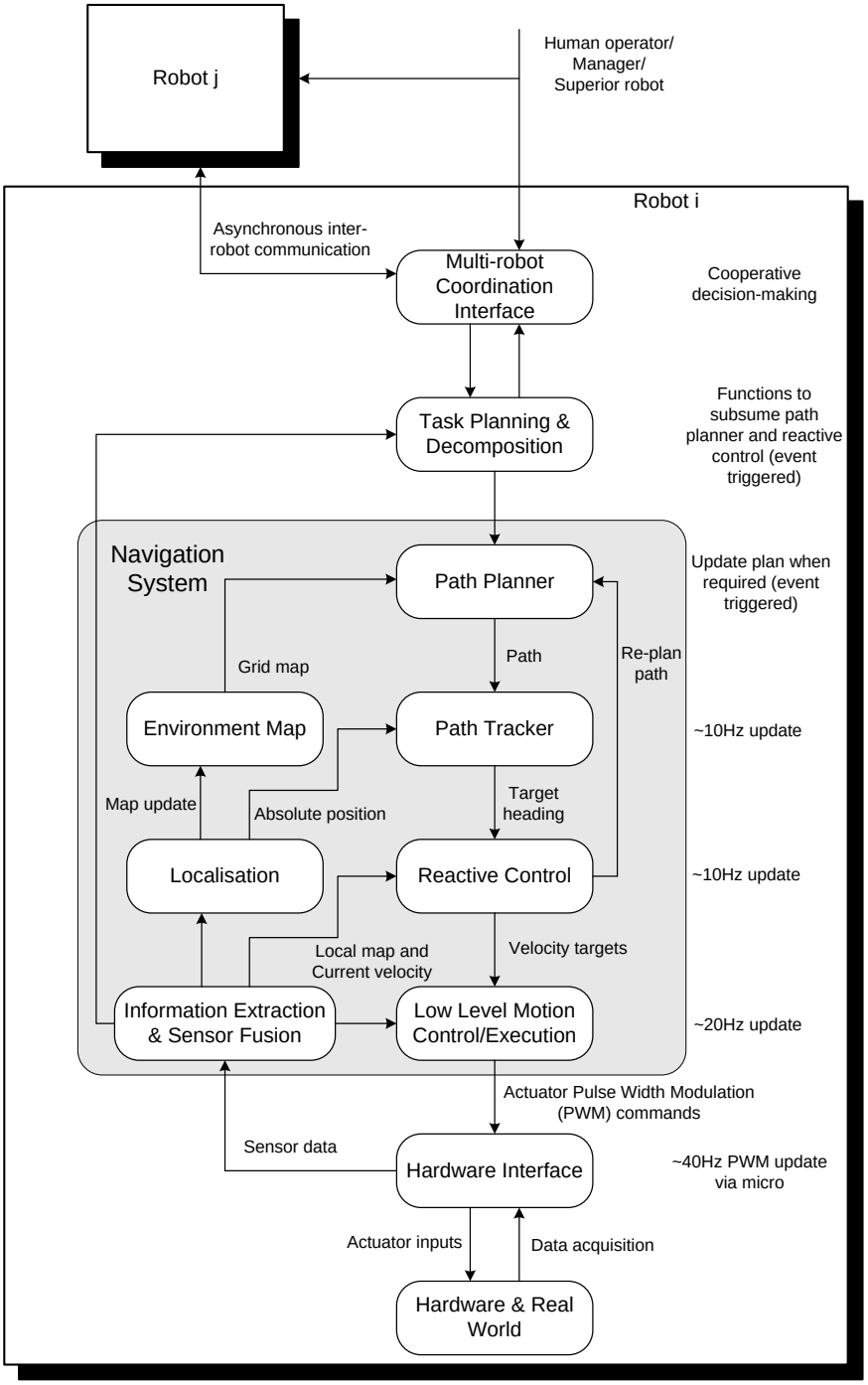


Figure 3.1: Generic mobile robot control architecture.

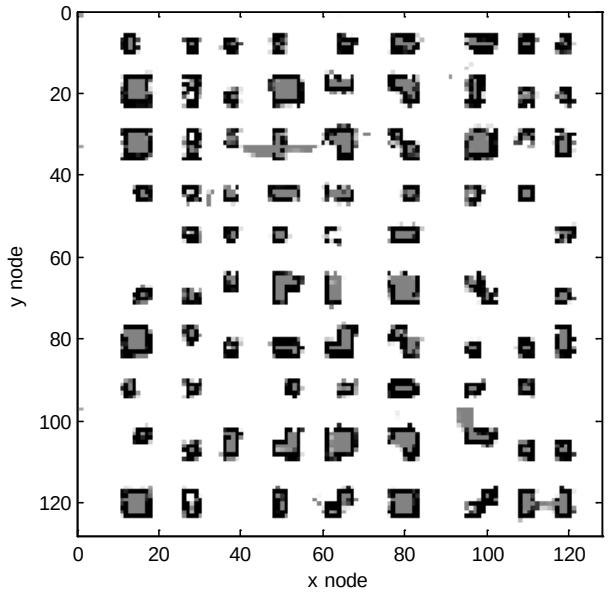


Figure 3.2: Rectangular occupancy grid map.

3.2.2 Path Planning

A single-tiered modified A* algorithm is employed for path planning if the occupancy grid to be searched can be stored in the robot's local memory. Where the entire occupancy grid map cannot be stored locally, a two-tiered (memory constrained) A* algorithm searches for a path (chapter 4).

The A* algorithm is a best-first heuristic search algorithm that ranks nodes based on the cost of travelling through them. Cost is usually represented by node distances where lower cost values denote a better path to travel. The total cost $f(x)$ of node x is the sum of two cost values, $g(x)$ and $h(x)$. $g(x)$ represents the cost of travelling from the start node to node x while $h(x)$ is a heuristic cost of travelling from x to the goal node.

$$f(x) = g(x) + h(x) \tag{3.1}$$

Normally, the A* algorithm [70] considers binary occupancy values where the nodes are either traversable or non-traversable. Hence, $g(x)$ is dependent on the node distance of the lowest cost path from the start node to the parent node x_{par} and the Euclidean distance $d_n(x,x_{par})$ between x and x_{par}. Heuristic cost $h(x)$ is generally an optimistic estimate represented by the Euclidean or Manhattan distance from the current node x to the goal node. Infinity cost is represented by the *Inf* variable in the MATLAB code. Alternatively, it can be represented by a large number such as 10^{12}.

$$g(x) = \begin{cases} g(x_{par}) + d_n(x, x_{par}) & \text{if traversable} \\ \infty & \text{otherwise} \end{cases} \tag{3.2}$$

Inspired by frontier-based exploration [8], the $g(x)$ and $h(x)$ costs have been modified to account for a varying degree of occupancy probability p_i ranging from 0.05 to 0.95. Nodes whose occupancy probabilities exceed a threshold P_T are eliminated from the cost calculation. The cost of all other nodes is linearly dependent on a cost multiplier c_m.

$$g(x) = \begin{cases} g(x_{par}) + d_n(x, x_{par}).c_m(x) & \text{if } p_i(x) < P_T \\ \infty & \text{otherwise} \end{cases} \tag{3.3}$$

$$h(x) = \begin{cases} d_n(x, x_{goal}).c_m(x) & \text{if } p_i(x) < P_T \\ \infty & \text{otherwise} \end{cases} \tag{3.4}$$

Before calculating c_m, occupancy probability $p_i(x)$ is modified by cost function c_p which can favour nodes within critical probabilities $p_{c\min}$ and $p_{c\max}$. This can be useful when exploring unknown environments.

$$c_p(x) = \begin{cases} p_i(x) & \text{if } p_{c\max} < p_i(x) < p_{c\min} \\ (1 - w_{p1})\, p_i(x) & \text{if } p_{c\min} \le p_i(x) \le p_{c\max} \ \&\ p_i(x) \neq 0.5 \\ (1 - w_{p2})\, p_i(x) & \text{if } p_i(x) = 0.5 \end{cases} \tag{3.5}$$

For path planning in known environments, weights w_{p1} and w_{p2} can be set to zero. In unknown environments the weights are unit interval values varied dynamically based on the progress distance d_{prog} towards the goal. The weights are multiplied by factor f_w when progress is below threshold $d_{prog\min}$, otherwise default initial values are used. In this way the balance between navigating through known and unknown terrain can be controlled.

$$w_{new} = \begin{cases} w_{old} . f_w & \text{if } d_{prog} < d_{prog\min} \\ w_{init} & \text{otherwise} \end{cases} \tag{3.6}$$

The unit interval cost multiplier c_m takes into account the occupancy probability cost $c_p(x)$ of node x. It also includes a mean occupancy probability cost $c_{prc}(x)$ representing a robot clearance of r nodes around node x. Weights w_{pi} and w_{prc} control the balance between the two inputs and are set to 0.5 for equal preference.

$$c_m(x) = 1 + w_{pi} c_p(x) + w_{prc} c_{prc}(x) \tag{3.7}$$

$$c_{prc}(x) = \frac{1}{r} \sum_{i}^{r} c_p(x_i) \tag{3.8}$$

If the path planner cannot find a path to the goal, it is able to retain a partial path closest to the goal. After the last node in a partial path the mobile robot can employ reactive control techniques in an attempt to reach the goal.

3.3 Reactive Control Overview

An outline of various approaches for reactive control of mobile robots has been presented in section 2.4. A reactive control strategy has been implemented that combines a modified dynamic window approach [38] with a polar histogram technique similar to the vector field histogram method [58]. Figure 3.3 shows a simplified block diagram of the two-stage optimisation process that can track paths and avoid obstacles. All heading (orientation) values are normalised to lie within the interval $[-\pi,\pi)$. A target heading angle is input to the direction sensor that produces a modified target heading as output. The modified target heading angle is decomposed into linear and angular wheel velocities (v,ω) by the modified dynamic window approach.

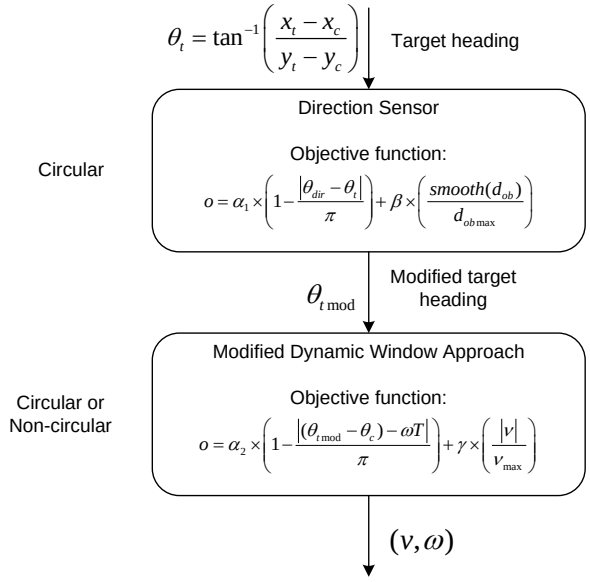


Figure 3.3: Overview of reactive control strategy.

3.4 Direction Sensor

At the direction sensor stage, the kinematic constraints, dynamic constraints and non-circular shapes are all ignored. A circular shape represents the robot as shown in Figure 3.4. Robot radius r_m is selected to encompass the entire robot. The current pose and goal pose of the robot are defined as (x_c,y_c,θ_c) and (x_g,y_g,θ_g) respectively. If the robot is following a planned path, (x_g,y_g,θ_g) represents the coordinates of a node that is N_t nodes ahead of the node closest to the robot. Otherwise, (x_g,y_g,θ_g) points to the final destination of the robot. Target heading angle θ_t is calculated from the current and goal pose of the robot and input to the direction sensor.

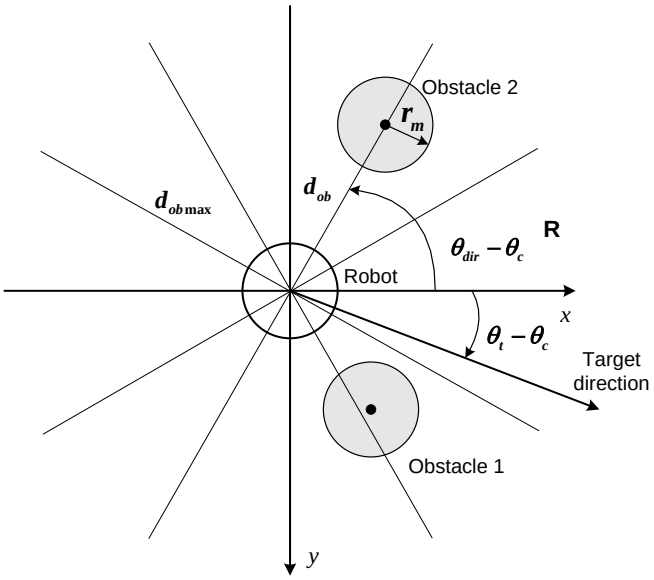


Figure 3.4: Direction sensor representation.

$$\theta_t = \tan^{-1}\left(\frac{x_g - x_c}{y_g - y_c}\right) \tag{3.9}$$

To determine the most appropriate direction, the robot is represented as a point and each obstacle is enlarged by the radius of the robot. The region surrounding the robot is then divided by an arbitrary number of lines N_θ (Table 3.2) to represent candidate orientations θ_{dir}. All orientation angles are converted to the robot's reference frame R by subtracting the current absolute orientation θ_c of the robot.

Goal directedness $|\theta_{dir} - \theta_t|$ and distance to obstacles d_{ob} is maximised by applying an objective function to each candidate orientation (3.10). A higher objective function

value denotes a better direction. The maximum obstacle distance, $d_{ob\max}$, is set to the maximum sensing range and the minimum obstacle distance is set to r_m. A smoothing function, *smooth*, is applied to obstacle distances to achieve better clearance around obstacles. Essentially, *smooth* is a one-dimensional filter that takes a weighted average of adjacent obstacle distances at θ_{dir}. α_1 and β are unit interval weightings for goal directness and obstacle clearance. Smaller weights and larger weights translate to lower and higher preferences respectively. Hence, an α_1 value of 0.5 and β value of 1 is sufficient to balance goal directedness and obstacle clearance for robots tested in this thesis.

$$o_{ds}(\theta_{dir}) = \alpha_1\left(1-\frac{|\theta_{dir}-\theta_t|}{\pi}\right)+\beta\left(\frac{smooth(d_{ob})}{d_{ob\max}}\right) \tag{3.10}$$

3.5 Modified Dynamic Window Method

In the dynamic window approach, a portion of the velocity space that is achievable within the next control cycle is searched for a velocity pair (v,ω). The achievable velocity space depends on the current velocities, acceleration and kinematic constraints and the shape of the robot. Originally, the dynamic window method [38] has been demonstrated on synchronous drive (holonomic) robots where linear and angular acceleration is independently varied at all speeds. However, in nonholonomic robots such as tricycle drive or differential drive robots, linear and angular acceleration are implicitly dependent. Linear and angular acceleration depend on the current velocity and kinematic constraints for these robot types.

The minimum velocity limit is dynamic in this thesis to enable robots to perform reversing manoeuvres when necessary. This is especially useful for tricycle drive robots as they cannot perform point turns. Also, the estimated distance to collision at velocity (v_{cd},ω_{cd}) DC_{cd} is not explicitly used to maximise the objective function since the direction sensor (section 3.4) adjusts the target heading using obstacle distances. Instead, DC_{cd} is employed as additional admissible velocity criteria to control reversing and avoid collisions.

Integrating the direction sensor with the dynamic window method requires the robot reference frame, R, to be translated to the physical centre of the robot as shown in Figure 3.5. The linear velocity v_1 and curvature radius r_1 also have to be adjusted to this new reference before the dynamic window approach can be applied. The curvature radius at the centre r is a function of the y axis offset in R and the minimum curvature radius r_{1min} (3.11). Linear velocity v at the centre of the robot can be computed from angular velocity ω and radius r (3.12).

$$r = \begin{cases} \sqrt{r_1^2 + \left|y_{off}\right|^2}, r_1 \geq r_{1\min} \\ -\sqrt{r_1^2 + \left|y_{off}\right|^2}, r_1 \leq -r_{1\min} \end{cases} \tag{3.11}$$

$$v = \frac{r}{r_1} v_1 = r\omega \tag{3.12}$$

A velocity-axis angle θ_v between velocity vector v and the robot frame y axis is calculated for each (v,ω) pair (3.13). This velocity-axis angle is employed to perform an axes rotation of the robot reference frame and obstacles so that the collision distance can be computed as described in [48].

$$\theta_v = \sin^{-1}\left(\frac{\left|y_{off}\right|}{r}\right) \tag{3.13}$$

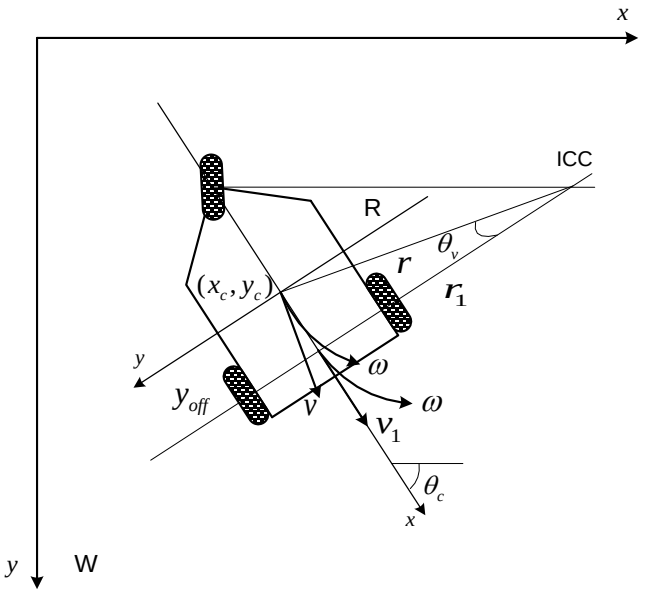


Figure 3.5: Modification of a robot's reference frame to allow compatibility between the direction sensor and dynamic window controls.

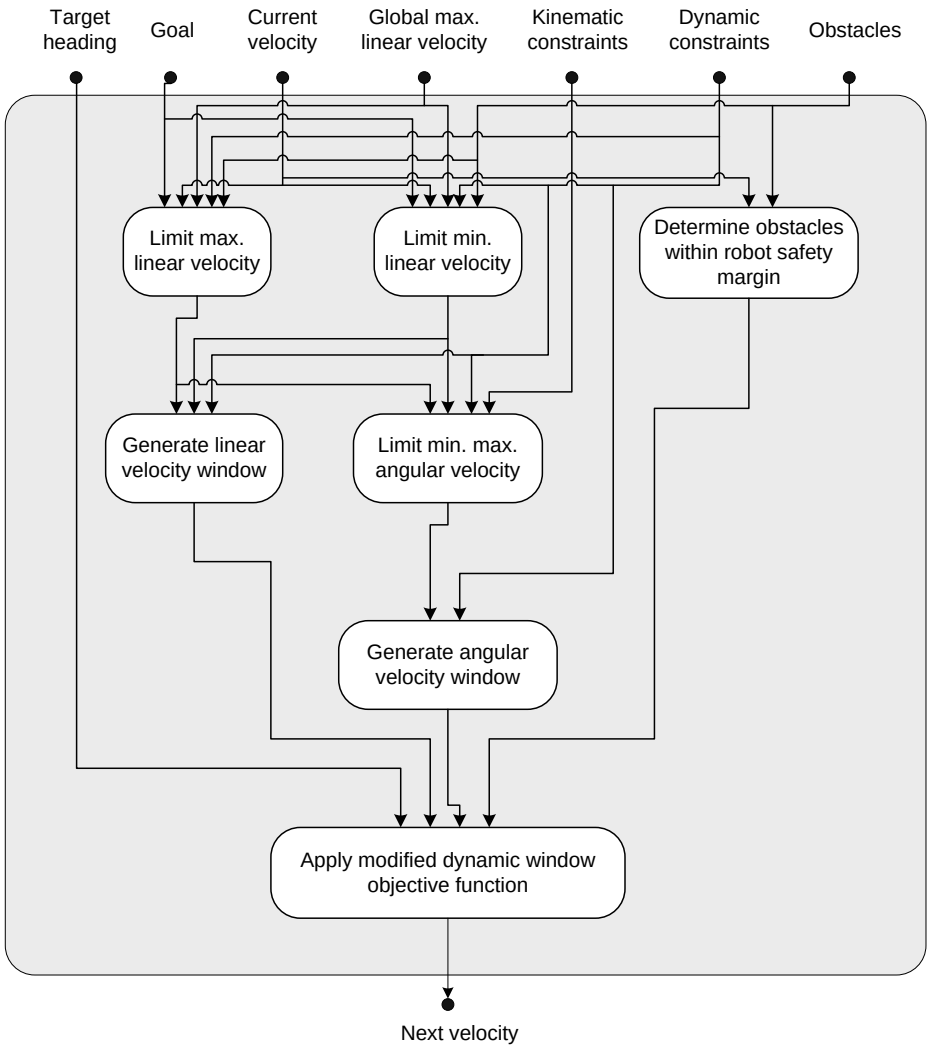


Figure 3.6: Overview of modified dynamic window method.

An overview of the modified dynamic window method employed on robots in this thesis is shown in Figure 3.6. There are seven major inputs to the algorithm. A target heading θ_{tmod} is the output from the direction sensor. The Euclidean distance to the final goal location d_{tgt}, current velocity (v_c, ω_c) and global maximum linear velocity $v_{g\max}$ are also input. Additionally, the kinematic constraints, dynamic constraints and obstacle distances d_{ob} are input to the dynamic window algorithm. These inputs limit the maximum and minimum linear and angular velocities used to generate velocity windows. The velocity windows, target heading and obstacles are evaluated with a modified dynamic window objective function (Figure 3.7) to select an optimal velocity pair (v_n, ω_n) for the next control cycle.

The maximum linear velocity $v_{\max}$ is derived from $v_{g\max}$ and varies depending on goal proximity d_{tgt} and front obstacle distances d_{obfr}. When the robot is within deceleration

and stopping distances, d_{decel} and d_{stop}, the goal maximum linear velocity limit $v_{gl\max}$ is linearly varied between $v_{g\max}$ and zero (3.14). The set of N_{fs} sensors situated at the front of the robot *FRS* determines the obstacle maximum linear velocity limit $v_{ob\max}$ (3.15)–(3.17). If the maximum front sensor distance d_{obfr} is greater than a minimum distance $d_{obfr\min}$ but less than a maximum limit $d_{obfr\max}$, $v_{ob\max}$ is linearly varied between $v_{g\max}$ and a minimum obstacle speed $v_{ob\min}$. The minimum of $v_{gl\max}$ and $v_{ob\max}$ is selected as $v_{\max}$ (3.18).

$$v_{gl\max} = \begin{cases} 0 & \text{if } d_{tgt} \le d_{stop} \\ v_{g\max} \cdot \dfrac{d_{tgt}}{d_{decel}} & \text{if } d_{stop} < d_{tgt} \le d_{decel} \\ v_{g\max} & \text{otherwise} \end{cases} \tag{3.14}$$

$$FRS = \left\{ d_{obfr1}, d_{obfr2}, ..., d_{obfrN_{fs}} \right\} \tag{3.15}$$

$$d_{obfr} = \min(FRS) \tag{3.16}$$

$$v_{ob\max} = \begin{cases} v_{ob\min} & \text{if } d_{obfr} \le d_{obfr\min} \\ v_{g\max} \cdot \dfrac{d_{obfr} - d_{obfr\min}}{d_{obfr\max} - d_{obfr\min}} + v_{ob\min} & \text{if } d_{obfr\min} < d_{obfr} \le d_{obfr\max} \\ v_{g\max} & \text{otherwise} \end{cases} \tag{3.17}$$

$$v_{\max} = \min(v_{gl\max}, v_{ob\max}) \tag{3.18}$$

To dynamically adjust the minimum linear velocity (for reversing) $v_{\min}$, obstacle distances at the front d_{obfr}, rear d_{obrr} and immediate rear d_{obirr} of the robot are obtained. The obstacle distance at the front of the robot is determined from the set of front sensors *FRS* using (3.15) and (3.16). Similarly, d_{obrr} and d_{obirr} are determined from the set of rear sensors *RRS* and immediate rear *IRRS* sensors respectively (3.19)–(3.22). The obstacle distances are compared with threshold values d_{obfrT1}, d_{obfrT2}, d_{obrrT} and d_{obirrT} to adjust the minimum velocity limit between zero and $-v_{g\max}$ (3.23). Reversing is subsumed if an obstacle is located directly behind the robot ($d_{obirr} < d_{obirrT}$) (3.23). Two threshold values are employed for the front sensors such that d_{obfrT2} is less than d_{obfrT1} (3.23). When d_{obfr} is less than d_{obfrT2}, a robot can back away from an obstacle. Forward motion can be resumed when d_{obfr} is greater than d_{obfrT1}. The minimum

velocity is also set to $-v_{g\max}$ when a tricycle drive robot is close to a goal location but requires orientation correction.

$$RRS = \left\{ d_{obrr1}, d_{obrr2}, ..., d_{obrrN_{rs}} \right\} \tag{3.19}$$

$$d_{obrr} = \min(RRS) \tag{3.20}$$

$$IRRS = \left\{ d_{obirr1}, d_{obirr2}, ..., d_{obirrN_{irs}} \right\} \tag{3.21}$$

$$d_{obirr} = \min(IRRS) \tag{3.22}$$

$$v_{\min} = \begin{cases} -v_{g\max} & \text{if } d_{obfr} < d_{obfrT2} \\ 0 & \text{if } d_{obirr} < d_{obirrT} \text{ and } d_{obfr} < d_{obfrT2} \\ 0 & \text{if } d_{obfr} > d_{obfrT1} \text{ or } d_{obrr} < d_{obrrT} \end{cases} \tag{3.23}$$

Dynamic constraints (linear acceleration a_a and deceleration a_d) and linear velocity limits ($v_{\min}$ and $v_{\max}$) are applied to the current velocity to produce a linear velocity window [$v_{w\min}$,$v_{w\max}$] for the next control cycle. The velocity window is discretised into an arbitrary number of divisions N_{vd} for evaluation.

The angular velocity of each robot has a global maximum $\omega_{g\max}$ and a global minimum $-\omega_{g\max}$. A minimum $c_{\min}$ and maximum $c_{\max}$ curvature for the next control cycle is determined from the current state, kinematic and dynamic constraints of a robot's drive. Dynamic constraints considered in a differential drive robot include the maximum linear wheel acceleration and deceleration while the state corresponds to the current linear speed of the robot's wheels. For a tricycle drive, the state corresponds to the current steering wheel angle while the steering rate is the dynamic constraint. Combinations of $c_{\min}$, $c_{\max}$, $v_{w\min}$ and $v_{w\max}$ are tested to determine the minimum and maximum angular velocity for the next control cycle (3.24)–(3.25).

$$\omega_{\min} = \min(c_{\max} v_{w\max}, c_{\min} v_{w\max}, c_{\max} v_{w\min}, c_{\min} v_{w\min}) \tag{3.24}$$

$$\omega_{\max} = \max(c_{\max} v_{w\max}, c_{\min} v_{w\max}, c_{\max} v_{w\min}, c_{\min} v_{w\min}) \tag{3.25}$$

The difference of $\omega_{\max}$ and $\omega_{\min}$ from current angular velocity ω_c is used to determine angular acceleration α_a and deceleration α_d. A minimum angular acceleration $\alpha_{\min}$

employed by the modified dynamic window objective function calculation is set to the minimum of the absolute values of α_a and α_d.

Dynamic constraints (angular acceleration α_a and deceleration α_d limits), angular velocity limits ($\omega_{\min}$ and $\omega_{\max}$) and global velocity limits are applied to the current angular velocity to produce an angular velocity window $[\omega_{w\min}, \omega_{w\max}]$ for the next control cycle. The angular velocity window is also discretised into an arbitrary number of divisions $N_{\omega d}$ for evaluation.

A safety margin SM is added to the robot's perimeter to allow it to stop or reverse before colliding with obstacles. If obstacle d_{ob} breaches the safety margin, a proximity flag PR_{ob} is set to enable velocity rejection in the obstacle's direction (3.26). The safety margin has a minimum value of $SM_{\min}$ and is increased proportionally based on the robot's current linear velocity v_c and a safety margin growth factor k_{SM} (3.27). Non-circular robots can have independent safety margins for each side of the robot. This arrangement enables the front and rear of the robot to have greater safety margins when travelling forward or reverse.

$$PR_{ob} = \begin{cases} 1 & \text{if } d_{ob} \le SM \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

$$SM = SM_{\min} + k_{SM} v_c \tag{3.27}$$

A flowchart of the process to select an optimal velocity pair (v_n, ω_n) for the next control cycle is shown in Figure 3.7. The angular and linear velocity windows are used to generate (v_{cd}, ω_{cd}) candidate velocity pairs for evaluation. Each candidate velocity pair is checked for curvature constraint satisfaction. Four curvature constraints, CC_1, CC_2, CC_3 and CC_4 are employed to determine the overall constraint satisfaction CC_T for tricycle robot candidate curvatures c_{cd} (3.28)–(3.32). These conditions coordinate forward and reverse movement while preventing deadlock at zero velocity. Candidate curvature c_{cd} needs to be within $[c_{\min}, c_{\max}]$ to satisfy the curvature constraint CC_D for differential drive robots.

$$CC_1 = (c_{cd} \le c_{\max}).(c_{cd} \ge c_{\min}).\overline{(v_{cd} = 0).(\omega_{cd} \ne 0)} \tag{3.28}$$

$$CC_2 = (c_{cd} \le c_{\max}).(c_{cd} \ge c_{\min}).(v_c > 0).(v_{cd} < 0) \tag{3.29}$$

$$CC_3 = (c_{cd} \le c_{\max}).(c_{cd} \ge c_{\min}).(v_c < 0).(v_{cd} > 0) \tag{3.30}$$

$$CC_4 = \overline{(v_c = 0).(v_{cd} = 0)} \tag{3.31}$$

$$CC_T = (CC_1 \mid CC_2 \mid CC_3).CC_4 \tag{3.32}$$

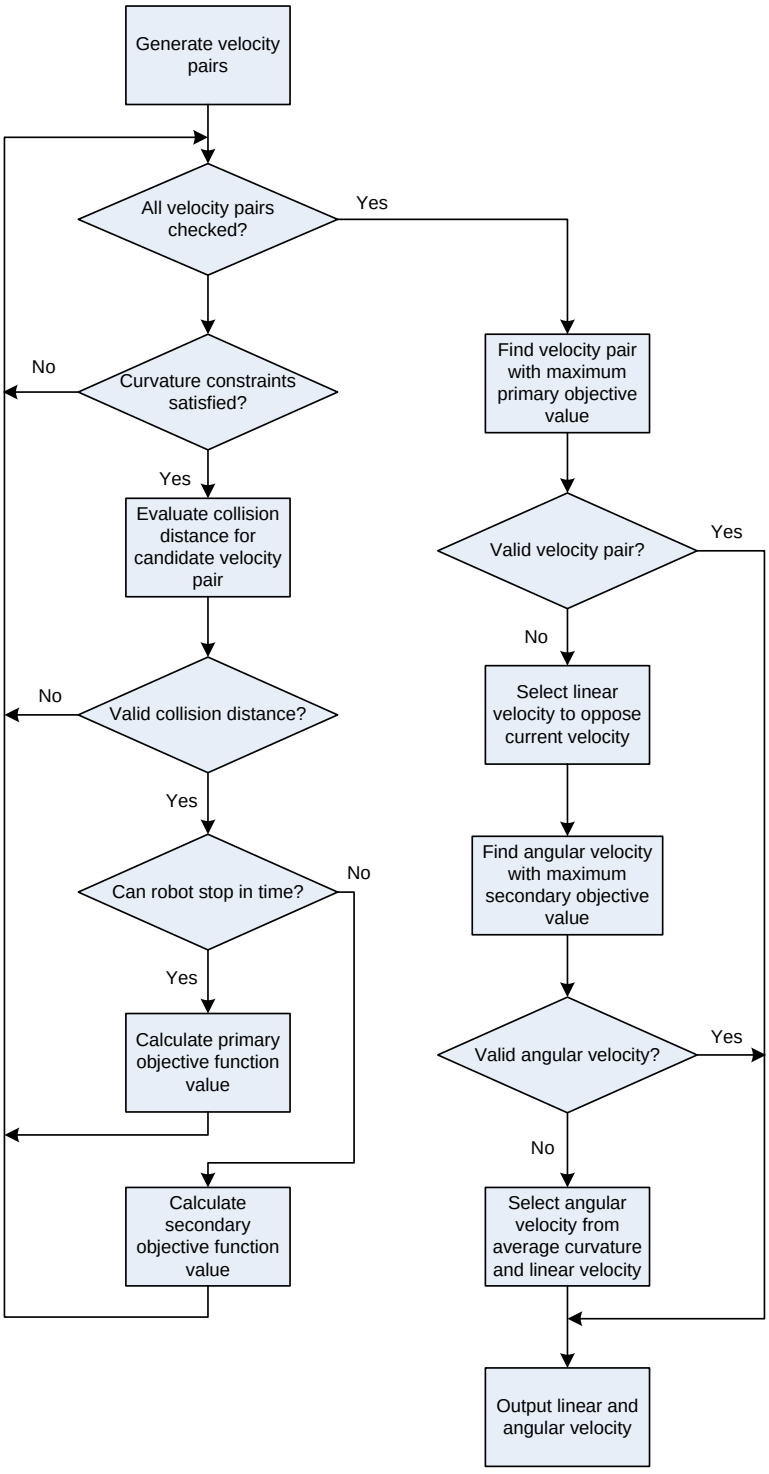


Figure 3.7: Flowchart of optimal velocity pair selection.

Next, the distance to collision DC_{cd} if the robot travels at the candidate velocity pair is determined [48]. Tricycle robot collision distances are multiplied by a reduction factor *DCF* if the candidate velocity pair steers the robot towards a close obstacle CL_{ob} on the side the robot. The reduction factor attempts to reduce instances of the robot being stuck in v-shaped corners that can result from closely positioned obstacles (Figure 3.8). If the distance to collision is satisfactory ($DC_{cd} > 0$), it is used to determine if the robot can stop in sufficient time to avoid collision. The Boolean variables ST_v and ST_ω represent the robot's ability to successfully stop at the candidate linear and angular velocities respectively (3.33)–(3.34). Tricycle drive robots utilise only ST_v for stopping distance checking.

$$ST_v = \begin{cases} 1 & \text{if } DC_{cd} > v_{cd}^2 / 2.a_a \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

$$ST_\omega = \begin{cases} 1 & \text{if } DC_{cd} > \omega_{cd}^2 / 2.\alpha_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{3.34}$$

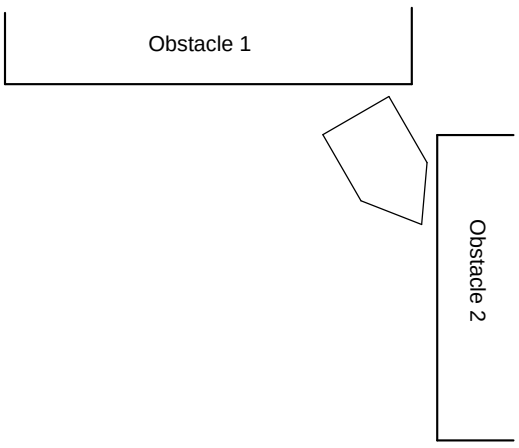


Figure 3.8: Tricycle robot stuck between closely positioned obstacles.

If the robot is able to stop in sufficient time to avoid collision, a primary objective function value is computed for the candidate velocity pair (3.35). A secondary objective function value is calculated for ω_{cd} if the robot is unable to stop in time. The secondary objective function attempts to steer the robot away from an obstacle even if it is too close to stop. Secondary objective functions $o_{dwst}(\omega_{cd})$ and $o_{dwsd}(\omega_{cd})$ are calculated for tricycle and differential drive robots respectively (3.36)–(3.37). Smaller

and larger weightings for each parameter translate to lower and higher preferences respectively. Hence, an α_2 value of 0.4 and γ value of 0.2 is sufficient to balance goal directedness and velocity for robots tested in this thesis. If α_2 is reduced, the robot may not be drawn to the goal location. On the other hand, an increase in α_2 can compromise obstacle avoidance as the robot may not deviate from its goal direction. Similarly, the robot may not travel at a reasonable speed if γ is too low. A large value of γ can compromise obstacle avoidance since it favours higher speeds. An α_3 value of 0.01 is sufficient to reduce objective values when collision distance DC_{cd} is below threshold $DC_{cd\min}$.

$$o_{dwp}(v_{cd},\omega_{cd}) = \begin{cases} \alpha_2\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right)+\gamma\left(\frac{|v_{cd}|}{v_{\max}}\right) & \text{if } ST_v \text{ and } DC_{cd} > DC_{cd\min} \\ \alpha_3\left(\alpha_2\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right)+\gamma\left(\frac{|v_{cd}|}{v_{\max}}\right)\right) & \text{if } ST_v \text{ and } DC_{cd} \le DC_{cd\min} \\ -1 & \text{otherwise} \end{cases}$$

(3.35)

$$o_{dwst}(\omega_{cd}) = \begin{cases} \alpha_2.\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right) & \text{if } \overline{CL_{ob}} \text{ and } \overline{ST_v} \text{ and } DC_{cd} \le DC_{cd\min} \\ \alpha_2.\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right).\frac{DC_{cd}^2}{DC_{cd\max}^2} & \text{if } CL_{ob} \text{ and } \overline{ST_v} \text{ and } DC_{cd} \le DC_{cd\min} \\ -1 & \text{otherwise} \end{cases}$$

(3.36)

$$o_{dwsd}(\omega_{cd}) = \begin{cases} \alpha_2.\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right) & \text{if } \overline{ST_v} \text{ and } \overline{(v_c=0).(v_{cd}=0)} \\ \alpha_2.\left(\frac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right) & \text{if } \overline{ST_v} \text{ and } ST_\omega \text{ and } (v_c=0).(v_{cd}=0) \\ -1 & \text{otherwise} \end{cases}$$

(3.37)

After all velocity pairs have been checked and evaluated, a velocity pair with the maximum primary objective value is searched for. If a valid velocity pair is found then it is set as the target velocity for the next control cycle. However, if a valid pair is not found then a linear velocity that opposes the current direction is selected. This enables the robot to avoid collisions in the current direction of movement. Candidate angular velocities are searched for the maximum secondary objective value. If a valid angular velocity is obtained then it is set as the angular velocity for the next control cycle. When the primary and secondary objective value searches both return invalid data, the next angular velocity is determined from the average curvature and linear velocity that opposes the current motion.

3.6 Simulation Experiments

The navigation system has been implemented for simulated robotic agents using MATLAB® 2007a. Figure 3.9 illustrates the MATLAB simulator developed to test the navigation system. Robot A (Table 3.1) is travelling from the top-left corner to bottom-right corner of a 20% obstacle density environment. The simulator accounts partially for uncertainties and delays by adding and filtering Gaussian noise to sensing and actuation signals.

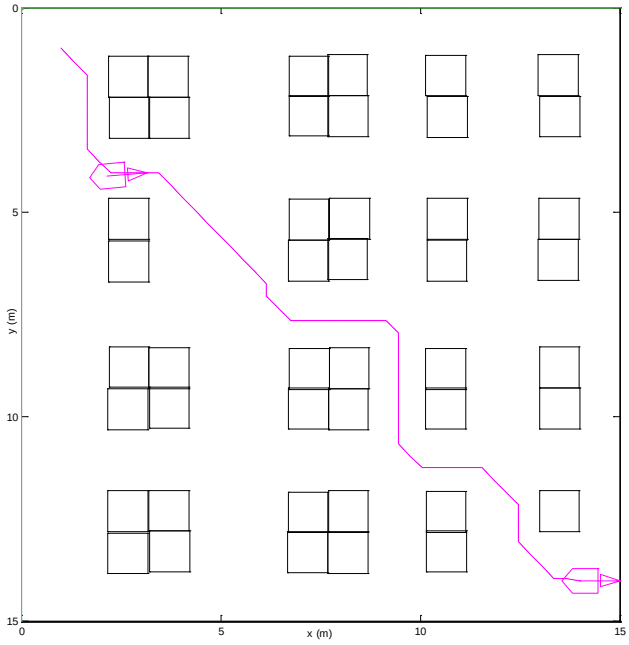


Figure 3.9: Tricycle robot traversing in a known environment.

Table 3.1: Simulated robot attributes.

Robot		**A**	**B**	**C**
Shape		pentagon	circle	rectangle
Drive		tricycle	differential	differential
Size (minimum circular radius) [m]		0.5	0.35	0.67
Max. linear velocity $v_{q\max}$ [m/sec]		0.5	0.5	0.3
Max. angular velocity $\omega_{q\max}$ [rad/sec]		$\pi/7$	$\pi/7$	$\pi/7$
IR Sensors	**Qty**	11	10	14
	Range [m]	1.5	1.5	1.5

3.6.1 Parameter Tuning

The tested robots (Table 3.1) used the direction sensor parameters outlined in Table 3.2. The direction sensor parameters have been empirically tuned. Initially, $d_{ob\max}$ is set to the maximum range of the infrared sensors. Next, N_θ is selected to balance computational effort and direction resolution. Finally, α_1 followed by β is adjusted to balance goal-directness and obstacle avoidance.

The modified dynamic window parameters (Table 3.3) have also been empirically tuned. Acceleration a_a and deceleration a_d are determined from the robot's dynamic constraints. Parameters that are largely independent of obstacles can then be tuned. Initially, α_2 and γ are tuned to attract the robot to a goal in environments with minimal obstacles. Next, N_{vd} and $N_{\omega d}$ can be adjusted to balance computational speed and motion smoothness. Following this, the parameters for decelerating and stopping the robot are determined. Stopping distance d_{stop} is calculated from velocity profile data [131], while decelerating distance d_{decel} is tuned to slow down the robot when it is near the goal location.

Table 3.2: Direction sensor parameter data.

Parameter	**Numerical Value**
α_1	0.5
β	1
$d_{ob\max}$	1.5 m
N_θ	41

Table 3.3: Modified dynamic window parameter data.

Parameter	Numerical Value
a_a	0.1 m/sec^2
a_d	-0.1 m/sec^2
α_2	0.4
α_3	0.01
γ	0.2
$DC_{cd\max}$	3 m
$DC_{cd\min}$	0.6 m
DCF	0
d_{decel}	2 m
$d_{obfr\max}$	2 m
$d_{obfr\min}$	0.5 m
d_{obfrT1}	0.95 m
d_{obfrT2}	0.75 m
d_{obirrT}	0.5 m
d_{obrrT}	0.7 m
d_{stop}	0.3162 m
k_{SM}	0.1 sec
N_{vd}	5
$N_{\omega d}$	11
$v_{ob\min}$	0.2 m/sec

Obstacle dependent parameters can be tuned after obstacle independent parameters have been tuned. The parameters used to determine $v_{ob\max}$ can be tuned first. $d_{obfr\max}$ and $d_{obfr\min}$ are approximately the maximum and minimum sensing ranges relative to the centre of the robot respectively. $v_{ob\min}$ is selected such that the robot can stop within a couple of control cycles if necessary. Next, the front and rear obstacle distances (d_{obfrT1}, d_{obfrT2}, d_{obirrT}, and d_{obrrT}) to control the minimum velocity limit for reversing are empirically tuned.

Parameters employed in optimal velocity pair selection can be tuned next. Maximum collision distance $DC_{cd\max}$ is determined from curvature data and the maximum sensing range of the robot. Safety margin growth gain k_{SM} is tuned to ensure that higher linear velocities in the dynamic window are rejected quickly when the robot travels close to its maximum velocity. Minimum collision distance $DC_{cd\min}$ and α_3 are tuned to reduce the preference of candidate velocities before ST_v becomes zero. Collision distance reduction factor DCF is tuned to quickly reject candidate velocities that steer a robot toward close side obstacles.

All parameters can be further tuned if the combined performance is not adequate.

3.6.2 Experimental Configurations

Table 3.1 details the attributes of three robots employed to evaluate the navigation system's performance. Evenly distributed obstacles at 5%, 10%, 15% and 20% densities were automatically generated for 15 m × 15 m worlds. It was computationally difficult to achieve evenly distributed automatically positioned obstacles at higher obstacle densities. Ten worlds were generated for each obstacle density. The performance of pure reactive and hybrid reactive-deliberative navigation in known and unknown environments has been evaluated. Experiments on each navigation technique were repeated ten times for all worlds tested. This produced a total of 100 samples for each obstacle density.

In pure reactive control, the robot attempts to travel in a straight line to the goal location while avoiding any obstacles it encounters. A path is initially planned and tracked while the robot navigates towards the goal in the hybrid reactive-deliberative approaches. In unknown environment navigation, the robot regularly updates the occupancy grid map and re-plans accordingly as it travels towards the goal.

Three additional configurations have been evaluated to compare the performance of the final reactive controller (FRC). Firstly, the performance of employing only the original dynamic window (ODW) method for reactive navigation is evaluated. Secondly, the original dynamic window method coupled with the developed direction sensor is evaluated (ODWDS). Finally, a reduced parameter modified dynamic window method is coupled with the developed direction sensor to produce a reduced final reactive controller (RFRC).

The original dynamic window method does not employ (3.18)–(3.23), (3.28)–(3.32) and (3.35)–(3.37). Hence, $v_{ob\max}$ is set to $v_{g\max}$ while $v_{\min}$ is set to $-v_{g\max}$. The tricycle curvature constraint CC_T is determined in a similar manner to the differential drive curvature constraint CC_D. Additionally, tricycle robot collision distances are not influenced by the reduction factor *DCF*. Equations (3.35)–(3.37) are replaced with (3.38). Obstacle clearance weight β_1 has been empirically tuned to 0.3. A β_1 weight of

0.005 has been tuned when the original dynamic window method is combined with the direction sensor. The direction sensor's ability to provide obstacle clearance causes the reduced weighting.

$$o_{dw}(v_{cd},\omega_{cd},DC_{cd})=\begin{cases}\alpha_2\left(\dfrac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right)+\gamma\left(\dfrac{\left|v_{cd}\right|}{v_{\max}}\right)+\beta_1\left(\dfrac{DC_{cd}}{DC_{cd\,\max}}\right) & \text{if } ST_v \text{ and } ST_\omega \\ -1 & \text{otherwise}\end{cases} \tag{3.38}$$

In the reduced parameter modified dynamic window method, parameters considered to have the least impact on performance have been removed. For tricycle robots, the collision distance reduction factor *DCF* is set to 1 and close obstacles CL_{ob} are not monitored. Weight α_3 is removed from the primary objective function (3.35) to produce (3.39). With the removal of CL_{ob}, the secondary objective function for tricycle robots (3.36) is reduced to (3.40).

$$o_{dwp}(v_{cd},\omega_{cd})=\begin{cases}\alpha_2\left(\dfrac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right)+\gamma\left(\dfrac{\left|v_{cd}\right|}{v_{\max}}\right) & \text{if } ST_v \\ -1 & \text{otherwise}\end{cases} \tag{3.39}$$

$$o_{dwst}(\omega_{cd})=\begin{cases}\alpha_2.\left(\dfrac{\left|(\theta_{t\,\mathrm{mod}}-\theta_c)-\omega_{cd}\right|}{\pi}\right) & \text{if } \overline{ST_v} \\ -1 & \text{otherwise}\end{cases} \tag{3.40}$$

It is unsuitable to compare the modified dynamic window method with the original approach since the modified method is tailored for use with the direction sensor. Consequently, unlike the original method, the modified dynamic window does not explicitly maximise obstacle clearance in its objective function.

Each of the navigation approaches has been evaluated for navigation time *NT*, path length index *PI*, success rate *SR* and average velocity *AV*. Navigation time is limited to 300 sec for 5% and 10% obstacles, while it is restricted to 450 sec for the remaining obstacle densities. Path length index is determined by comparing the length of the optimal planned path with the actual path traversed by the robot. It is possible for a robot to achieve a path length index greater than unity since the length of the optimally planned path may not necessarily be the shortest path. The optimally

planned path is not always the shortest path because obstacle clearance is also accounted for during path planning. Success rate, path length index and average velocity have been combined with equal weightings to produce a score representing overall success *OS*.

$$OS = \left(SR + PI + AV \right) / 3 \tag{3.41}$$

In the results figures, each bar represents the average value and the corresponding error bar illustrates standard deviation. A paired sample t-test with two-sided p-values is used to compare the results of employing the various navigation approaches. Comparisons are statistically significant if p-values are less than or equal to 0.05 (5% statistical significance level). Performance ratios are computed to determine superiority or inferiority by dividing the results obtained from one method by the results obtained from another method.

3.6.3 Results

Figure 3.10 – Figure 3.15 illustrates the performance of the various navigation strategies for the tested robots over obstacle density combinations. Each bar in the graphs represents the mean value of the test configuration (obstacle density – robot combination). The error bars denote the standard deviation of the tested configurations.

Generally, the FRC, HRDK, HRDU, ODWDS and RFRC methods show similar trends for each robot as obstacle density is varied. As obstacle density increases, navigation time increases. Path length index generally decreases with increasing obstacle density. Success rate tends to decrease for Robots A and B as obstacle density increases. Average travelling velocity also decreases with obstacle density. The kinematic constraints of Robot A (tricycle robot) generally cause it to travel longer paths, requiring greater navigation time, than Robots B and C. Robots B and C have the advantage of point turn manoeuvres since they are differential drive robots. However, Robot C is the largest robot and its success rate deteriorates at higher obstacle densities. Robot B, the smallest robot, has a circular shape and produces very high success rates for all navigation strategies except the ODW method.

The FRC results are presented in Figure 3.10 (a)–(e). This navigation strategy can work well on all three of the tested robots. The navigation time for all robots is less than the maximum set limit (Figure 3.10 (a)). Good path length indices (>0.83) can be achieved with Robots B and C (Figure 3.10 (b)).

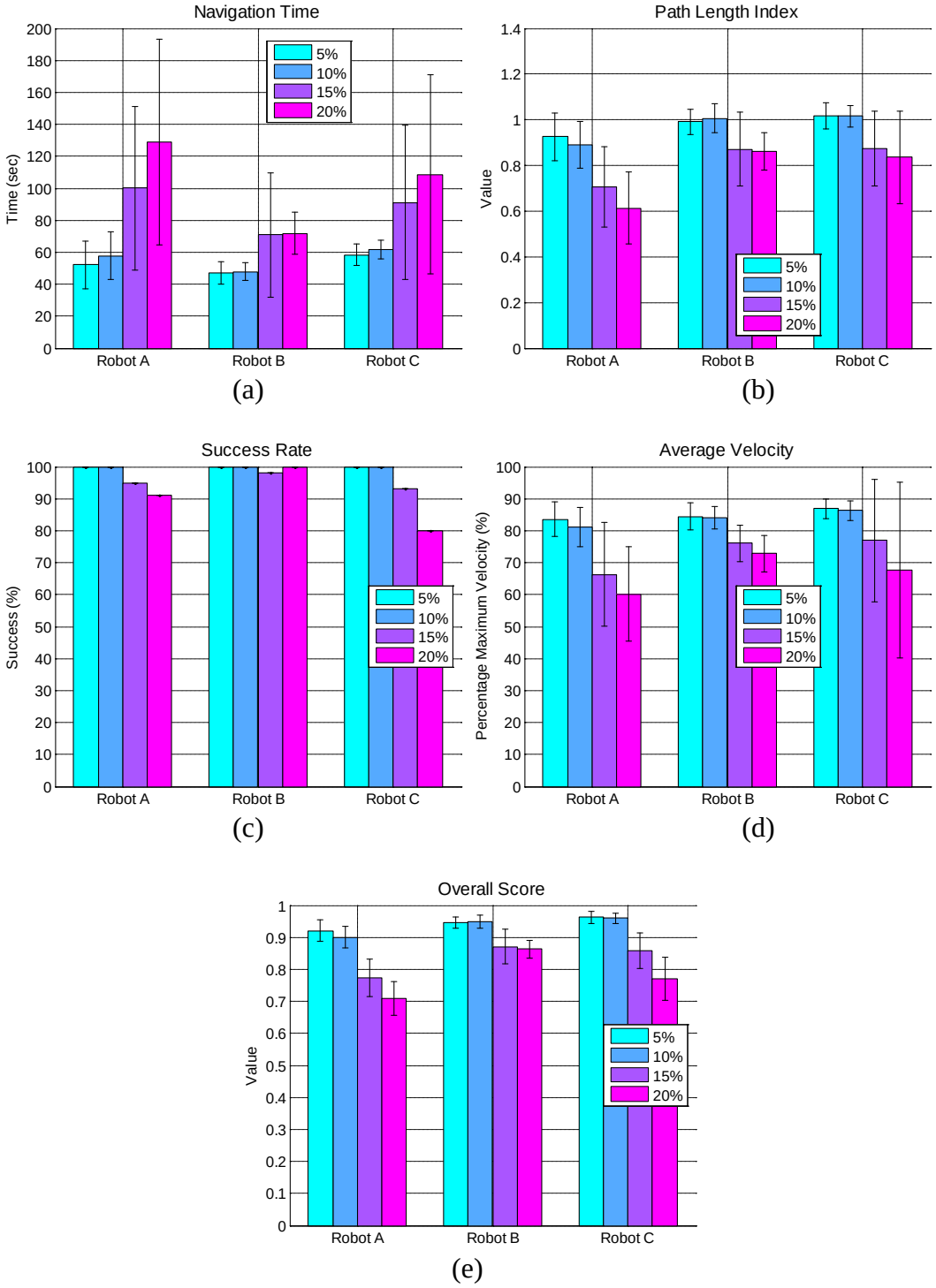


Figure 3.10: Final reactive controller (FRC) results.

Robot C's success rate in 20% obstacle density environments is limited to 80%, while Robots A and B achieve higher success rates (>88%) (Figure 3.10 (c)). The mean overall scores of Robots A, B and C are greater than 0.75, 0.86 and 0.77 respectively (Figure 3.10 (e)).

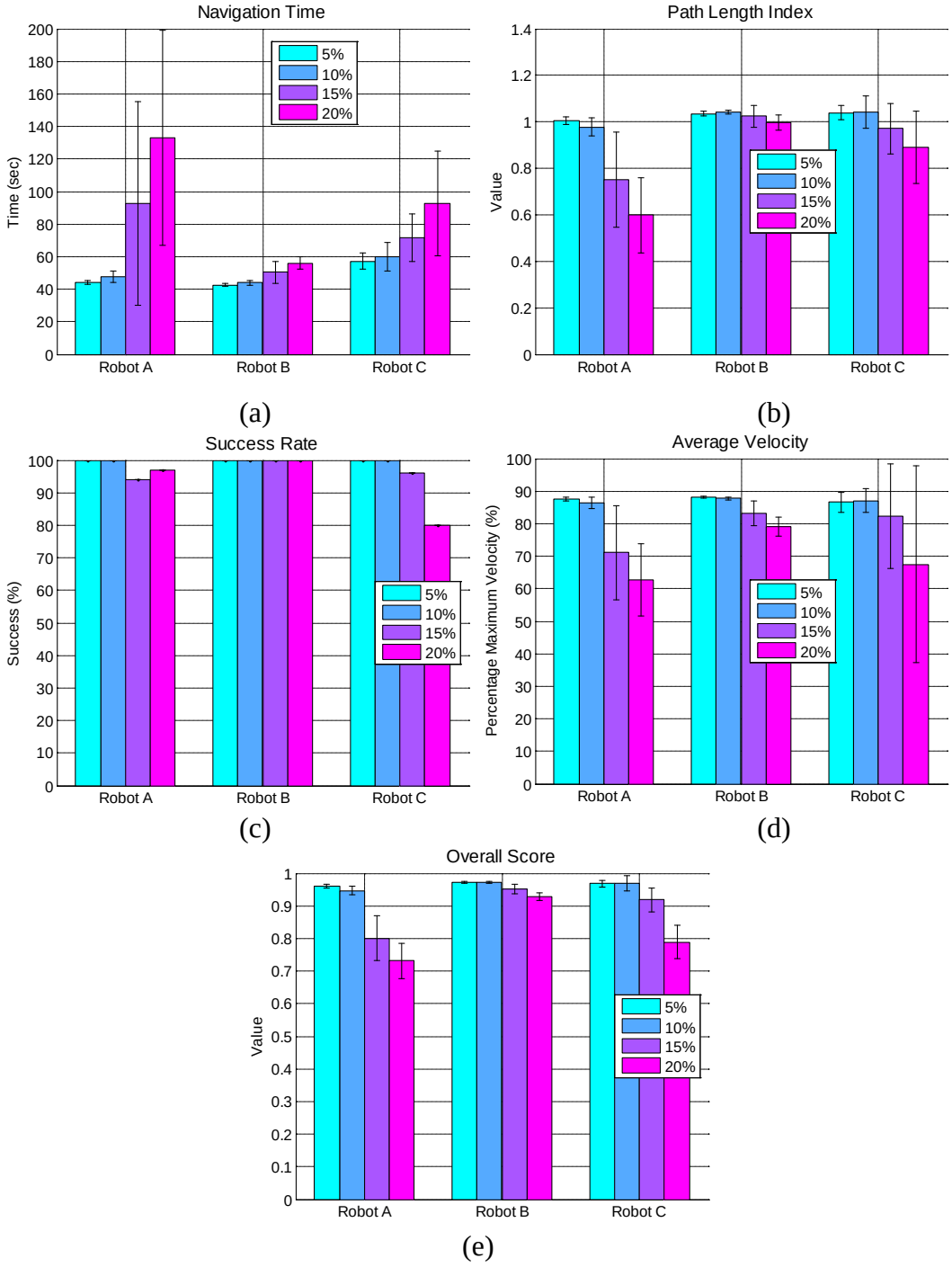


Figure 3.11: Results for hybrid reactive-deliberative navigation in known environments (HRDK).

Figure 3.11 (a)–(e) illustrates the results for HRDK navigation. Again, this method works well on all three tested robots. Good path length indices (>0.89) are achievable for Robots B and C at all obstacle densities (Figure 3.11 (b)). High success rates (>94%) are possible for Robots A and B (Figure 3.11 (c)). The mean overall score of each robot – obstacle density combination is comparable and varies between 0.73 and 0.97 (Figure 3.11 (e)).

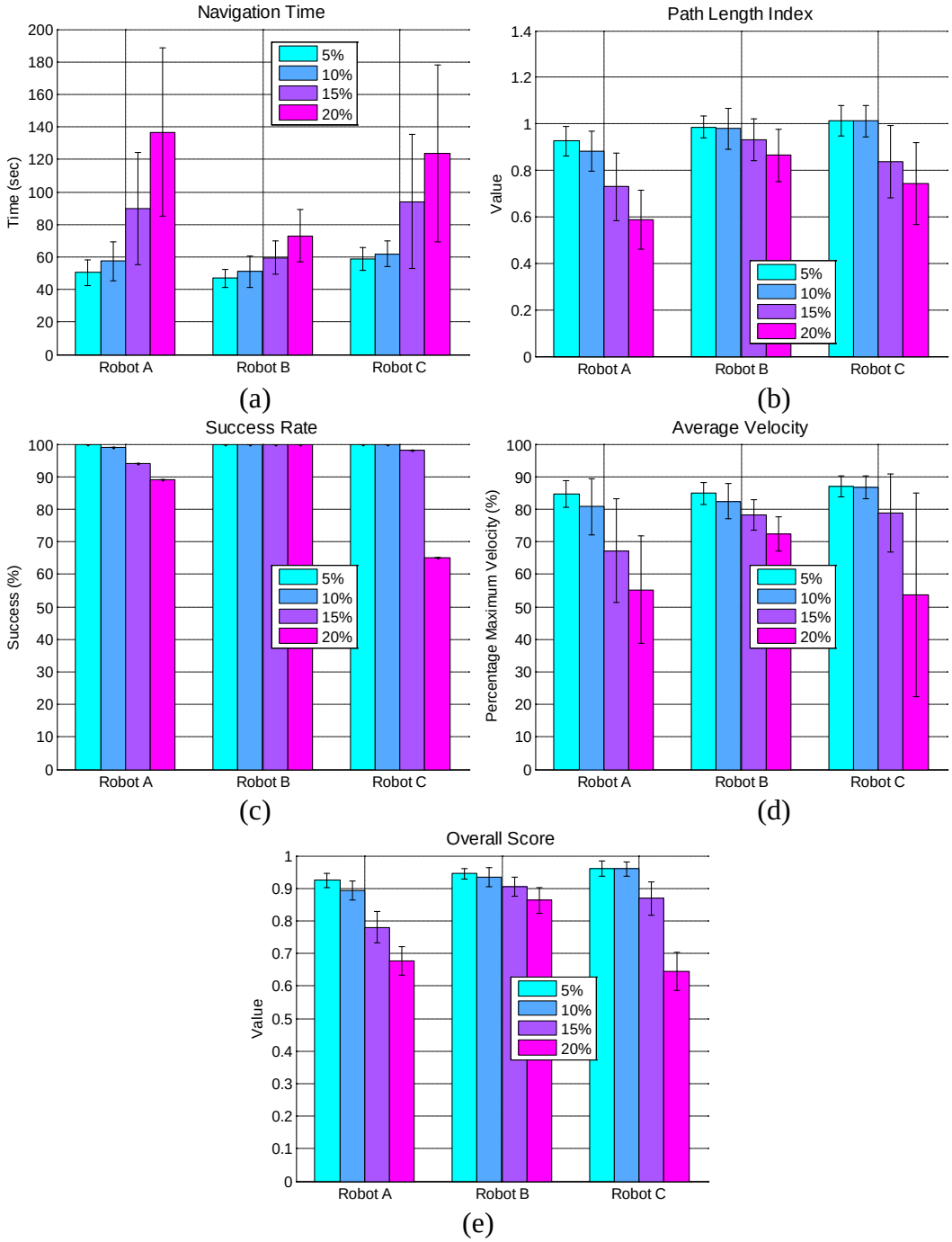


Figure 3.12: Hybrid reactive-deliberative navigation in unknown environments (HRDU).

Hybrid reactive-deliberative navigation experiments in unknown environments (HRDU navigation) results are shown in Figure 3.12 (a)–(e). This navigation technique can also work on all three robots that have been tested. Navigation time varies between 46 sec and 136 sec for the different combinations tested (Figure 3.12 (a)). Good path length indices (>0.80) are achievable with Robot B (Figure 3.12 (b)). Robots A and C achieve path length indices greater than 0.59 and 0.74 respectively (Figure 3.12 (b)).

Robot B can achieve 100% success in the tested environments (Figure 3.12 (c)). Robot A has a good success rate (>81%) while Robot B can attain a success rate greater than 65% (Figure 3.12 (c)). The mean overall scores of Robots A, B and C are greater than 0.68, 0.85 and 0.77 respectively (Figure 3.12 (e)).

The evaluation of the ODW method is presented in Figure 3.13 (a)–(e). This method yields poor performance on all the tested robots. Despite the average velocity of each robot being greater than 53% (Figure 3.13 (d)), navigation time is significantly high (Figure 3.13 (a)), indicating poor goal-directedness. The low path length index values (<0.48) for all robots also indicate poor goal-directedness when relying solely on the DWA method for navigation (Figure 3.13 (b)).

Extremely poor success rates (<5%) have been attained for Robots A and C (Figure 3.13 (c)). Robot B produces above average success rates at 15% and 20% obstacle densities (Figure 3.13 (c)). Consequently, the overall scores are also poor with only Robot B achieving above the 0.5 level at 15% and 20% obstacle densities (Figure 3.13 (e)).

When the direction sensor is combined with the ODW method, it can perform adequately on all the tested robots (Figure 3.14 (a)–(e)). The average velocity achieved by the robots is good (>70%) (Figure 3.14 (d)). For Robots B and C, above average path length indices (>65%) are achievable (Figure 3.14 (b)). However, path length index values are poor (0.28–0.50) for Robot A at 15% and 20% obstacle densities (Figure 3.14 (b)).

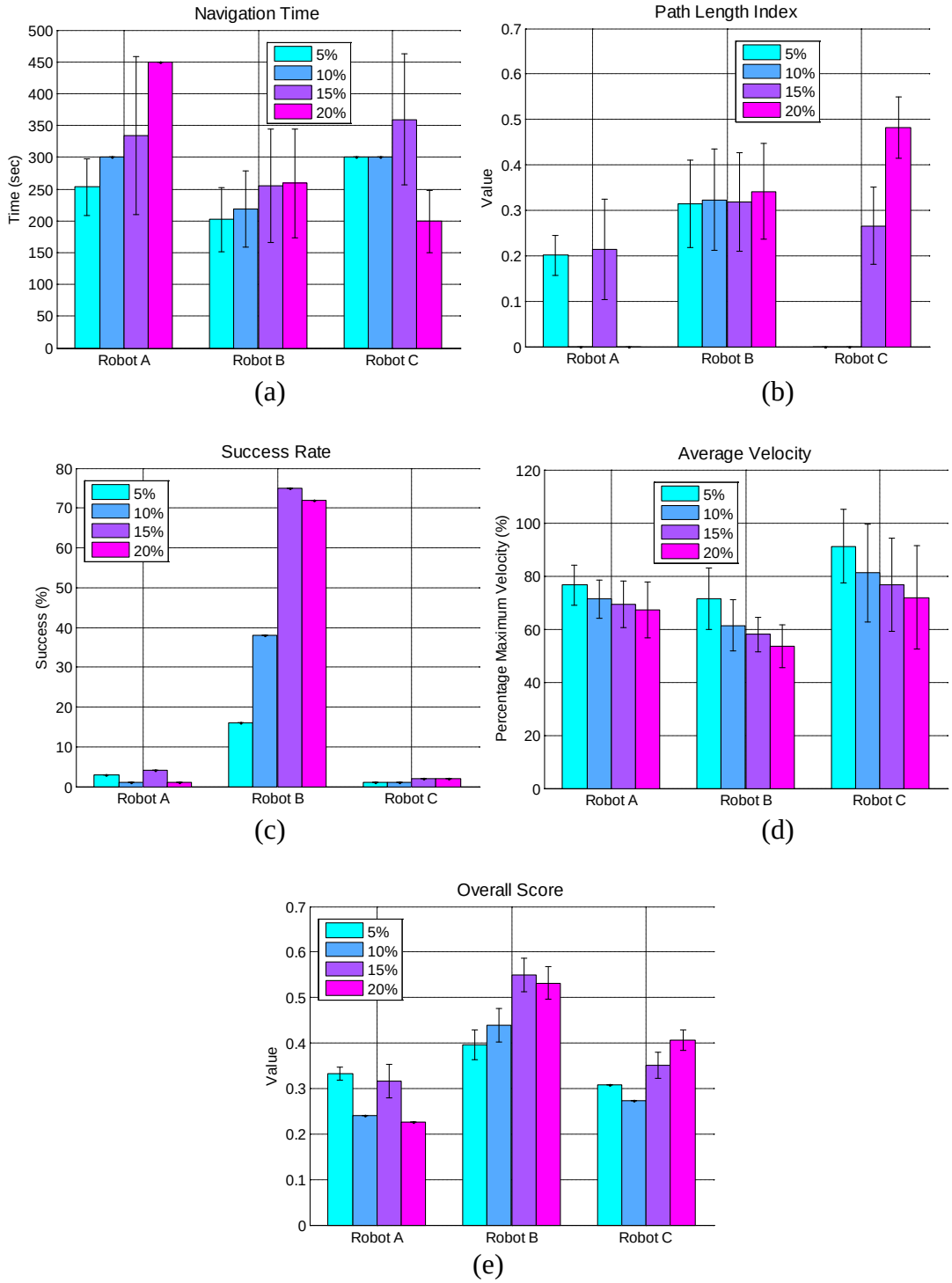


Figure 3.13: Original dynamic window (ODW) reactive navigation results.

The success rate of Robot B is high (>94%) for all obstacle densities (Figure 3.14 (c)). However, Robots A and C have significantly lower success rates at higher obstacle densities (Figure 3.14 (c)). The mean overall scores of Robots A, B and C are greater than 0.51, 0.78 and 0.64 respectively (Figure 3.14 (e)).

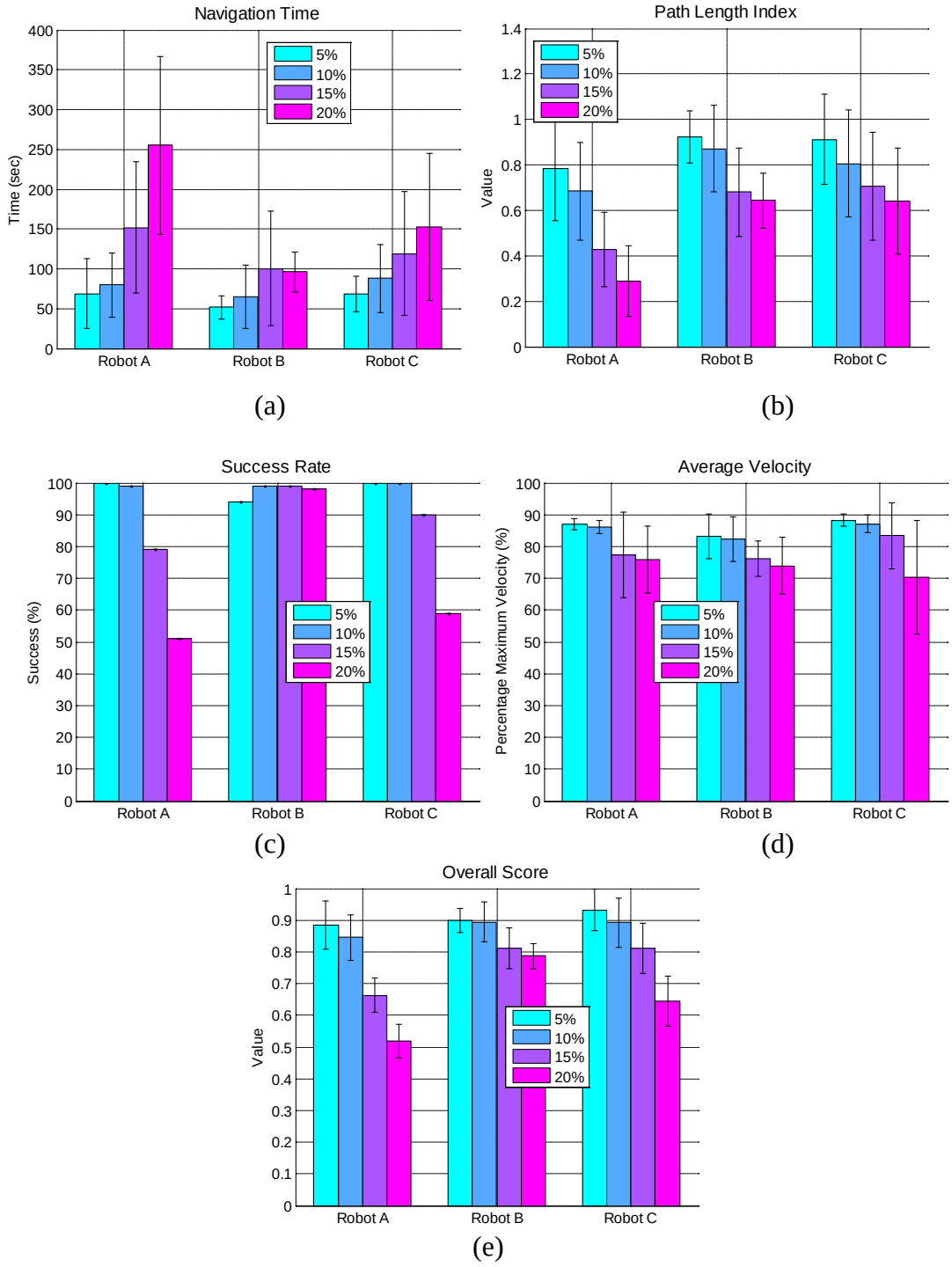


Figure 3.14: Reactive navigation with original dynamic window approach and direction sensor (ODWDS).

A reduced parameter reactive controller (RFRC) has also been evaluated (Figure 3.15 (a)–(e)). Similar to the FRC, this navigation strategy is capable of performing on all the tested robots. The navigation time for all robots is less than the maximum set limit (Figure 3.15 (a)). Robots B and C have good path length indices (>0.81) (Figure 3.15 (b)). Robot B has a high success rate (>91%), while Robots A and C have at least

76% and 75% success respectively (Figure 3.15 (c)). The mean overall score of each robot – obstacle density combination is comparable and varies between 0.70 and 0.94 (Figure 3.15 (e)).

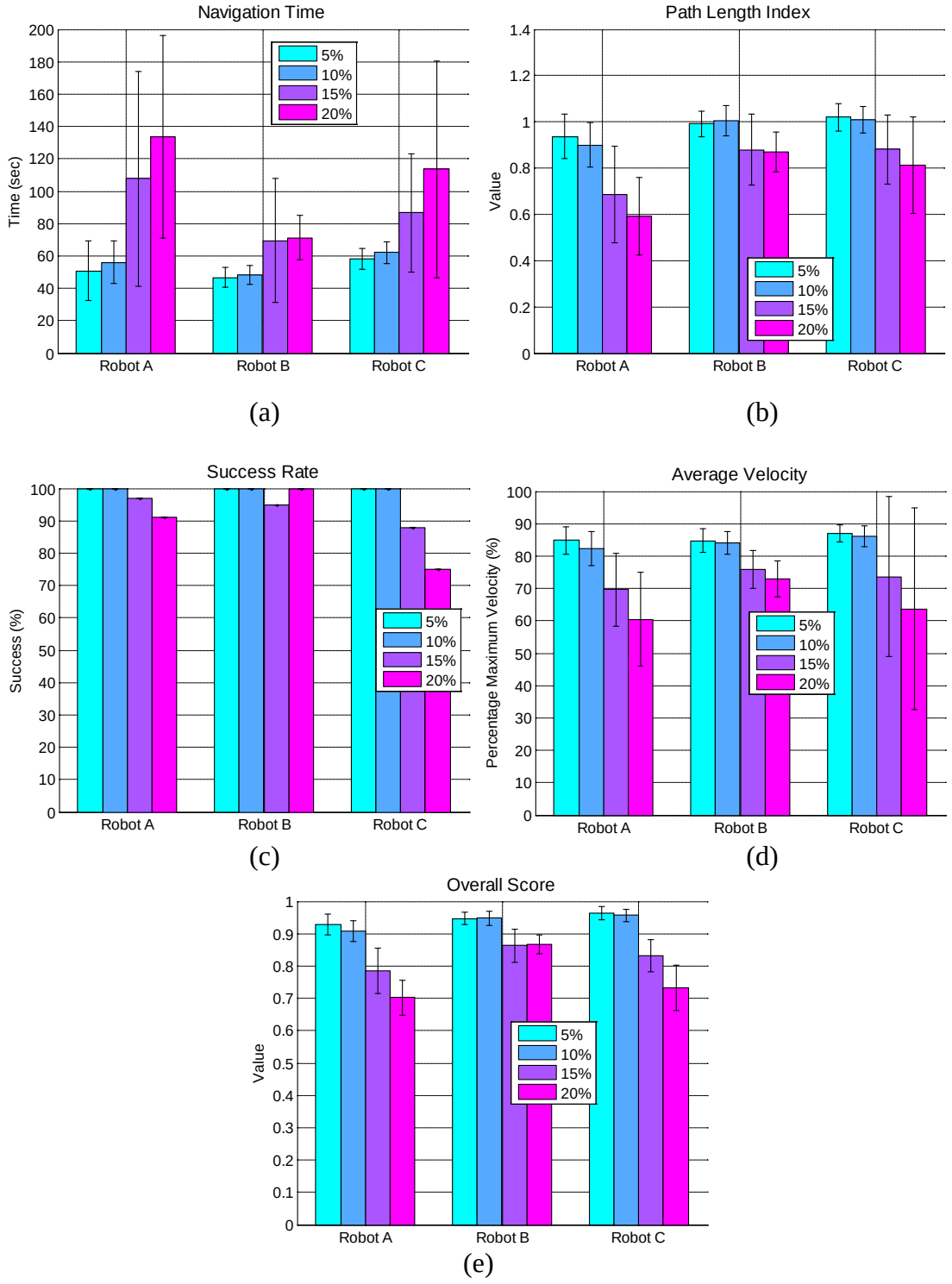
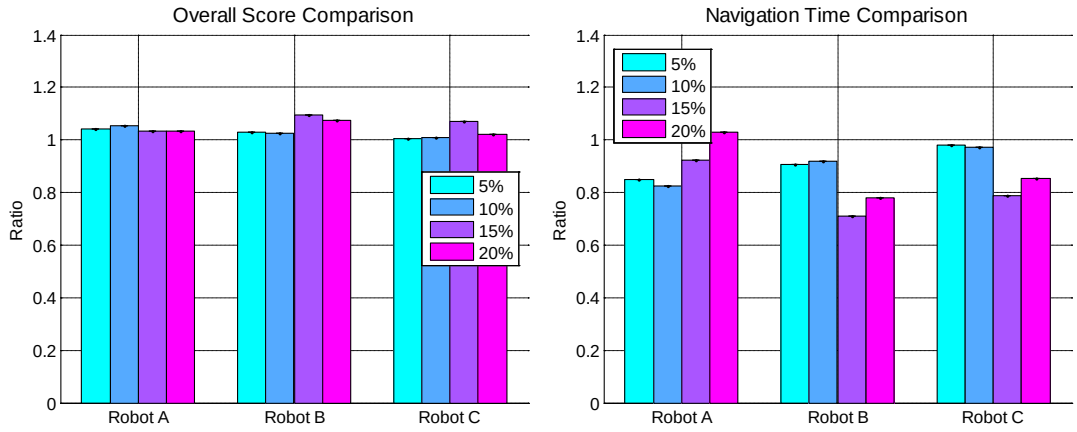


Figure 3.15: Reduced parameter reactive controller (RFRC) results.

A comparison of the overall scores and navigation times of the various navigation strategies is illustrated in Figure 3.16 – Figure 3.20. In each case, the alternative navigation methods are compared with the final reactive controller (FRC). The first two parts of each figure ((a) and (b)) illustrates the overall score comparison's ratio and navigation time ratio, respectively. Overall score ratio is calculated by dividing the alternative navigation technique's overall score by the FRC overall score. A ratio less than unity represents inferior performance while a ratio greater than unity indicates superior performance. The alternative navigation technique's navigation time divided by the FRC navigation time determines the navigation time ratio. Superior performance is denoted by a ratio less than unity, while a ratio greater than unity indicates inferior performance. The third part of each figure ((c)) shows the p-values for the overall score and navigation time comparisons. P-values less than 0.05 (5% statistical significance level) are highlighted in blue to indicate comparisons that are statistically significant.



(a) (b)

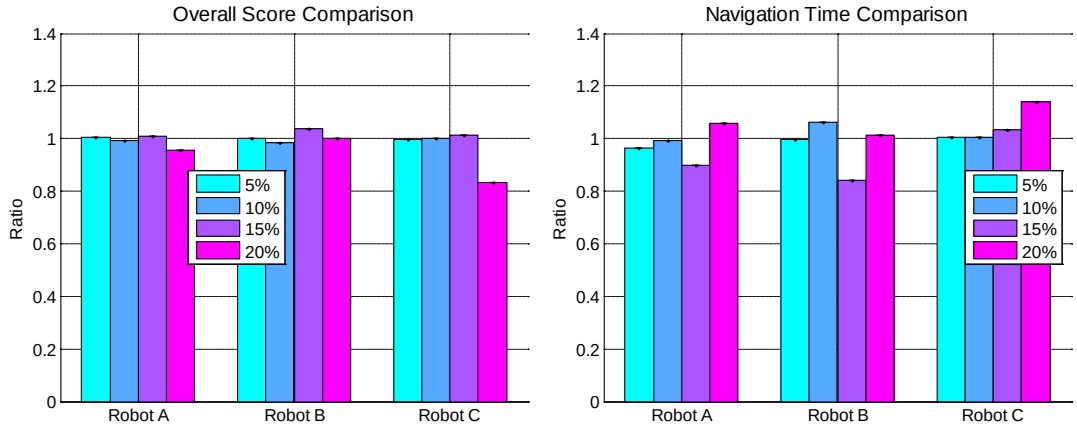
p-values (two decimal places, < 0.05 shaded blue)						
Obstacle Density	Overall Score Comparison (a)			Navigation Time Comparison (b)		
	Robot A	Robot B	Robot C	Robot A	Robot B	Robot C
5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.08
10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.06
15%	< 0.01	< 0.01	< 0.01	0.18	< 0.01	< 0.01
20%	< 0.01	< 0.01	0.02	0.34	< 0.01	0.01

(c)

Figure 3.16: Comparison of HRDK navigation and FRC.

Figure 3.16 (a)–(c) shows a comparison of HRDK navigation and the FRC. For all test configurations, the overall score ratio is greater than unity (Figure 3.16 (a)). The

comparison is statistically significant for all cases (Figure 3.16 (c)), with best case improvement for HRDK navigation performance of 8% (Robot B in 15% obstacle density environment). Figure 3.16 (c) also indicates that the comparison is statistically significant for majority of the cases even at a 1% significance level. Navigation time comparison suggests that the FRC is inferior (Figure 3.16 (b),(c)). Eight out of twelve cases are statistically significant (Figure 3.16 (c)). The four cases with p-values greater than 5% (Figure 3.16 (c) have navigation time ratios closer to unity than the other cases.



(a) (b)

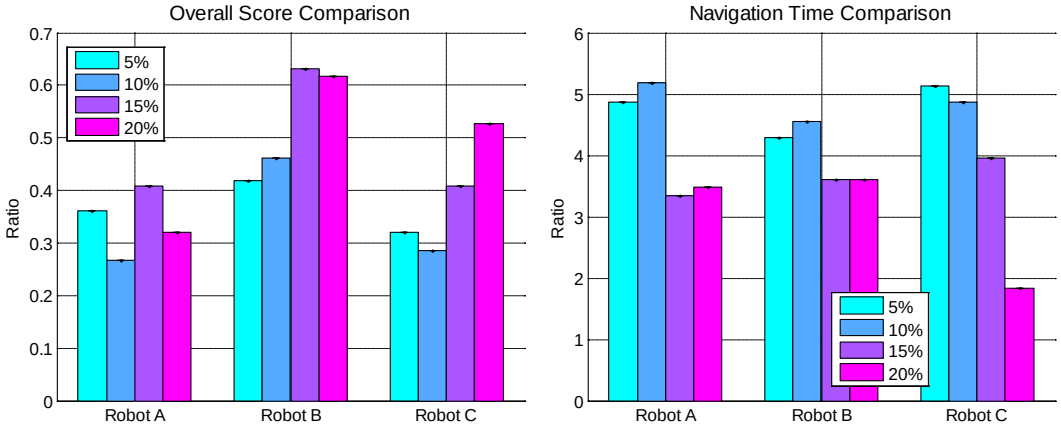
p-values (two decimal places, < 0.05 shaded blue)						
Obstacle Density	Overall Score Comparison (a)			Navigation Time Comparison (b)		
	Robot A	Robot B	Robot C	Robot A	Robot B	Robot C
5%	0.20	0.40	0.29	0.15	0.45	0.36
10%	0.07	< 0.01	0.45	0.42	< 0.01	0.38
15%	0.18	< 0.01	0.08	0.05	< 0.01	0.31
20%	< 0.01	0.45	< 0.01	0.18	0.30	0.03

(c)

Figure 3.17: Comparison of HRDU navigation and FRC.

A comparison of HRDU navigation and the FRC is shown in Figure 3.17 (a)–(c). The overall score ratio is 0.83–1.04 with 11 out of 12 cases greater than 0.9 (Figure 3.17 (a)). However, only four out of twelve cases are statistically significant (Figure 3.17 (c)). Hence, at lower obstacle densities (5%–15%) HRDU navigation and FRC have similar performance. However, at 20% density HRDU navigation is generally inferior (Figure 3.17 (c)). Navigation time ratio is 0.84–1.15 with five out of twelve cases greater than unity (Figure 3.17 (b)). Three out of twelve cases are statistically significant (Figure 3.17 (c)). Two out these three statistically significant cases

indicate the HDRU navigation has inferior performance. The inferior performance of HRDU navigation can be attributed to time delays before re-planning is triggered. Additionally, path planning in unknown environments can favour unexplored areas of the environment as re-planning frequency rises.



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)						
Obstacle Density	Overall Score Comparison (a)			Navigation Time Comparison (b)		
	Robot A	Robot B	Robot C	Robot A	Robot B	Robot C
5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

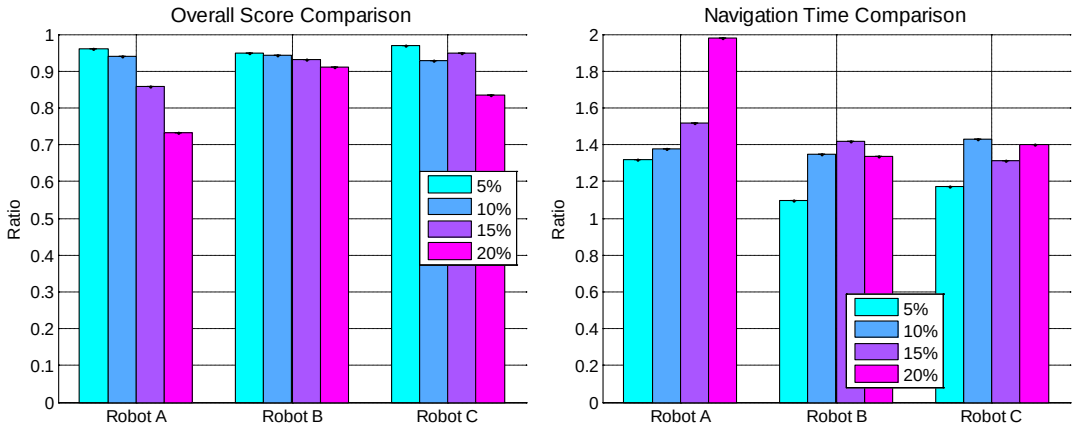
(c)

Figure 3.18: Comparison of ODW and FRC.

The original dynamic window (ODW) method has also been compared with the FRC (Figure 3.18 (a)–(c)). From the overall score comparison (Figure 3.18 (a),(c)), the FRC clearly outperforms the ODW method. The ODW method's performance is 37%–74% lower than the FRC (Figure 3.18 (a)). A similar trend appears when navigation time is compared (Figure 3.18 (b),(c)). Navigation time is increased by a factor of up to 5.2 when the ODW method is used (Figure 3.18 (b)). The navigation time comparison can be misleading for higher obstacle densities since navigation difficulty is disproportionate to the maximum allowable navigation time.

Figure 3.19 (a)–(c) compares the ODWDS method and the FRC. All comparisons of the overall scores are statistically significant with p-values less than 1% (Figure 3.19 (c)). The FRC outperforms the ODWDS method by up to 27.5% (Figure 3.19 (a)). Similar to the overall score comparison, the navigation time ratios are

statistically significant with p-values less than 1% (Figure 3.19 (c)). The navigation time of the ODWDS method is 10%–98% greater than the FRC.



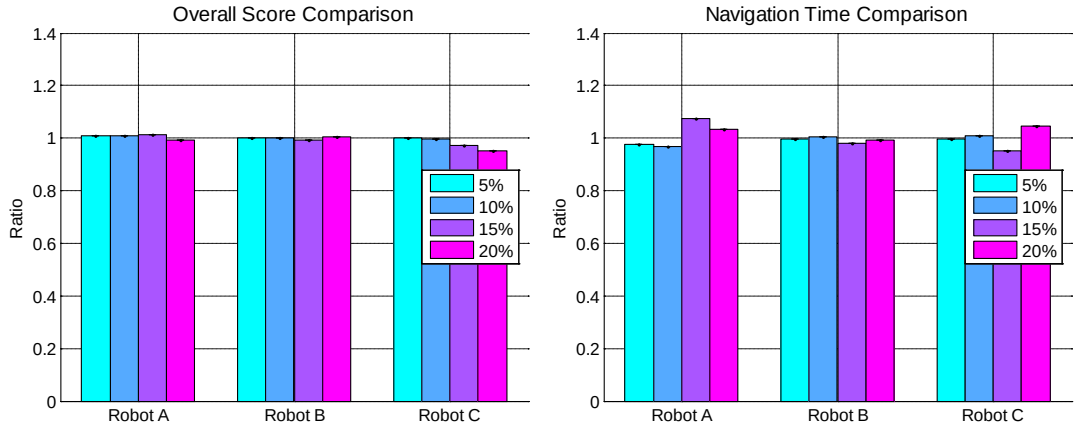
(a) (b)

p-values (two decimal places, < 0.05 shaded blue)						
Obstacle Density	Overall Score Comparison (a)			Navigation Time Comparison (b)		
	Robot A	Robot B	Robot C	Robot A	Robot B	Robot C
5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

(c)

Figure 3.19: Comparison of ODWDS and FRC.

A comparison of the performance of the RFRC and FRC is shown in Figure 3.20 (a)–(c). There is no statistically significant difference in the overall score comparison for Robot B (Figure 3.20 (c)). However, for Robot C there is a statistically significant difference in performance in 15% and 20% obstacle density environments (Figure 3.20 (c)) indicating the RFRC is inferior (Figure 3.20 (a)). Additionally, the RFRC is inferior in 20% obstacle density environments for Robot A (Figure 3.20 (a),(c)). This result is expected since the additional features in the FRC are designed to improve performance in higher obstacle density environments. Navigation time comparison yields no statistically significant difference (Figure 3.20 (c)). There is still no statistically significant difference in navigation time if the level of significance is changed to 10% (Figure 3.20 (f)).



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)						
Obstacle Density	Overall Score Comparison (a)			Navigation Time Comparison (b)		
	Robot A	Robot B	Robot C	Robot A	Robot B	Robot C
5%	0.06	0.38	0.37	0.29	0.41	0.47
10%	0.06	0.39	0.12	0.18	0.42	0.28
15%	0.11	0.15	< 0.01	0.18	0.41	0.29
20%	0.04	0.23	< 0.01	0.30	0.37	0.30

(c)

Figure 3.20: Comparison of RFRC and FRC.

3.7 Physical Robot Experiments

Figure 3.21: The tricycle robot Scratchy.

Limited experiments have been conducted on real robots. Reactive and hybrid reactive-deliberative navigation has been implemented on Itchy and Scratchy (Figure 3.21), a pair of functionally equivalent tricycle robots in the VUW mechatronics group [15]. The MATLAB control algorithms were converted to Visual C++ to ensure compatibility with the existing rudimentary control system on the robots.

Figure 3.22 shows a screen shot of the developed Visual C++ GUI. Controls for manually and autonomously driving the robots are on the left side of the GUI. The right side of the GUI has seven tabs for hardware, low-level control, robot settings, other robot state, reactive control, environment map and planning.

Figure 3.23 displays the reactive control and environment map tabs. The environment map tab displays the position of the robot in the world. Robot specific settings files are required for network communication, sensing, low-level control, reactive control and planning.

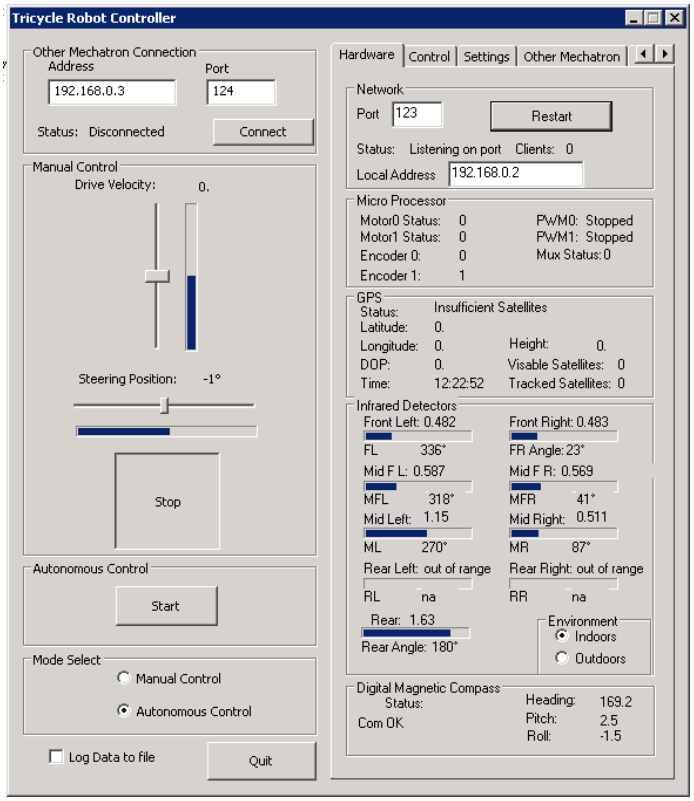


Figure 3.22: Screen shot of tricycle robot GUI.

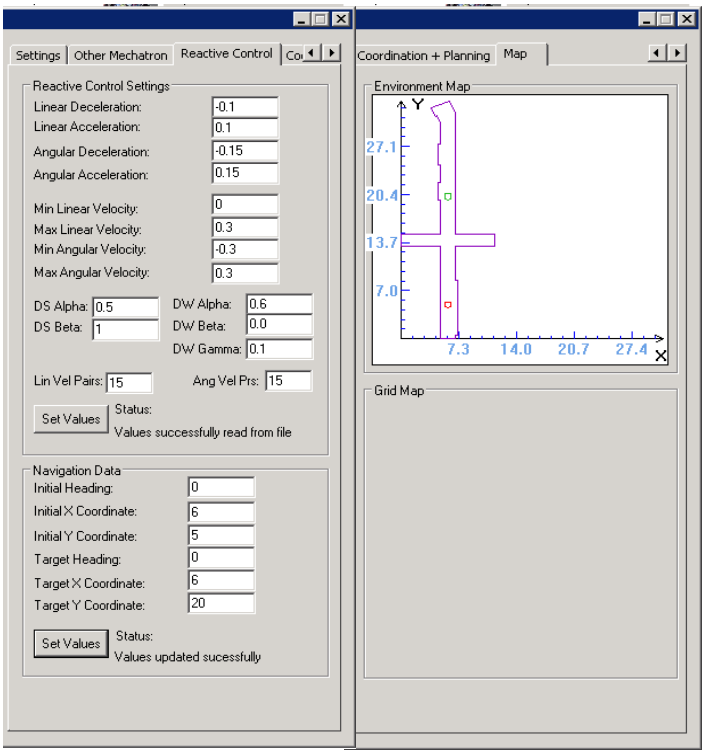


Figure 3.23: Reactive control and environment map tabs.

Table 3.4 lists the acceleration and velocity constraints determined from experimental data obtained from the pair of robots. Navigation experiments were conducted in an office corridor environment. The reactive control parameters tuned in the simulations required minimal adjustment.

Table 3.4: Acceleration and velocity constraints.

Parameter	**Numerical Value**
Linear acceleration/deceleration	$\pm$ 0.1 m/sec^2
Angular acceleration/deceleration	$\pm$ 0.15 rad/sec^2
Maximum/minimum linear velocity	$\pm$ 0.3 m/sec
Maximum/minimum angular velocity	$\pm$ 0.3 rad/sec

Figure 3.24 shows the path travelled by one of the robots when reactive control is combined with deliberative control. In this experiment, the robot was required to navigate from one end of the corridor (4 m, 6 m) and perform a right turn to reach a goal location in the adjacent corridor (14 m, 4 m). Orange lines in Figure 3.24 illustrate the corridor walls. The robot successfully reached the goal while avoiding collision with the corridor wall. However, loss of localisation accuracy, actuator noise and sensor noise resulted in the robot having to perform several reversing manoeuvres to avoid collision. This indicates that the reactive control component can override

deliberative control to avoid collision. The black dots in Figure 3.24 represent points of the corridor walls sensed by the robot as it traversed through the corridor.

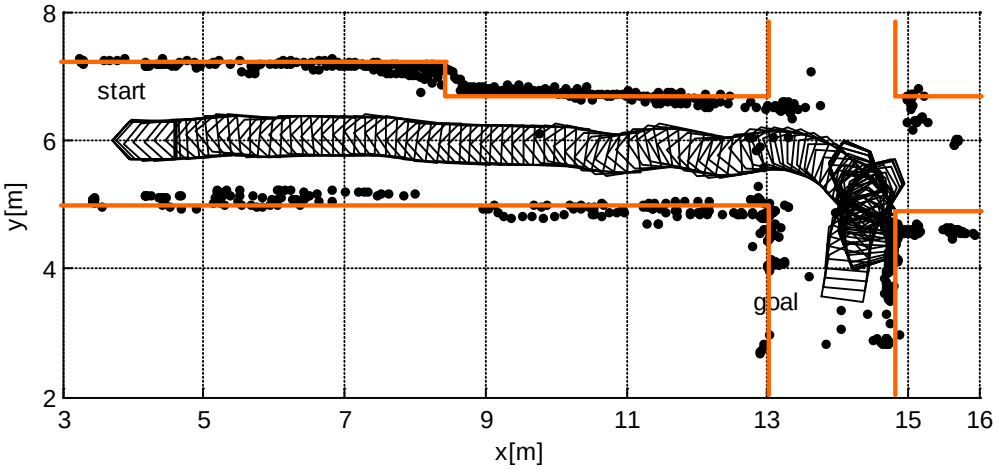


Figure 3.24: Tricycle robot hybrid reactive-deliberative navigation.

3.8 Alternative Techniques

The reactive control component of the developed hybrid-reactive deliberative navigation system is based on a combination of directional and velocity space methods. It is able to function on robots with different shapes and drive types in scattered obstacle environments. Additionally, it works with limited sensors such as infrared rangefinders and does not use complex algorithms.

Originally, the dynamic window approach (DWA) [38], a velocity space method, was designed for and tested on a synchronous drive (holonomic) robot. In contrast, the system developed in this chapter can function on the differential drive and tricycle drive robots employed in this thesis that are non-holonomic in nature. The curvature velocity method [59] is similar to the dynamic window method. Its main limitation is that it assumes a circular shape for a robot. Many of the curvature calculations rely on this assumption making it unsuitable for non-circular shapes. In contrast, the developed reactive system can function on non-circular robots.

The Vector Field Histogram (VFH) [58] directional method also assumes a circular-shaped robot and is similar to the direction sensor component of the reactive system.

It does not account for the dynamics and kinematics of a robot. Enhanced versions of the VFH method (VFH+ [58] and VFH* [63]) are more complex and also assume circular shapes. This limits their use to circular robots unlike the reactive system developed in this chapter.

In the Nearness Diagram (ND) navigation method [61], a robot's kinematics, dynamics, and shape are accounted for. This method has three stages to achieve reactive navigation: a nearness diagram stage, a motion generator stage, and a shape corrector stage. Neither of these stages employs the DWA or a polar histogram. A combination of the ND method and DWA has been developed [41] but this adds more complexity to the existing ND method. In contrast, the developed reactive system employs only two stages thus reducing complexity for limited processing robots. Experiments are conducted in unknown, non-predictable, unstructured, cluttered, dense and complex environments. However, the robots employed in the experiments have sophisticated sensing such as 2D and 3D laser rangefinders and stereo cameras. This is unlike the developed reactive system that works with limited sensors such as infrared rangefinders albeit in less complex environments.

The ego-kinodynamic space approach [62] can also function on robots with different shapes. A complex vehicle abstraction layer is employed to allow compatibility with a variety of obstacle avoidance techniques. Hence, there will still be at least two stages for reactive control using this method. Potentially, this offers no reduction in complexity when compared with the developed reactive system. The technique is tested using a potential field method [53] on a differential drive robot with a 2D laser sensor (wheelchair) in an office environment. Performance on other types of robots and in other environments is not evaluated.

Difficult environments such as dense and cluttered are considered in the obstacle-restriction method (ORM) [50]. This reactive control method also employs two stages to take into account a robot's kinematics, dynamics, and shape. Free space is identified and sub-goals are selected in the first stage. The second stage computes motion towards the sub-goals while avoiding collisions. Identifying sub-goals requires sophisticated sensors with long range such as laser rangefinders.

There are many navigation systems that employ hybrid deliberative-reactive control. A convergent dynamic window approach [39] combines the dynamic window approach, artificial potential functions, model predictive control and control Lyapunov functions. This nonlinear method is designed and tested to work in known environments using a circular robot. On the other hand, the hybrid navigation system presented in this chapter also functions in unknown environments and performs on heterogeneous robots. Using non-linear control also adds computational complexity unlike the linear approach taken in this chapter.

The global dynamic window approach [47] has been tested on a circular holonomic robot equipped with a laser rangefinder. An NF1 path planner is combined with the DWA to allow the robot to navigate in unknown environments. However, the method relies on periodic path planning (operates at 15 Hz).

A hybrid of the NF1 path planner, DWA, and elastic band method [42] has been developed to produce smooth and efficient obstacle avoidance for a differential drive tour guide robot with laser scanners. The robot is represented as a circular shape in the control algorithms and the method also relies on periodic path planning for successful navigation. Hence, a computationally powerful robot (Power PC G3) is utilised for processing. Lookup tables are also utilised by the DWA which consumes additional memory.

The A* path planner and DWA have been combined for navigation on a circular differential drive robot with a laser sensor [44]. This technique also relies on periodic path planning. Path planning is performed at every control cycle for navigation in unknown environments. Planning time has been improved by using the D* algorithm [46] with the DWA [45].

In contrast to [42, 44, 45, 47] the hybrid navigation system developed in this chapter does not rely on periodic path planning or lookup tables and represents non-circular shapes. Periodic path planning can be problematic if a limited memory robot cannot store an entire map in its local memory (chapter 4). This makes the developed navigation system more suitable for heterogeneous limited capability robots.

A reduced dynamic window approach for polygonal robots [48] has been tested on differential drive robots equipped with laser rangefinders. In this method, the NF1 planner in the planning stage provides a linear velocity for the model stage (a reduced dynamic window that selects an angular velocity). Hence, this method requires a model of the environment or requires replanning whenever an obstacle is discovered in unknown environments. In contrast, the navigation system presented in this chapter can avoid discovered obstacles without the need to replan and functions with limited range sensors.

Another dynamic window based navigation system [49] has been tested on non-circular, differential drive, and tricycle drive robots equipped with laser rangefinders. It uses lookup tables to store precalculated curvature related information. As mentioned previously, this can be memory expensive and is not utilised in the navigation system developed in this chapter. A wave front expansion algorithm generates intermediate way points for global navigation. This navigation system has only been tested in known environments. In contrast, the system presented in this chapter also functions in unknown environments.

Unlike [42, 44, 47-49], the navigation system developed in this chapter utilises a polar histogram with the DWA to allow robots with limited range sensors such as infrared rangefinders to achieve adequate goal directedness and obstacle avoidance.

Lee-Johnson's emotion based navigation system [52] is similar to the technique presented in this chapter. However, it is tailored for indoor navigation and is more complex utilising a computationally powerful circular shaped robot. The deliberative control component employs stimuli, moods, emotions, and several occupancy grid maps. In the reactive control component non-linear product objective functions are utilised rather than linear weighted sum objective functions. Complexity is further increased by utilising additional sub-components for directional and velocity control. In contrast, the hybrid navigation system presented in this chapter relies on a single occupancy grid map and has less complexity in the reactive control component.

3.9 Summary

A hierarchical hybrid navigation system that can be employed on heterogeneous mobile robots with limited sensing (such as infrared rangefinders) has been developed and evaluated. It can be utilised on robots with limited memory, provided there is sufficient memory to store a global map locally on the limited memory robot. The presented navigation system is generic and can be employed when heterogeneity arises due to robot shape and drive type.

It comprises both deliberative and reactive components for operation in known and unknown environments. The deliberative path planning component employs the A* algorithm to search an occupancy grid map. A two stage polar histogram and modified dynamic window method is utilized for the reactive system. This combines the benefits of directional methods and velocity space techniques for reactive navigation to produce a system that does not require periodic path planning.

Simulation experiments demonstrate the navigation system's effectiveness in known and unknown environments. The developed final reactive system (FRS) produces favourable results on the tested robot – obstacle density combinations. When the FRS is coupled with the deliberative system, navigation performance in known environments is improved by up to 14%. For navigation in unknown environments, the deliberative component does not affect performance significantly when compared with reactive navigation. This is a desirable result as it removes the dependence on periodic path planning for hybrid navigation in unexplored obstructed environments. Physical robot experiments using a tricycle drive robot demonstrate that the navigation system can function in the real world.

A drawback of the system is that it utilises many empirically tuned parameters to achieve a reduced dependence on periodic path planning. The empirical tuning is also performed in several steps. However, a fixed set of empirically tuned parameters has been functional across the various robot and environment configurations in the simulation experiments. A method to automatically tune the various parameters could be explored as future work.

Employing the developed navigation system on limited memory robots requires an alternative path planning technique if the global map cannot be entirely stored locally. Chapter 4 presents a novel path planning strategy for limited memory robots that exploits the benefits of hierarchical heterogeneous multi-robot systems.

4 Memory Constrained Path Planning

4.1 Overview

In applications such as multi-robot exploration and mapping [1], storing an entire global map of a large environment at the required resolution on every robot can be memory and cost expensive when there are many robots. In a hierarchical and heterogeneous multi-robot system [2] (chapter 1), computationally powerful robots (managers) can be employed for processor and memory intensive requirements. Limited computational ability robots (task executers or workers) can be employed primarily for tasks such as exploring within localised regions of a global environment.

An essential part of exploration is the need for (task executer or worker) robots to navigate beyond locally discovered space. Global path planning can be performed to enable navigation beyond local regions. Manager robots capable of storing the entire global map can apply standard search algorithms (section 2.2) to plan global paths for worker robots if they have sufficient processing resources available.

However, due to limited processing ability or memory constraints, it may not be possible for a manager robot to perform global path planning and maintain full communication with all explorer robots. Additionally, the wireless communication bandwidth and range of the explorer robots themselves are limited. This means that a manager robot may be unable to maintain communication with a large number of worker robots continuously in real-time.

Rather than rely on a single centralised path planning robot, global path planning can be forwarded to some of the worker robots. However, this can be problematic since the memory constraints of the worker robots will likely not permit them to store the entire global map.

Alternative global path planning strategies based on multi-tiered and memory efficient algorithms are discussed in section 2.5. However, these methods are not directly suitable for the hierarchical heterogeneous system comprising limited memory robots presented in this thesis. This chapter presents and evaluates an approach that is

significantly different from those reported in the literature. The technique involves initially dividing a large global map into smaller local maps based on the worker robot's memory capacity. Following this, a two-tiered A* (memory constrained) algorithm that executes entirely on the worker robot, searches the local maps for a path to the destination.

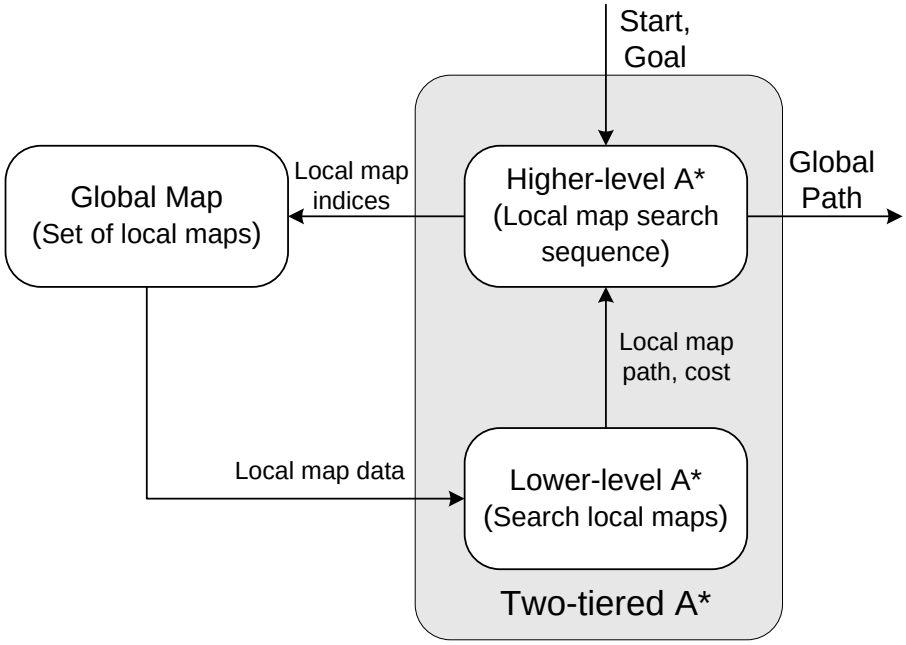


Figure 4.1: Two-tiered A* algorithm overview.

Lower-level and higher-level A* algorithms are combined to produce the two-tiered A* algorithm (Figure 4.1). The higher-level A* algorithm directs the lower-level A* algorithm to search the local maps. Pairs of neighbouring local maps are searched for a path using the lower-level A* algorithm. The path returned by the lower-level A* algorithm is stored in the robot's local memory and is used by the higher-level A* algorithm as the cost of travelling through the local maps. A complete path from the start to the goal is obtained by combining the paths through the minimum cost local maps.

The effect of varying the size and quantity of local maps for different global map sizes and obstacle densities is empirically investigated. Planning time, volume of data transmitted and path length are the performance metrics used in the experiments. While the approach is demonstrated on an occupancy grid environment

representation, it can be extended to other node based representations such as topological maps.

Simulation experiments show that memory constrained path planning can achieve superior or comparable execution times to non-memory constrained planning if the global map size is significantly larger than the local map size. In other words, employing smaller local map sizes reduces memory constrained path planning time. Furthermore, employing multiple memories of smaller local map sizes potentially improves memory constrained planning time.

4.2 Global and Local Map Representations

Assume that a grid map representation of the global environment initially exists and has been constructed by taking the size of the robots into consideration. This global map has $Rows_{GMap}$ rows and $Cols_{GMap}$ columns. Thus, the total number of nodes in the global map is:

$$Nodes_{GMap} = Rows_{GMap} \times Cols_{GMap} \tag{4.1}$$

The data stored for each node of the global map represents the probability that the region the node represents is occupied. Let the memory required to store the data for each node of the global map be Mem_{MNode} bytes. The value of Mem_{MNode} will vary depending on the precision of the data type used to store the probability information.

To facilitate path planning, the two-tiered A* algorithm requires index and adjacency references for memory constrained portions of the global map. In the context of this chapter, a memory constrained portion of the global map is defined as a local map. Assume that each limited memory robot has $Mem_{LMapMaxTot}$ bytes available for local map storage. $Mem_{LMapMaxTot}$ is used to store to a total of (Q + 2) local map memories. A fixed quantity (or number) Q of local map memories is employed to store map data retrieved from the manager robot. An additional memory equivalent to the size of two local map memories is utilised by the lower-level A* algorithm (section 4.3) for storing a merged local map. Hence, the maximum available memory per local map $Mem_{LMapMax}$ can be determined from (4.2). Each local map can store a maximum of

$Nodes_{LMapMax}$ nodes derived from the maximum available local map memory $Mem_{LMapMax}$ as shown in (4.3).

$$Mem_{LMapMax} = Mem_{LMapMaxTot} \big/ (Q+2) \tag{4.2}$$

$$Nodes_{LMapMax} = Mem_{LMapMax} \big/ Mem_{MNode} \tag{4.3}$$

Using $Nodes_{LMapMax}$ and the dimensions of the global map, the local map index and adjacency references can be determined for the global map. The algorithm pseudo code is shown in Figure 4.2. Square-shaped local maps are generated although rectangular local maps can be produced if desired.

function *partition_global_map*($Rows_{GMap}$, $Cols_{GMap}$, $Nodes_{LMapMax}$)
1 % decompose maximum number of local map nodes into local map row and column sizes
2 $RowCol_{LMap} = floor(sqrt(Nodes_{LMapMax}))$;
3 % determine the number of row and column partitions of the global map
4 $Rows_P = ceil(Rows_{GMap} \big/ RowCol_{LMap})$;
5 $Cols_P = ceil(Cols_{GMap} \big/ RowCol_{LMap})$;
6 % generate the dimensions and neighbours of each local map
7 $index_{LMap} = 1$;
8 for $Row_P = 1$ to $Rows_P$ and $Col_P = 1$ to $Cols_P$
9 $Dim_{LMap}(index_{LMap}) =$ dimensions of the local map at (Row_P, Col_P);
10 $Neighbours_{LMap}(index_{LMap}) =$ neighbours of the local map at (Row_P, Col_P);
11 $index_{LMap} = index_{LMap} + 1$;
12 end;
13 return Dim_{LMap}, $Neighbours_{LMap}$;

Figure 4.2: Local map index and adjacency references algorithm pseudo code.

The dimensions of each local map are stored as four variables in Dim_{LMap}. These variables correspond to the minimum and maximum row and column indices of the local map in the global map's reference frame. Each local map also has between two and four neighbours and the indices representing these neighbours are stored in $Neighbours_{LMap}$ as adjacency references.

In Figure 4.3, nine local grid maps have been created by the *partition_global_map* function for a 40 nodes × 40 nodes global grid map with a maximum of 200 local map nodes. Cells with a dark blue colour represent obstacles and the bold black lines mark the boundaries of the local maps. Figure 4.4 details the corresponding index, dimensions and neighbours of each local map. The actual maximum number of nodes in the square local map after generating the local map index and adjacency references is given by:

$$Nodes_{LMap} = RowCol_{LMap} \times RowCol_{LMap} \tag{4.4}$$

Thus, the actual maximum memory utilized to store the local map is given by:

$$Mem_{LMap} = Nodes_{LMap} \times Mem_{MNode} \tag{4.5}$$

The memory savings from using local maps for path planning is expected to vary depending on the size of the global and local maps as well as the structure of the environment.

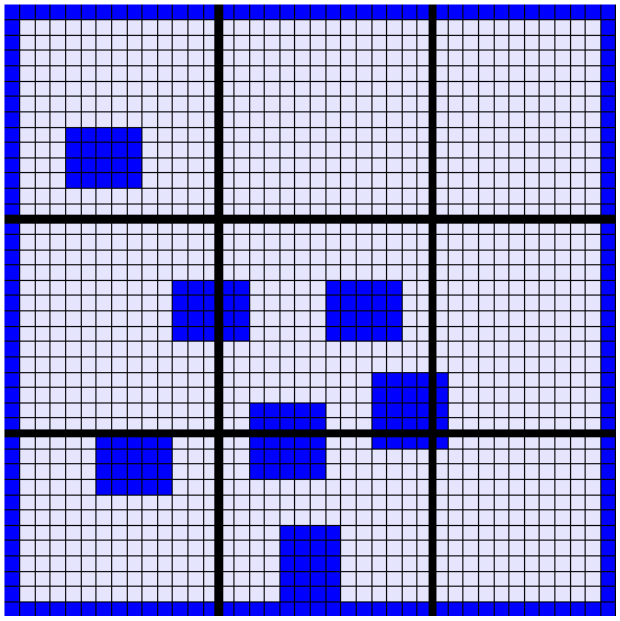


Figure 4.3: Global map divided into local maps.

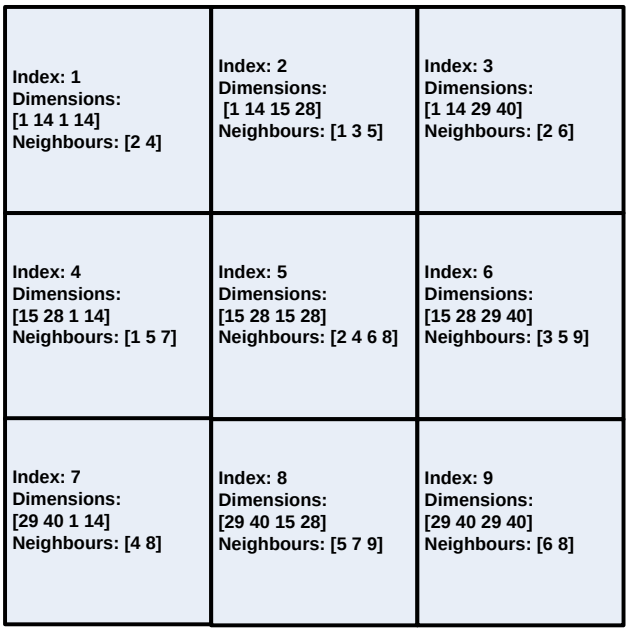


Figure 4.4: Local map indices, dimensions and neighbours.

4.3 Lower-Level A* Algorithm

The lower-level A* algorithm is a modification of the standard A* algorithm [3] that includes node occupancy probabilities in cost calculations and is similar to the path planning method presented in section 3.2.2. The algorithm searches for a path within a merged local map that is created from two adjacent local maps. One of these maps is the minimum cost local map (section 4.5), while the other adjacent map is one of four neighbouring local maps (section 4.2 and section 4.4).

A merged local map is searched to ensure that a robot can traverse between local maps. The start location, goal location and merged local map are provided by the higher-level A* as inputs to the algorithm. Similar to the standard A* algorithm, an eight-direction search is performed within the merged local map to find a path to the goal. The four adjacent and four diagonal nodes to the node under evaluation are the eight-directions that are searched.

Inspired by frontier-based exploration [4], the $g(x)$ and $h(x)$ costs of the A* algorithm have been modified to account for a varying degree of occupancy probability p_i ranging from 0.05 to 0.95. Nodes whose occupancy probabilities exceed a threshold P_T are eliminated from the cost calculation. The cost of all other nodes is linearly

dependent on a cost multiplier c_m. The reference index of the local map adjacent to the minimum cost map, $index_{LMap}$, corresponds to node *x'*. Generally, a path is previously obtained to an exit point (section 4.5) in the minimum cost local map. Hence, the adjacent node *x'* tends to dominate the search cost within the merged local map. Thus the $g(x)$ and $h(x)$ costs are expressed as $g(x,x')$ and $h(x,x')$ respectively for the merged local map (or neighbour local map).

$$g(x,x') = \begin{cases} g(x_{par},x') + d_n(x,x_{par},x').c_m(x,x') & \text{if } p_i(x,x') < P_T \\ \infty & \text{otherwise} \end{cases} \tag{4.6}$$

$$h(x,x') = \begin{cases} d_n(x,x_{goal},x').c_m(x,x') & \text{if } p_i(x,x') < P_T \\ \infty & \text{otherwise} \end{cases} \tag{4.7}$$

The unit interval cost multiplier $c_m(x,x')$ takes into account the occupancy probability $p_i(x,x')$ of node *x* inside local map *x'*. It also includes a mean occupancy probability cost $c_{prc}(x,x')$ representing a robot clearance of *r* nodes around node *x* within local map *x'*. Weights w_{pi} and w_{prc} control the balance between the two inputs and are set to 0.5 for equal preference.

$$c_m(x,x') = 1 + w_{pi}\,p_i(x,x') + w_{prc}c_{prc}(x,x') \tag{4.8}$$

$$c_{prc}(x,x') = \frac{1}{r}\sum_{i}^{r} p_i(x_i,x') \tag{4.9}$$

The overall cost $f(x,x')$ is the sum of the $g(x,x')$ and $h(x,x')$ costs.

4.4 Higher-Level A* Algorithm

The higher-level A* algorithm is based on the standard A* algorithm [3]. It determines the order in which the local maps are searched. Each local map's reference index $index_{LMap}$ (section 4.2) represents its node *x'*. An initial start location and a final goal location, Dim_{LMap} and $Neighbours_{LMap}$ are input to the higher-level A* algorithm. The initial start and final goal locations are converted to their equivalent initial and final local map nodes. Local map data are retrieved from the high memory capacity task manager robots and stored in a fixed number of local map memories during the path planning process.

The higher-level A* algorithm performs a four-direction search where each of the neighbouring local maps around the minimum cost local map is merged with the minimum cost local map to produce a merged local map. The merged local map, the end point of the path through the minimum cost local map (start location), and a desired exit point from the merged local map (goal location) is passed into the lower-level A* algorithm. The path returned by the lower-level A* algorithm, $Path_{LLA^*}(x')$, is stored with reference to the corresponding neighbour of the minimum cost local map. Additionally, the total cost of travel through the neighbour local map $f(x,x')$ is retained with the path.

Two cost methods ($f_1'(x')$ and $f_2'(x')$) have been employed to search the local maps for a global path. The two methods arise since cost of travelling through the neighbour of the minimum cost local map can be evaluated in two ways, $g_1'(x')$ and $g_2'(x')$. In cost method 1 ($g_1'(x')$), the length of $Path_{LLA^*}(x')$, $length(Path_{LLA^*}(x'))$, determines the cost of travelling through the neighbouring local map. Alternatively, cost method 2 ($g_2'(x')$) retains the total cost of travel through the neighbour local map $f(x,x')$ as the cost of travel. The planning times, volume of data transmitted and path lengths of the two cost methods can differ and are compared in section 4.6.

The overall cost of travelling across the neighbour local map ($g_1'(x')$ or $g_2'(x')$) depends on the cost of travel within the neighbour local map ($length(Path_{LLA^*}(x'))$ or $f(x,x')$) and the lowest cost path from the starting local map to the parent local map ($g_1'(x'_{par})$ or $g_2'(x'_{par})$) (4.10),(4.11). The cost of travelling to the final goal from each local map, $h'(x')$ or $d_n(x',x'_{goal})$, is estimated as the Euclidean distance between the last node of the path through the local map and the final goal (4.12). Total costs $f_1'(x')$ and $f_2'(x')$ are shown in (4.13),(4.14). Hence, (4.10), (4.12) and (4.13) are employed in cost method 1, while (4.11), (4.12) and (4.14) are utilised in cost method 2.

$$g_1'(x') = g_1'(x'_{par}) + length(Path_{LLA^*}(x')) \tag{4.10}$$

$$g_2'(x') = g_2'(x'_{par}) + f(x,x') \tag{4.11}$$

$$h'(x') = d_n(x', x'_{goal}) \tag{4.12}$$

$$f_1'(x') = g_1'(x') + h'(x') \tag{4.13}$$

$$f_2'(x') = g_2'(x') + h'(x') \tag{4.14}$$

4.5 Two-Tiered A* Algorithm

Combining the lower-level and higher-level A* algorithm's yields the two-tiered A* algorithm. A flowchart of the main steps of the two-tiered A* algorithm is presented in Figure 4.5. Figure 4.6 is a more detailed description of the planning algorithm. The complete path is reconstructed in a similar manner to that of the standard A* algorithm. The main difference is that paths through local maps are combined to form a complete path from the initial start to the final goal instead of combining nodes to produce a path from the start to the goal. There is a small amount of overlap between the paths during reconstruction that is easily removed by comparing the start and end of adjacent partial paths.

To determine a set of exit points for each local map, the perimeter of the local map is scanned to identify free space clusters. If a free space cluster is larger than the robot's diameter (or length or width), then three safe exit points are determined for the cluster: one at either end of the cluster and one at the centre of the cluster. The exit points at either extreme of the cluster have a clearance equivalent to the robot's radius. For example, in Figure 4.7 each node is equivalent to the robot's radius. Hence, a minimum of three unoccupied cells (nodes) is required to produce a free space cluster. The red dots represent exit points within each free space cluster. It is possible for all three exit points to correspond to the same point (single red dot in the lower side free space cluster of Figure 4.7). In such a case, only the single unique exit point is retained for evaluation. Similarly, the three exit points can be mapped to two unique exit points (two red dots in upper left free space cluster of Figure 4.7) and, only two exit points are retained for evaluation.

Each exit point's utility is determined as a weighted sum of its distance to the final goal, $Dist_{EP-Goal}$, and its distance from last node of the path through the parent local map, $Dist_{EP-Par}$ (4.15). The weights k_3 and k_4 are set to 0.5 for equal preference. The exit points are then ranked in ascending order of utility. These ranked exit points are

converted to the merged local map's reference frame to be used by the lower-level A* algorithm.

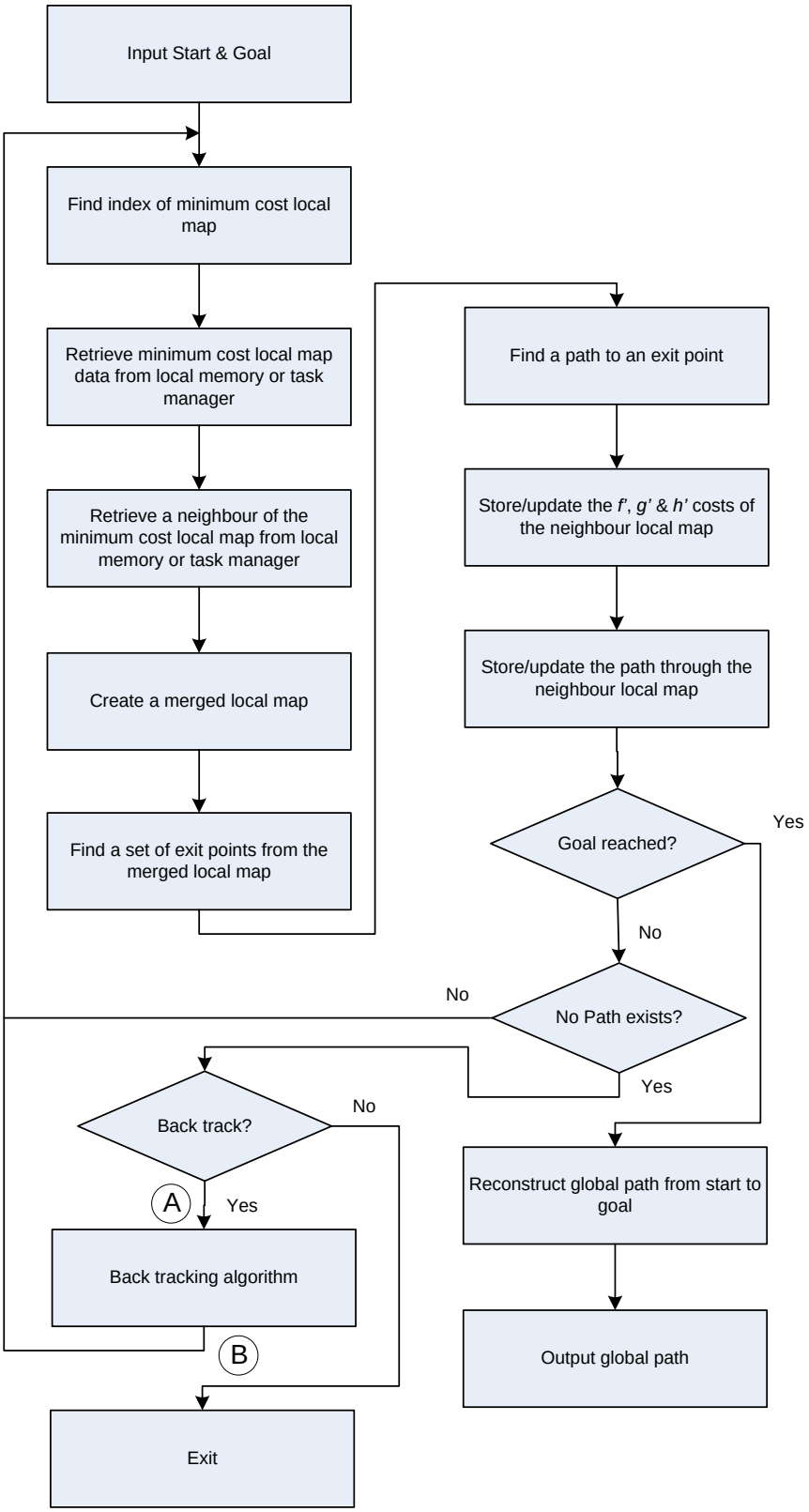


Figure 4.5: Flowchart of the two-tiered A* algorithm.

```
Input initial start location, final goal location, Dim_LMap ,and Neighbours_LMap .
Determine the initial and final local map references.
Set initial local map cost g'(x') to 0.
Determine the initial local map h'(x') and f'(x') costs.
Store initial local map reference in the open list.
While the open list is not empty and a path to the final goal is not found
Find the minimum cost local map reference on the open list, remove it from the open
list and add it to the closed list.
   If the minimum cost local map data does not exist locally
      If there is at least one local map memory free
         Retrieve data from the task manager robot(s).
      Else
         Remove the local map with the highest cost from local memory.
         Retrieve data from the task manager robot(s).
   For all neighbours of the minimum cost local map not on the closed list
      If the neighbour local map data does not exist locally
         If there is at least one local map memory free
            Retrieve data from the task manager robot(s).
         Else
            Remove the local map with the highest cost from local
            memory.
            Retrieve data from the task manager robot(s).
      If exit points have not been determined for the neighbour local map
         Find a set of safe exit points on the boundaries of the neighbour
         local map and rank them.
      If the neighbour local map exit points set is not empty and the boundary
         between the two local maps is safe to pass
         Create a merged local map from the minimum cost local map
         and the neighbour local map data.
         Set start location to last node of path through minimum cost local map.
         Set goal location to first exit point.
         While a path is not found and all exit points haven't been checked
            Find a path in the merged local map from the start location to
            the goal location using the lower-level A* algorithm.
            Set goal location to next unchecked exit point.
         If a path is found
            Determine/Update the g'(x'), h'(x'), and f'(x') costs of the neighbour local
            map.
            Determine/Update the parent of the neighbour local map.
            Store/Update the path as the path through the neighbour local map.
            Add the neighbour local map reference to the open list.
If a path to the final goal is found
   Reconstruct and return a complete path from the initial start to the final goal.
```

Figure 4.6: Two-tiered A* algorithm description.

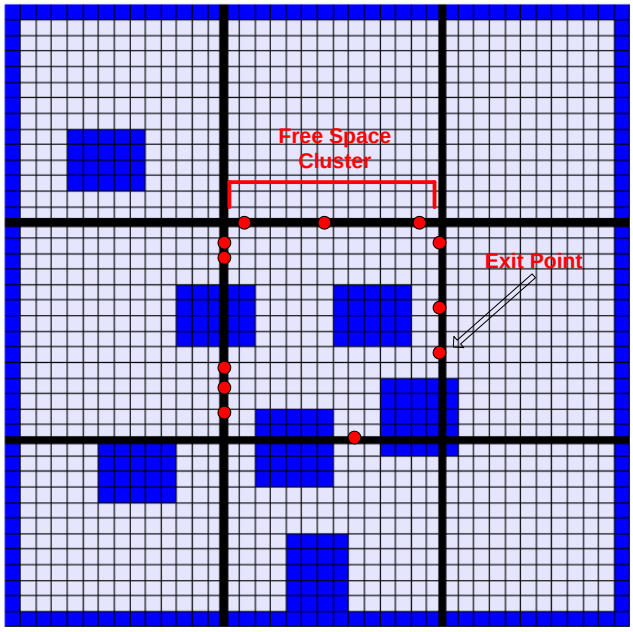


Figure 4.7: Free space clusters and exit points.

$$Utility_{EP} = k_3 \times Dist_{EP-Goal} + k_4 \times Dist_{EP-Par} \quad (4.15)$$

The boundary between the minimum cost local map and each of its neighbours needs to be checked for safe traversal. To determine the boundary (or border) row or column of neighbouring local maps, their dimensions are compared. The boundary rows or columns of the minimum cost map and its neighbour are then checked incrementally to determine if a passage greater than the robot's diameter exists. If such a passage exists then checking is terminated and the boundary is deemed safe to pass. The merged local map is created by comparing the dimensions of the neighbour local map with the minimum cost map for correct positioning (i.e. top, bottom, left or right).

In some initial tests, especially when an entire subsection of a local map is blocked due to large obstacles, the higher-level algorithm failed to find a path that existed. This is due to the nature of the A* algorithm as it does not re-evaluate local maps stored in the closed list. To deal with this situation a back tracking method has been added to the algorithm.

Figure 4.8 details the back tracking procedure. When the open list becomes empty and a complete path to the final goal has not been found (Figure 4.5), the partial path to the final goal is stored in memory. Then, all the local map references which are in the closed list but not in the partial path are removed from the closed list. Following this,

the algorithm back tracks to the parent local map of the last local map on the optimal partial path. This parent local map is removed from the closed list and placed in the open list. Finally, the last local map in the optimal partial path is removed from the parent local map's neighbour list and also from the closed list. If the algorithm back tracks to the very first local map that was searched then this means that a path to the final goal does not exist and the search is terminated.

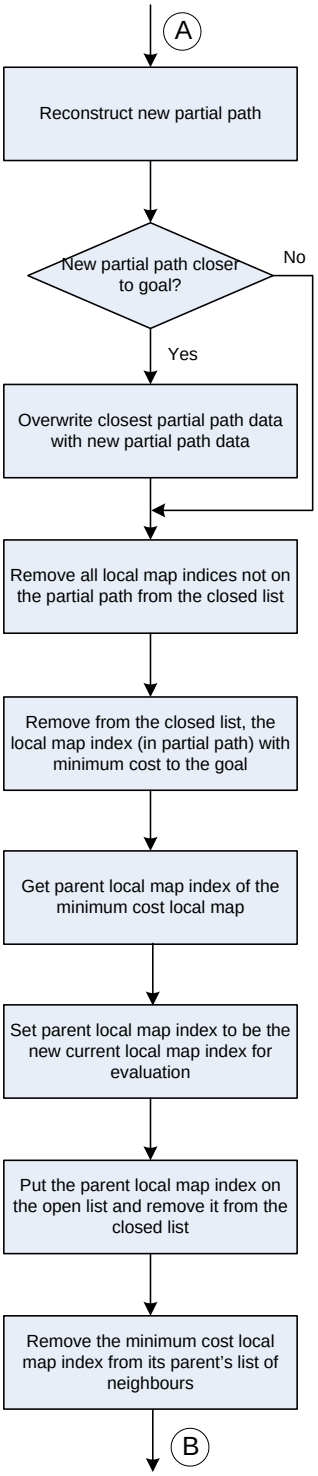


Figure 4.8: Back tracking algorithm flowchart.

4.6 Simulation Experiments

The two-tiered (memory constrained) A* path planner has been implemented and evaluated with Matlab 2007a. Randomly positioned obstacles at 5%, 10%, 15%, 20% and 25% densities have been populated in rectilinear global worlds of six different sizes. A grid map representation of these global worlds at 10 cm resolution resulted in global maps ranging from 10^4 nodes to 5×10^6 nodes. Assuming eight bytes per node, the global maps require 78 KB to 38.15 MB of memory. It is arbitrarily assumed that a 15 cm radius circular worker robot is required to travel inside each world. Eight local map sizes (64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB) have been tested for each global map size. A wireless communication link is assumed to be present between the task manager and worker robots.

Thirty worlds have been generated and tested for the 5% to 20% obstacle density global worlds. At 25% obstacle density, thirty global worlds have been evaluated for the smaller global map sizes (78 KB, 390 KB, 781 KB, 3.81 MB). Ten worlds are evaluated at 25% obstacle density for the larger global map sizes (7.63 MB and 38.15 MB) due to lengthy planning times. Experiments in each world have been repeated ten times. For each test, a path is planned from a start location (S) near the top left corner of the global map to a goal location (G) near the bottom right corner of the map. Planning time, volume of map data received and path length have been evaluated in the experiments. Three quantities of local map memories (one, five and fifteen) have been evaluated for the worker robots. Additionally, the two cost methods presented in section 4.4 have been tested.

In the results figures, each bar represents the average value and the corresponding error bar illustrates standard deviation. A paired sample t-test with two-sided p-values is used to compare the results of employing the various approaches. Comparisons are statistically significant if p-values are less than or equal to 0.05 (5% statistical significance level). Performance ratios are computed to determine superiority or inferiority by dividing the results obtained from one method by the results obtained from another method.

4.6.1 General Trends of Local Map Size and Global Map Size Variation

Figure 4.9 – Figure 4.14 illustrate the results of employing only path length for cost calculations (cost method 1) in the higher-level A* component of the memory constrained A* algorithm. The local map memory quantity was set to one for the results of Figure 4.9 – Figure 4.14. In Figure 4.9 – Figure 4.14, the last set of bars in each graph (Inf) illustrates non-memory constrained A* planning data. Note that the vertical scale of planning time changes as the global map size is varied in Figure 4.9 – Figure 4.14.

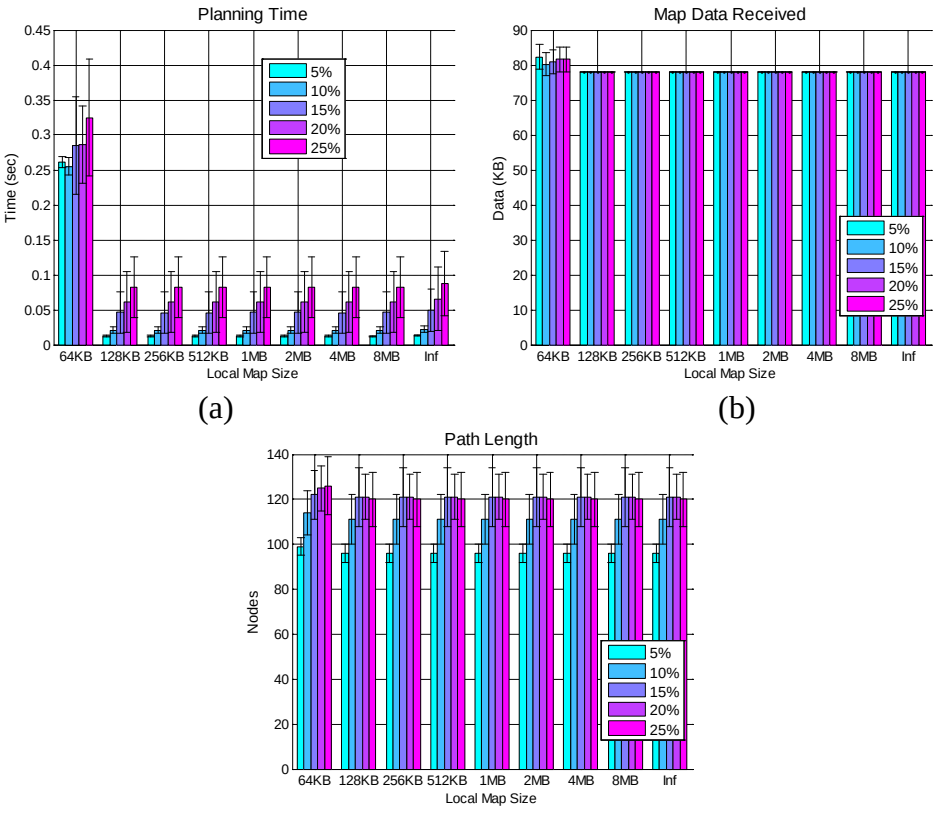
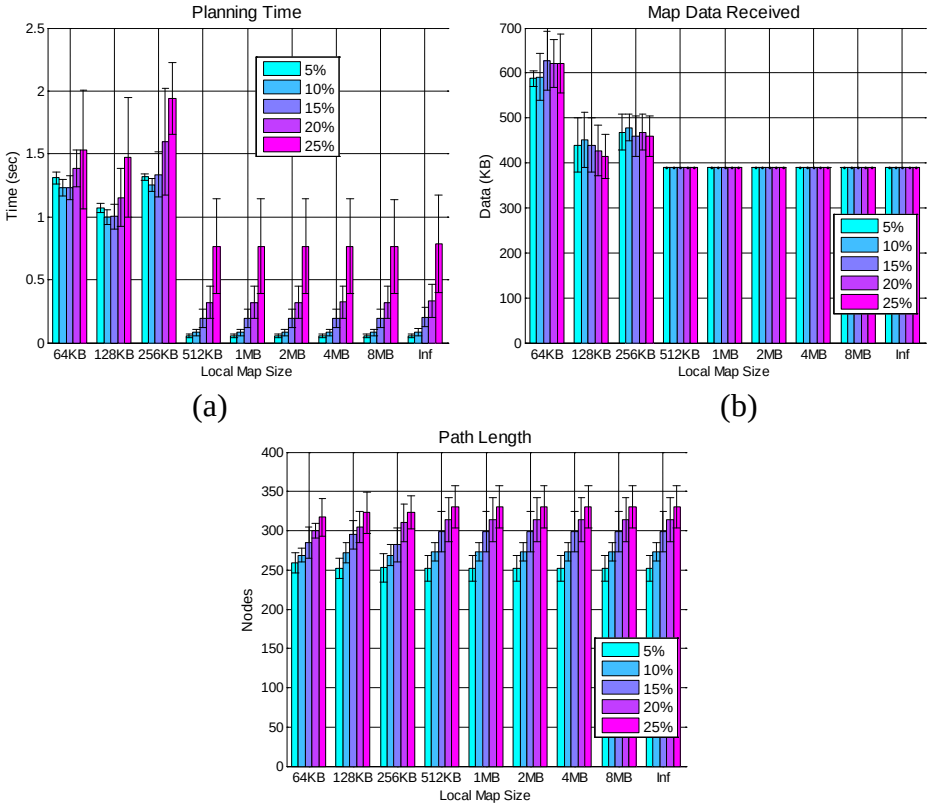


Figure 4.9: Results for planning in a 78 KB global map with 1 local map memory using cost method 1.

As expected, planning time is generally proportional to obstacle density for all global map sizes (Figure 4.9(a) – Figure 4.14(a)). Figure 4.12(a) – Figure 4.14(a) depict that this trend is particularly evident when the local map size (64 KB, 128 KB) is much smaller than the global map size (3.81 MB, 7.63 MB, 38.15 MB). This trend is also evident as the quantity of local map memories is varied to five or fifteen.

Smaller local map sizes (64 KB, 128 KB) tend to produce lower planning times than larger local map sizes if the global map is at least 3.81 MB and cannot be entirely stored in local memory (Figure 4.12(a) – Figure 4.14(a)). This suggests that using smaller local map sizes can be more efficient if the entire global map cannot be stored locally (i.e. when the memory available for local map storage is smaller than the global map size). This validates the approach.



(a) (b)

(c)

Figure 4.10: Results for planning in a 390 KB global map with 1 local map memory using cost method 1.

As the global map size is increased, smaller local map sizes (64 KB, 128 KB) receive a lower volume of map data than larger local map sizes if the global map is at least 3.81 MB and cannot be entirely stored in local memory (Figure 4.12(b) – Figure 4.14(b)). This indicates that a reduced search space may have contributed towards planning time reduction. The reduced search space may affect the obstacle clearance quality of path planning, particularly in higher obstacle density (20%, 25%) environments.

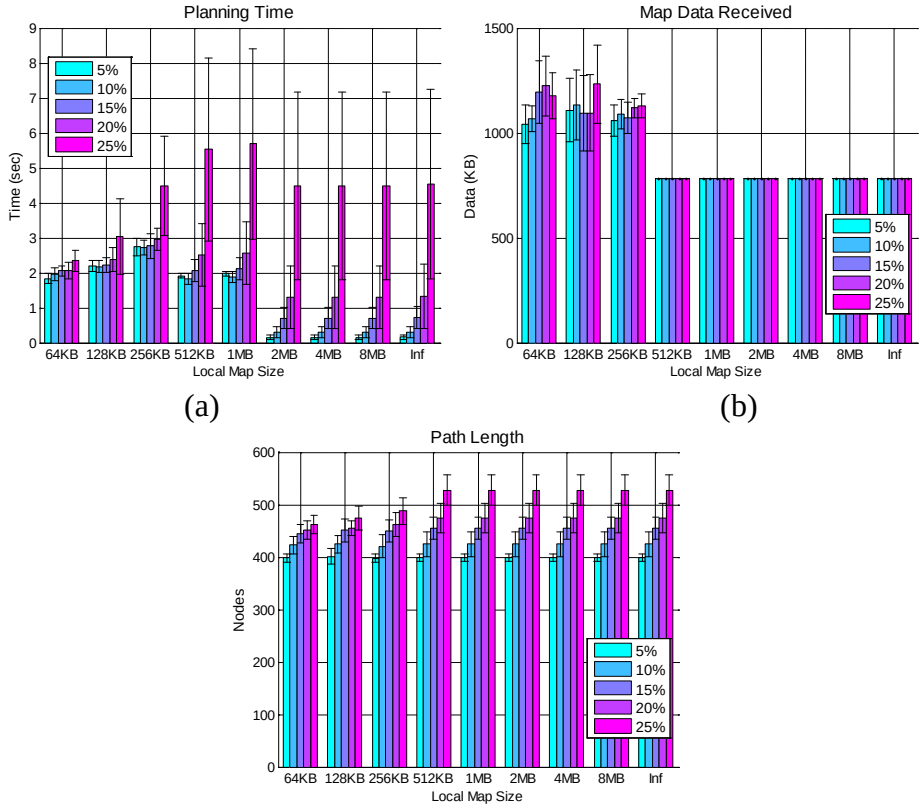


Figure 4.11: Results for planning in a 781 KB global map with 1 local map memory using cost method 1.

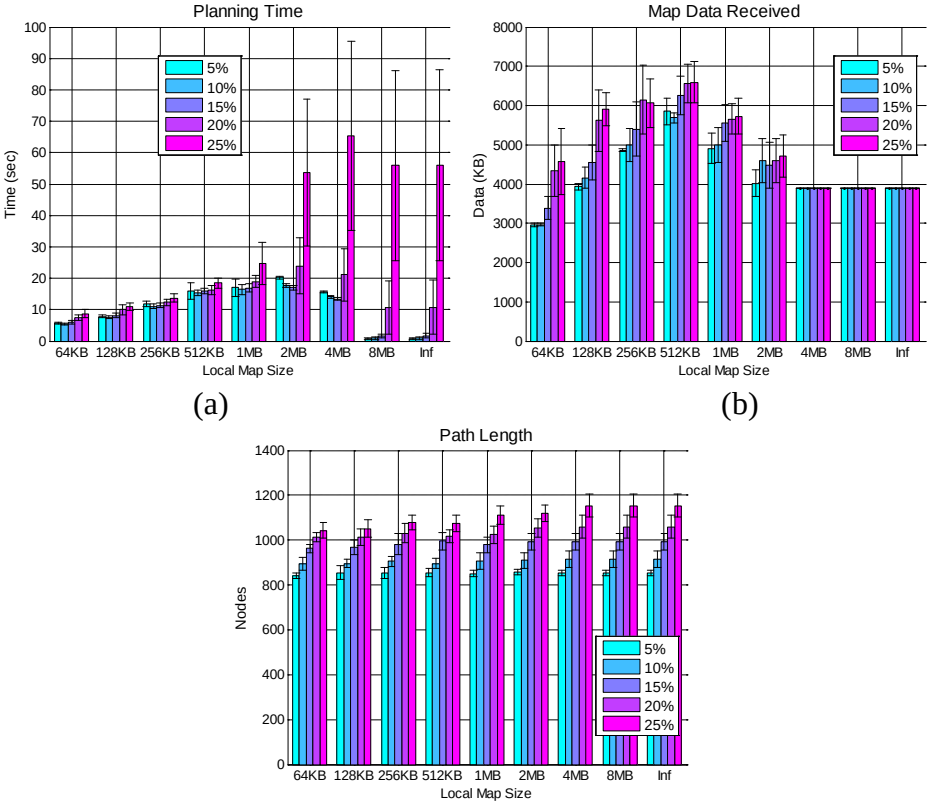
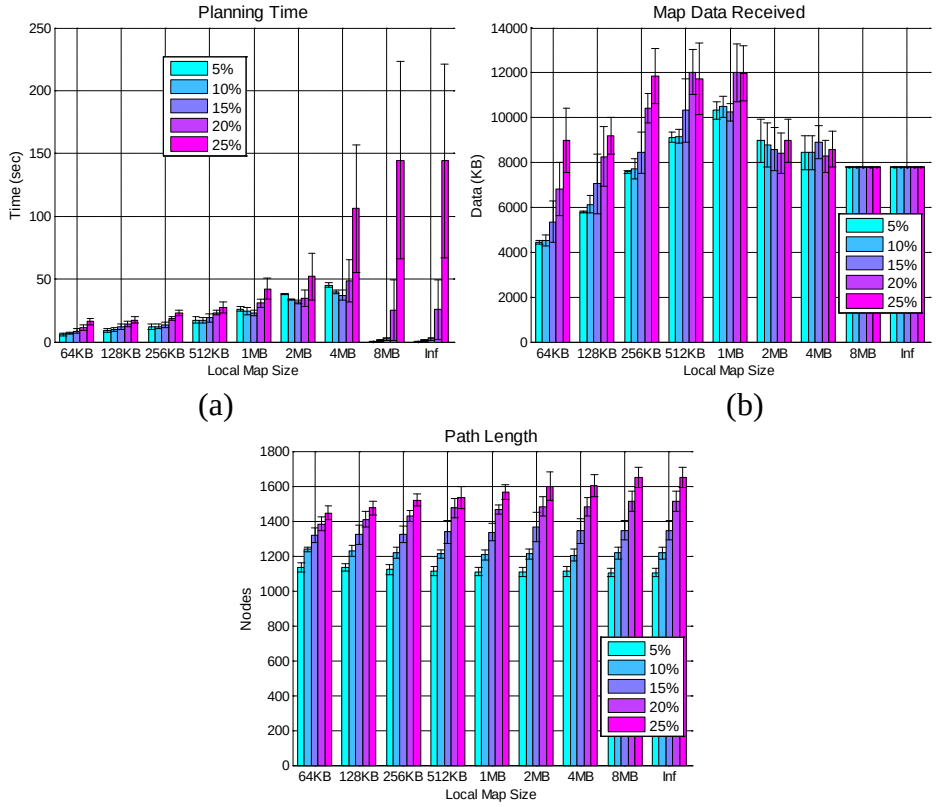


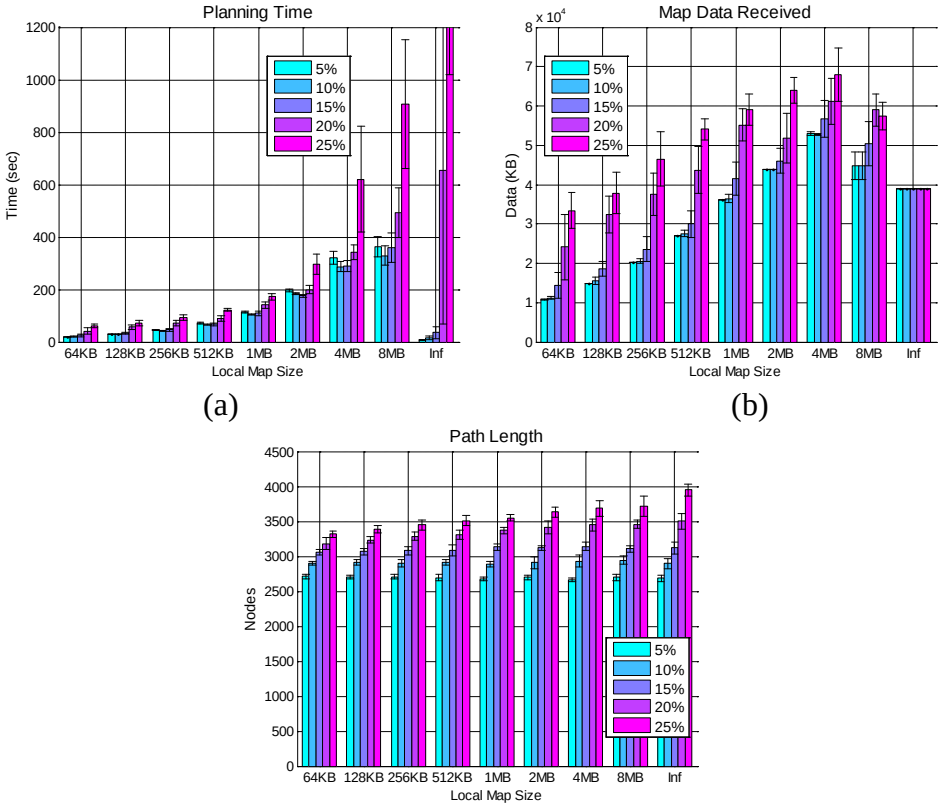
Figure 4.12: Results for planning in a 3.81 MB global map with 1 local map memory using cost method 1.



(a) (b)

(c)

Figure 4.13: Results for planning in a 7.63 MB global map with 1 local map memory using cost method 1.



(a) (b)

(c)

Figure 4.14: Results for planning in a 38.15 MB global map with 1 local map memory using cost method 1.

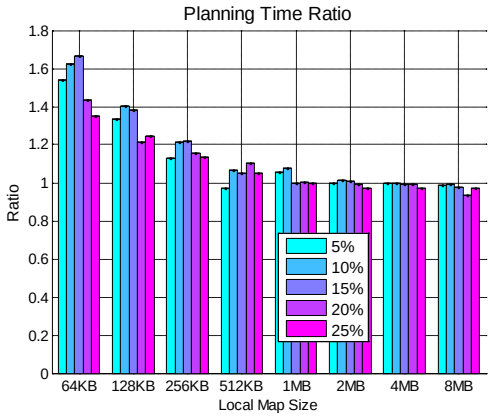
While Figure 4.11(c) – Figure 4.14(c) show shorter memory constrained paths at higher obstacle densities when smaller local map sizes are employed, this does not necessarily indicate superiority. On the other hand, a reduction in the volume of map data received can minimise communication latency (which is excluded in the planning time graphs). The trends present in the one local map memory experiments are also evident if the quantity of local map memories is changed to five or fifteen.

All of the trends discussed above for cost method 1 are also present when the total cost of travel through a local map ($f(x,x')$) is employed for cost calculations in the higher-level A* algorithm (cost method 2).

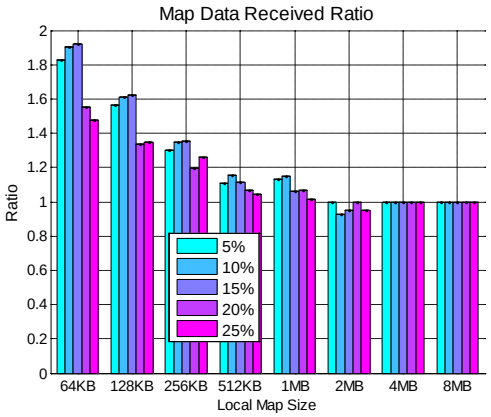
4.6.2 Comparison of Cost Method 1 and Cost Method 2

The goal of the memory constrained planning method is to be efficient when the global map size is much larger than the local map size. Hence, performance in the larger global maps is statistically compared for both cost methods (Figure 4.15 – Figure 4.17). Cost method 2 data is divided by cost method 1 data to calculate planning time, map data received and path length ratios. For all data types, a ratio less than unity indicates superiority of cost method 2. A statistically significant comparison has a p-value threshold of 0.05 (5%).

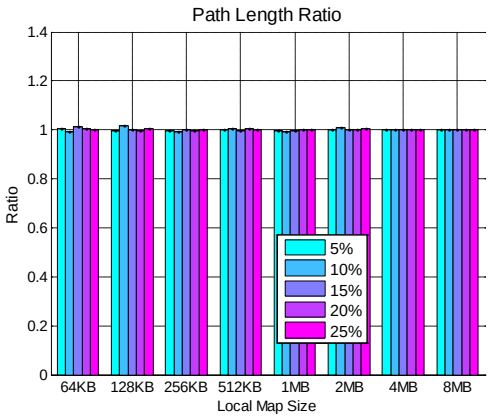
Figure 4.15(a),(d), Figure 4.16(a),(d) and Figure 4.17(a),(d) reveal that planning time increases by 16% – 65%, 25% – 160%, and 40% – 280% respectively when cost method 2 is employed for smaller local map sizes (64 KB, 128 KB, 256 KB). There is also a statistically significant increase in planning time when 512 KB and 1 MB local maps are employed in 38.15 MB global worlds (Figure 4.17(a),(d)). The volume of map data received for smaller local map sizes (64 KB, 128 KB, 256 KB, 512 KB) increases by 5% – 90%, 20% – 70% and 40% – 360% when cost method 2 is employed in 3.81 MB (Figure 4.15(b),(d)), 7.63 MB (Figure 4.16(b),(d)) and 38.15 MB (Figure 4.17(b),(d)) global worlds respectively. This suggests that the second cost method has an increased search space when smaller local map sizes are used. It also means that the second cost method has greater data communication delay costs if the global map size is larger than the local map size.



(a)



(b)

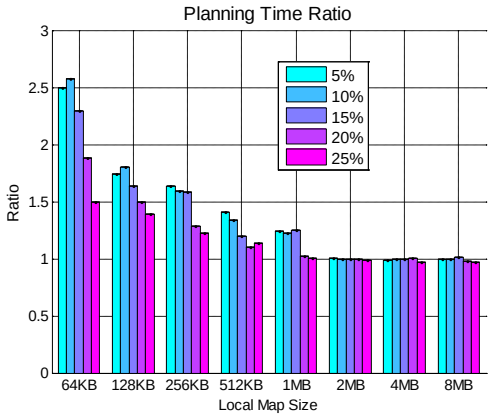


(c)

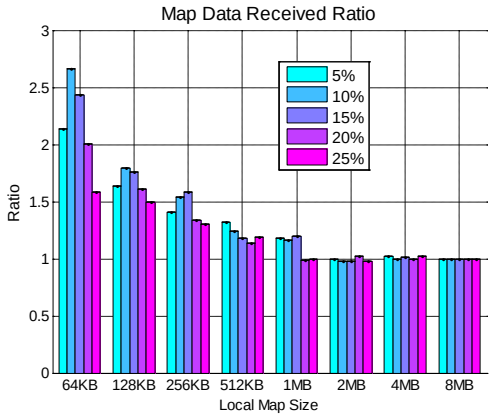
p-values (two decimal places, < 0.05 shaded blue)									
	Obstacle Density	Local Map Size							
		64 KB	128 KB	256 KB	512 KB	1 MB	2 MB	4 MB	8 MB
Planning Time (a)	5%	< 0.01	< 0.01	< 0.01	0.10	< 0.01	0.42	0.33	0.41
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.20	0.47
	15%	< 0.01	< 0.01	< 0.01	< 0.01	0.35	0.12	0.10	0.33
	20%	< 0.01	< 0.01	< 0.01	< 0.01	0.47	0.43	0.43	0.26
	25%	< 0.01	< 0.01	< 0.01	< 0.01	0.50	0.30	0.33	0.35
Map Data Received (b)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.50	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.50	< 0.01
	25%	< 0.01	< 0.01	< 0.01	< 0.01	0.04	< 0.01	< 0.01	< 0.01
Path Length (c)	5%	0.01	0.20	0.18	0.50	0.18	0.50	0.50	0.50
	10%	0.02	< 0.01	0.02	0.15	0.09	0.06	0.50	0.50
	15%	< 0.01	0.34	0.44	0.29	0.34	0.42	0.50	0.50
	20%	0.06	0.28	0.28	0.12	0.36	0.50	0.50	0.50
	25%	0.42	0.18	0.50	0.42	0.50	0.09	0.50	0.50

(d)

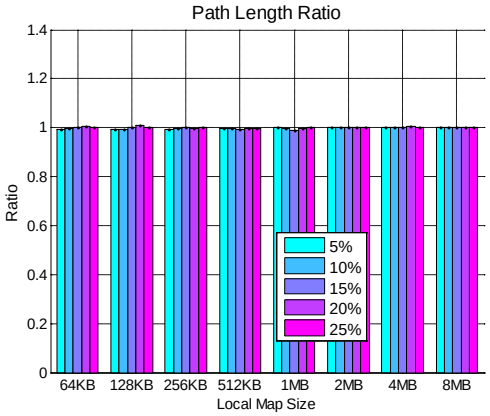
Figure 4.15: Comparison of the two cost methods in 3.81 MB worlds.



(a)



(b)

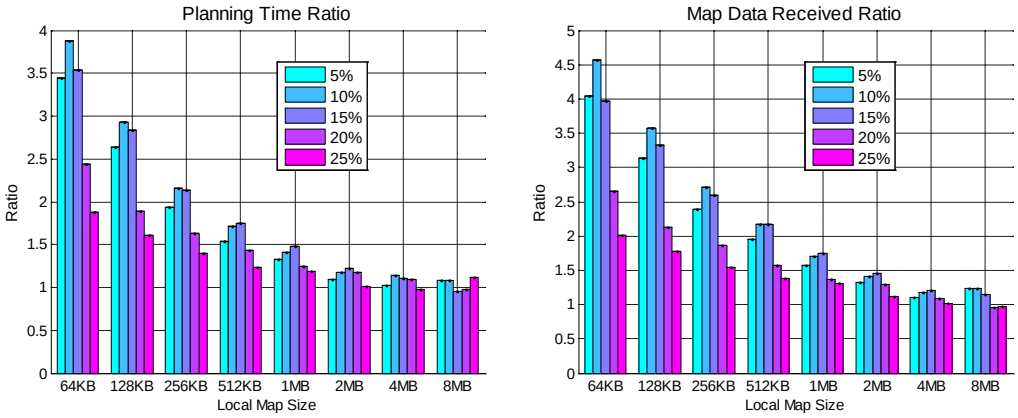


(c)

p-values (two decimal places, < 0.05 shaded blue)									
	Obstacle Density	Local Map Size							
		64 KB	128 KB	256 KB	512 KB	1 MB	2 MB	4 MB	8 MB
Planning Time (a)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.13	0.06	0.31
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.09	0.50	0.48
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.40	0.45	0.42
	20%	< 0.01	< 0.01	< 0.01	< 0.01	0.11	0.50	0.49	0.43
	25%	< 0.01	< 0.01	< 0.01	< 0.01	0.48	0.43	0.31	0.36
Map Data Received (b)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.50	0.07	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.07	0.50	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.06	0.05	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	0.10	0.06	0.50	< 0.01
	25%	< 0.01	< 0.01	< 0.01	< 0.01	0.50	0.07	0.08	< 0.01
Path Length (c)	5%	< 0.01	0.02	0.02	0.19	0.37	0.50	0.40	0.50
	10%	< 0.01	0.03	0.10	0.28	0.32	0.40	0.50	0.50
	15%	0.50	0.50	0.44	0.20	0.05	0.40	0.50	0.50
	20%	0.13	0.05	0.09	0.35	0.18	0.40	0.27	0.50
	25%	0.43	0.32	0.42	0.41	0.44	0.50	0.42	0.50

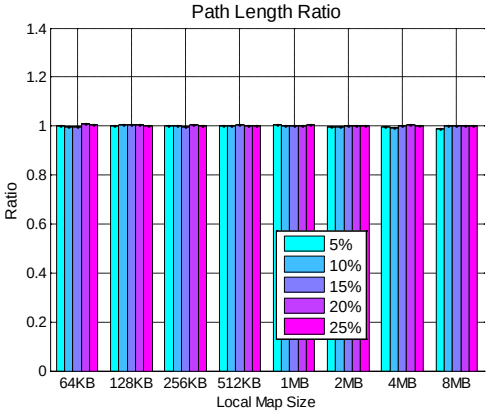
(d)

Figure 4.16: Comparison of the two cost methods in 7.63 MB worlds.



(a)

(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)									
	Obstacle Density	Local Map Size							
		64 KB	128 KB	256 KB	512 KB	1 MB	2 MB	4 MB	8 MB
Planning Time (a)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.03	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.23
	25%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.20	0.29	< 0.01
Map Data Received (b)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	25%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.18	< 0.01
Path Length (c)	5%	0.22	0.07	0.19	0.34	0.05	0.01	0.02	< 0.01
	10%	0.02	< 0.01	0.34	0.36	0.43	0.32	0.05	0.40
	15%	0.08	0.07	0.28	0.02	0.15	0.17	0.19	0.11
	20%	< 0.01	< 0.01	0.14	0.33	0.50	0.34	0.16	0.50
	25%	< 0.01	0.35	0.24	0.34	< 0.01	0.39	0.35	0.34

(d)

Figure 4.17: Comparison of the two cost methods in 38.15 MB worlds.

There is no statistically significant difference in the path lengths of the two cost methods for majority of the tested configurations (Figure 4.15(c),(d), Figure

4.16(c),(d) and Figure 4.17(c),(d)). This indicates that cost method 1 is preferred if communication delay and planning time is to be minimised.

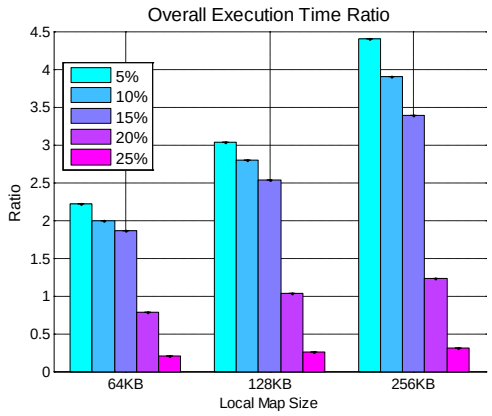
4.6.3 Comparison of Overall Execution Time in the Three Largest Global Worlds

Figure 4.18 – Figure 4.20 shows a comparison of the ratios of memory constrained and non-memory constrained overall planning times (inclusive of data communication delay) for the three largest global map sizes. Cost method 1 is employed in these experiments. Again, the larger global map sizes are selected as they demonstrate the purpose of memory constrained planning more appropriately. Data communication delay consists of a number of factors such as processing delay, queuing delay, transmission delay and propagation delay [5]. While processing delay and propagation delay are generally negligible, queuing delay can be ignored if the traffic intensity [5] is minimal.

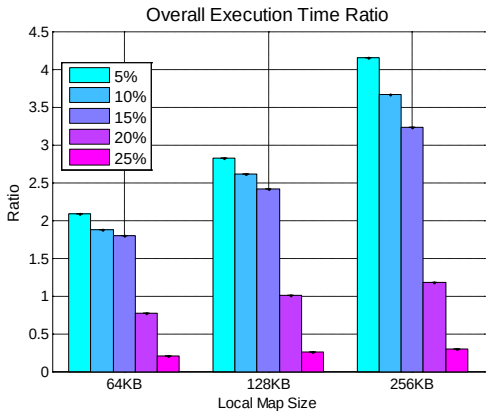
A 54 MBit/sec 802.11g wireless communication is arbitrarily assumed to exist on the high powered manager robot for map data transmission [6]. The low powered worker robots are assumed to have 802.11b devices (maximum transmission rate of 11 Mbit/sec) as these are now low cost and easily available. However, as the worker robots have limited processing capabilities and may encounter periods of inability to communicate with the managers, or face other errors, it is assumed that the effective data transmission rate is halved to approximately 5.5 MBit/sec.

Assuming there are only few mother nodes (approximately 2–3) for memory constrained planning, queuing delay is expected to be negligible if the local map size is small (64 KB to 256 KB). Moreover, Figure 4.9(a) – Figure 4.14(a) indicate that using smaller local map sizes reduces planning time if the entire global map cannot be stored locally. If larger local map sizes (512 KB to 8 MB) are employed, queuing delay will become significant thus further increasing the overall planning time. Hence, data communication delay is represented solely as transmission delay in Figure 4.18 – Figure 4.20 where only smaller local map sizes (64 KB to 256 KB) are analysed. The ratio of memory constrained and non-memory constrained overall planning time (Figure 4.18 – Figure 4.20) is calculated by dividing memory constrained data by non-memory constrained data. Superiority is denoted by a ratio

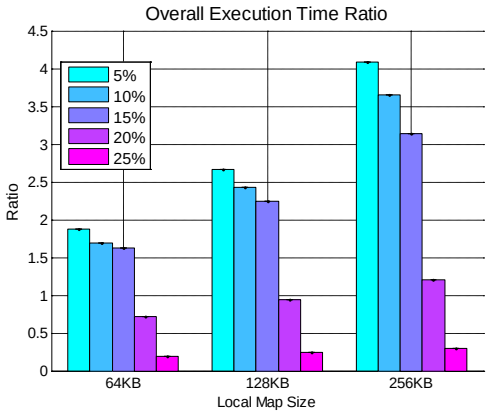
less than unity. A statistically significant comparison has a p-value less than 0.05 (5%).



(a)



(b)



(c)

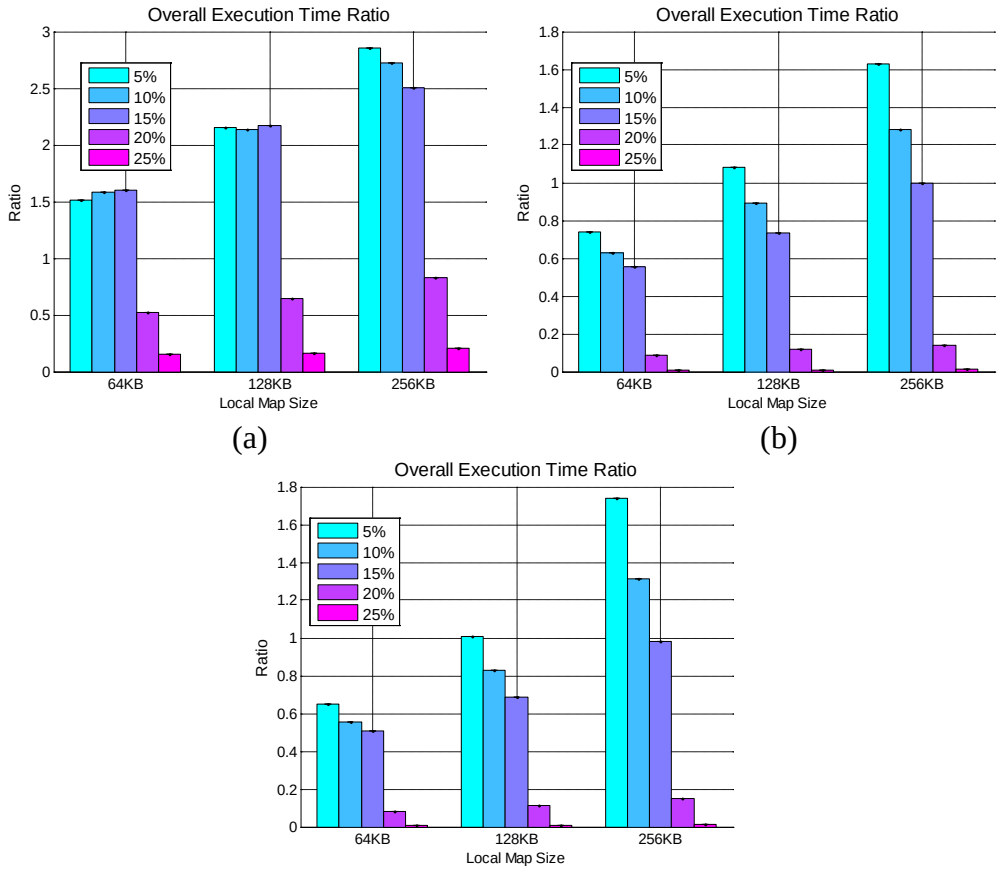
p-values (two decimal places, < 0.05 shaded blue)									
Obstacle Density	**1 Local Map Memory (a)**			**5 Local Map Memories (b)**			**15 Local Map Memories (c)**		
	64 KB	**128 KB**	**256 KB**	**64 KB**	**128 KB**	**256 KB**	**64 KB**	**128 KB**	**256 KB**
5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
20%	0.02	0.40	0.04	0.01	0.50	0.08	< 0.01	0.29	0.07
25%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

(d)

Figure 4.18: Overall execution time comparison in 3.81 MB global worlds employing one (a), five (b) and fifteen (c) local map memories.

In 3.81 MB worlds (Figure 4.18), memory constrained path planning yields inferior performance at lower obstacle densities (≤ 15%). At higher obstacle densities (≥ 20%), memory constrained planning is generally superior or comparable to non-memory constrained planning (Figure 4.18). The use of exit points in memory constrained planning is likely to reduce its search space at higher obstacle densities (≥ 20%) resulting in superior or comparable performance. On the other hand,

inferiority at lower obstacle densities (≤ 15%) can be attributed to a larger overall search space in comparison to non-memory constrained planning. Similar results are obtained in 7.63 MB worlds (Figure 4.19). The key difference is that the ratio of memory constrained to non-memory constrained data is lower.



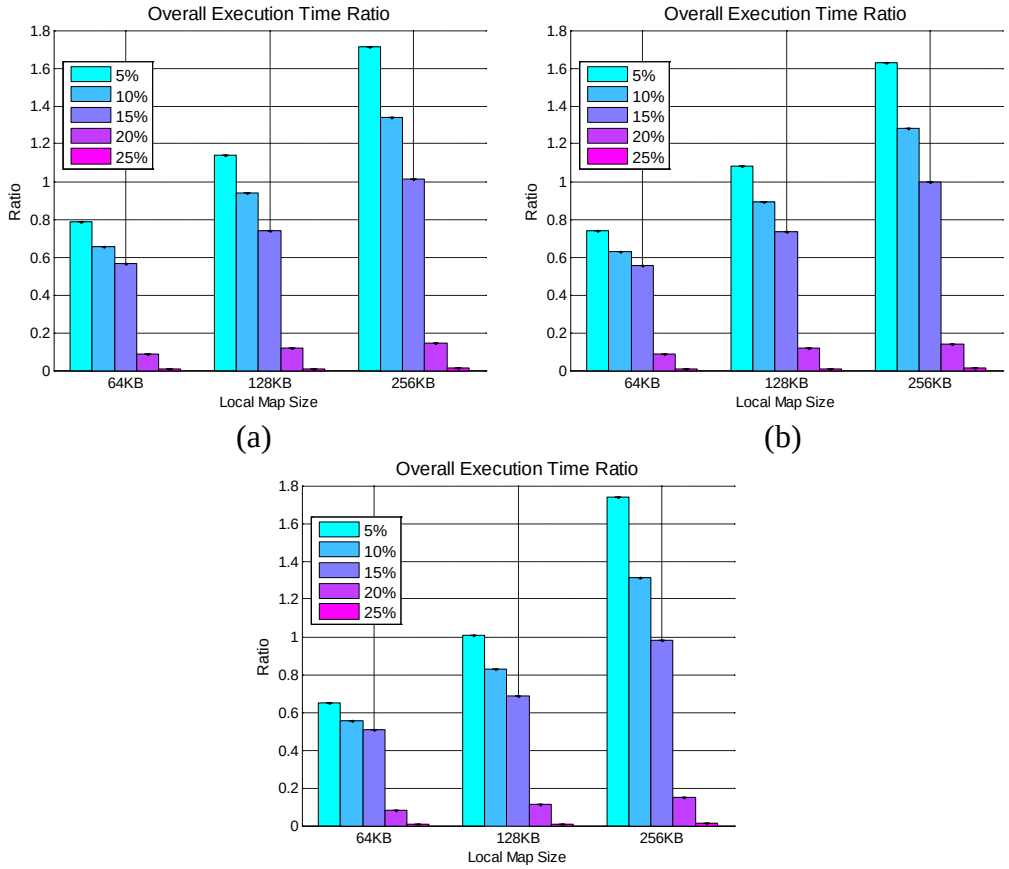
p-values (two decimal places, < 0.05 shaded blue)									
Obstacle Density	1 Local Map Memory (a)			5 Local Map Memories (b)			15 Local Map Memories (c)		
	64 KB	128 KB	256 KB	64 KB	128 KB	256 KB	64 KB	128 KB	256 KB
5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.05	< 0.01	< 0.01
10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	< 0.01	< 0.01
15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
20%	< 0.01	< 0.01	0.06	< 0.01	< 0.01	0.03	< 0.01	< 0.01	< 0.01
25%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

(d)

Figure 4.19: Overall execution time comparison in 7.63 MB global worlds employing one (a), five (b) and fifteen (c) local map memories.

The memory constrained algorithm achieves superior or comparable execution times to the non-memory constrained method in 38.15 MB worlds for 64 KB and 128 KB local map sizes (Figure 4.20). For a local map size of 256 KB, the memory constrained algorithm is inferior at lower obstacle densities (≤ 15%). This inferiority

is likely to be due to the overall search space for memory constrained planning being greater than the non-memory constrained search space. On the other hand, the use of exit points in memory constrained planning is likely to reduce its search space at higher obstacle densities (≥ 20%) resulting in superior performance.



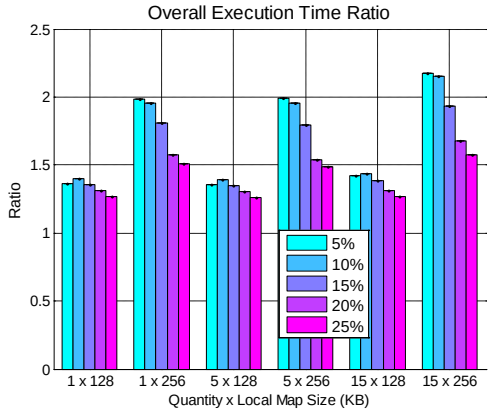
p-values (two decimal places, < 0.05 shaded blue)									
Obstacle Density	1 Local Map Memory (a)			5 Local Map Memories (b)			15 Local Map Memories (c)		
	64 KB	128 KB	256 KB	64 KB	128 KB	256 KB	64 KB	128 KB	256 KB
5%	0.02	0.13	< 0.01	< 0.01	0.25	< 0.01	< 0.01	0.47	< 0.01
10%	< 0.01	0.28	< 0.01	< 0.01	0.15	0.01	< 0.01	0.04	0.01
15%	< 0.01	< 0.01	0.45	< 0.01	< 0.01	0.50	< 0.01	< 0.01	0.43
20%	0.02	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
25%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

(d)

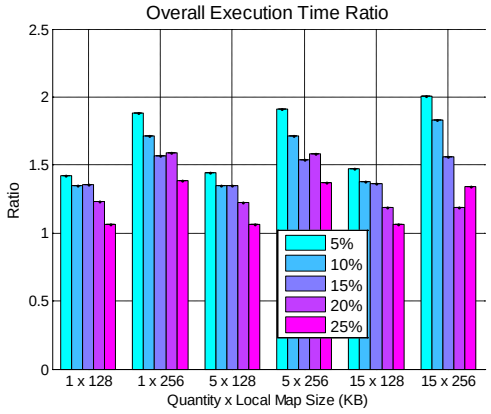
Figure 4.20: Overall execution time comparison in 38.15 MB global worlds employing one (a), five (b) and fifteen (c) local map memories.

From the analysis of Figure 4.18 – Figure 4.20, memory constrained path planning is capable of achieving superior or comparable execution times to non-memory constrained planning if the global map size much larger than the local map size.

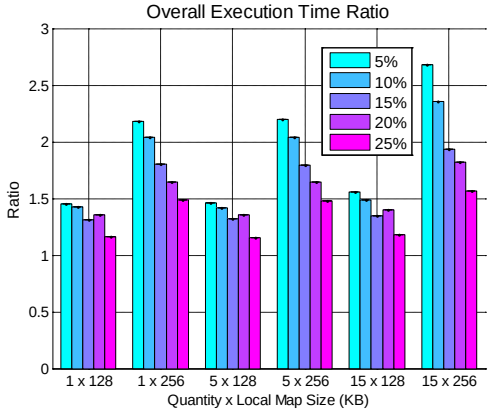
4.6.4 Comparison of Overall Execution Time Using the Three Smallest Local Map Sizes



(a)



(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)							
	Obstacle Density	Quantity × Local Map Size (KB)					
		1×128	1×256	5×128	5×256	15×128	15×256
3.81 MB Global Worlds (a)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	0.01	< 0.01
	25%	0.02	< 0.01	0.02	< 0.01	0.02	< 0.01
7.63 MB Global Worlds (b)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	0.03	< 0.01	0.03	< 0.01	0.07	0.07
	25%	0.28	< 0.01	0.29	< 0.01	0.30	< 0.01
38.15 MB Global Worlds (c)	5%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	10%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	15%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	20%	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	25%	0.09	< 0.01	0.10	< 0.01	0.07	< 0.01

(d)

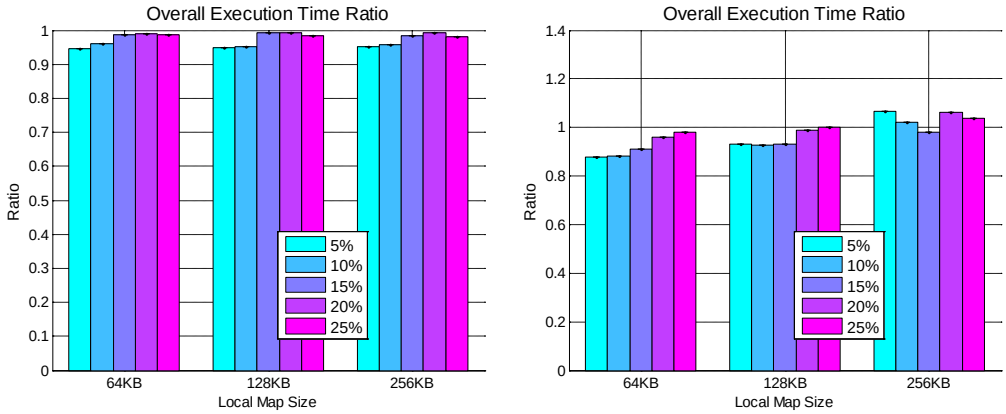
Figure 4.21: Overall execution time of memory constrained planning relative to equivalent quantities of 64 KB local maps in 3.81 MB (a), 7.63 MB (b) and 38.15 MB (c) global worlds.

Figure 4.21 illustrates the overall execution time of memory constrained planning relative to equivalent quantities of 64 KB local maps for the three largest global map sizes (3.81 MB, 7.63 MB and 38.15 MB). In the comparison, the larger local map (128 KB and 256 KB) data are divided by the 64 KB local map data. Superiority of the 64 KB data is denoted by a ratio greater than unity. A statistically significant comparison has a p-value threshold of 0.05 (5%). It is clearly evident that the 64 KB local map size achieves a superior overall execution time regardless of global map size or local map quantity. Hence, a smaller local map size reduces the overall execution time of memory constrained path planning.

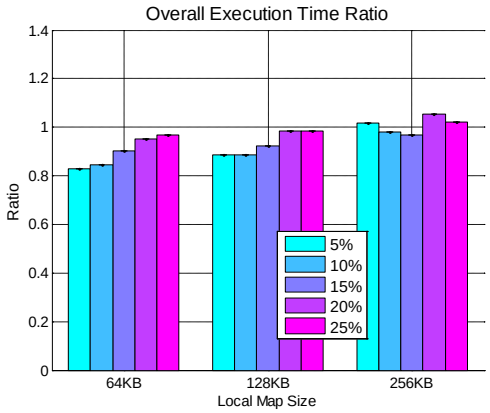
4.6.5 Comparison of Overall Execution Time Using Various Local Map Quantities

A comparison of the ratios of overall memory constrained planning times (inclusive of communication latency) in 38.15 MB worlds as the quantity of local map memories is varied is shown in Figure 4.22. The largest global map size is selected since it demonstrates the purpose of memory constrained planning most appropriately.

In all comparisons, the ratio is computed by dividing the larger local map quantity data by the smaller local map quantity data. Superiority is denoted by a ratio less than unity. A statistically significant comparison has a p-value threshold of 0.05 (5%). Figure 4.22 (a),(d) reveals no statistically significant difference in overall planning time when five local map memories are employed instead of one. There is also no statistically significant difference in overall planning time when fifteen local map memories are employed instead of five (Figure 4.22 (b),(d)). When fifteen local map memories are employed instead of one a statistically significant difference exists at 5% obstacle density when 64 KB local maps are employed (Figure 4.22 (c),(d)). Figure 4.22(d) illustrates that as the local map quantity is increased, the overall execution time ratios and corresponding p-values are lowered for 64 KB local map sizes. This is a desirable result as it indicates that maintaining multiple smaller sized local map memories potentially improves memory constrained planning time. The potential improvement is due to a lower volume of map data being transmitted when multiple local map memories are employed.



(a) (b)



(c)

p-values (two decimal places, < 0.05 shaded blue)									
Obstacle Density	**1 & 5 Local Map Memories (a)**			**5 & 15 Local Map Memories (b)**			**1 & 15 Local Map Memories (c)**		
	64 KB	**128 KB**	**256 KB**	**64 KB**	**128 KB**	**256 KB**	**64 KB**	**128 KB**	**256 KB**
5%	0.30	0.32	0.33	0.13	0.28	0.30	0.04	0.15	0.45
10%	0.35	0.33	0.35	0.14	0.25	0.43	0.07	0.14	0.43
15%	0.45	0.47	0.44	0.20	0.26	0.44	0.17	0.24	0.38
20%	0.46	0.47	0.48	0.36	0.46	0.30	0.33	0.44	0.32
25%	0.45	0.44	0.43	0.42	0.50	0.37	0.38	0.44	0.43

(d)

Figure 4.22: Overall execution time comparison of memory constrained planning in 38.15 MB global worlds with 1 and 5 local map memories (a), 5 and 15 local map memories (b), and 1 and 15 local map memories (c).

4.6.6 Sample Paths

Figure 4.23 and Figure 4.24 illustrate sample paths obtained when a 64 KB local map is employed in memory constrained path planning. Note that these figures have varying magnification due to differing global map sizes. Generally, the memory constrained paths are similar for either cost method. Figure 4.24 has a closely matching path for both cost methods when memory constrained planning is employed.

The increase in volume of map data received when cost method 2 is employed (Figure 4.15(b),(d), Figure 4.16(b),(d) and Figure 4.17(b),(d)) does not justify its use if minimising communication delay is important.

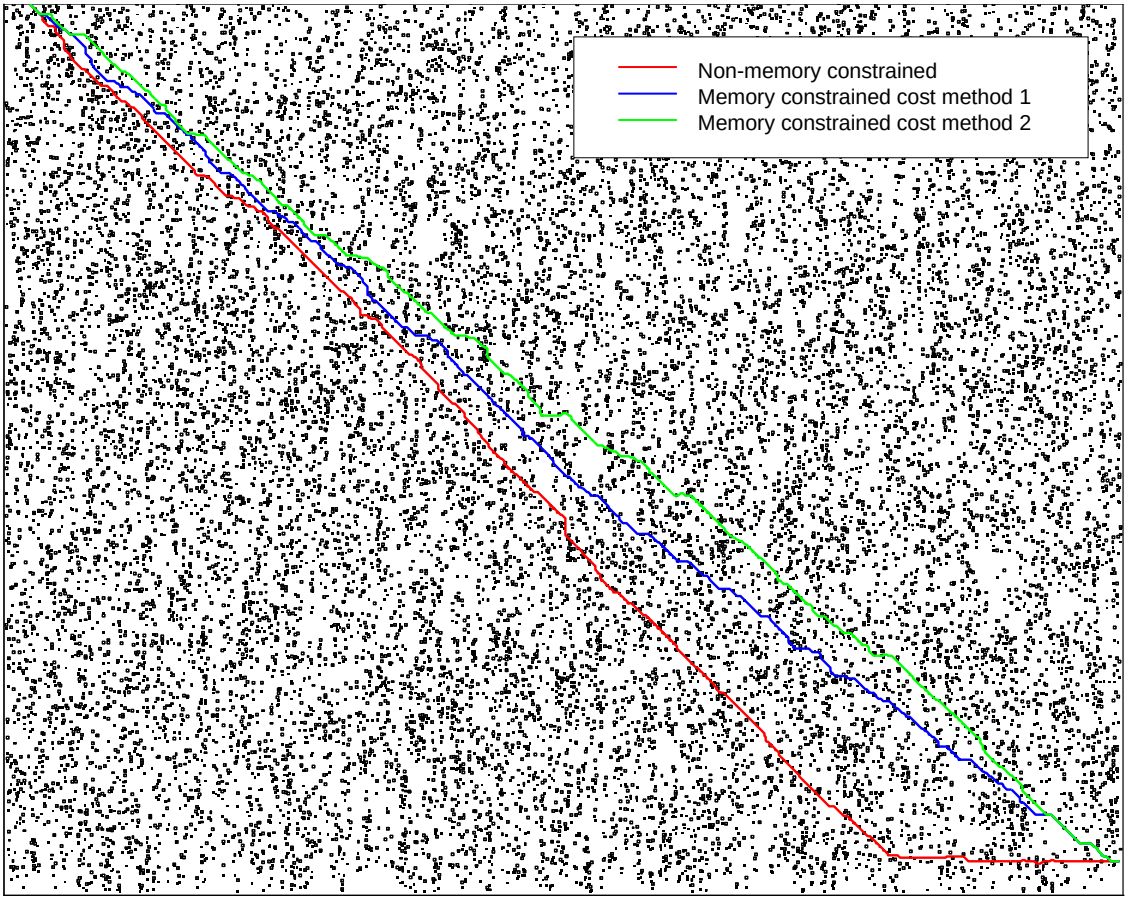


Figure 4.23: Path comparison in a 5% obstacle density 38.15 MB global world employing 64 KB local maps.

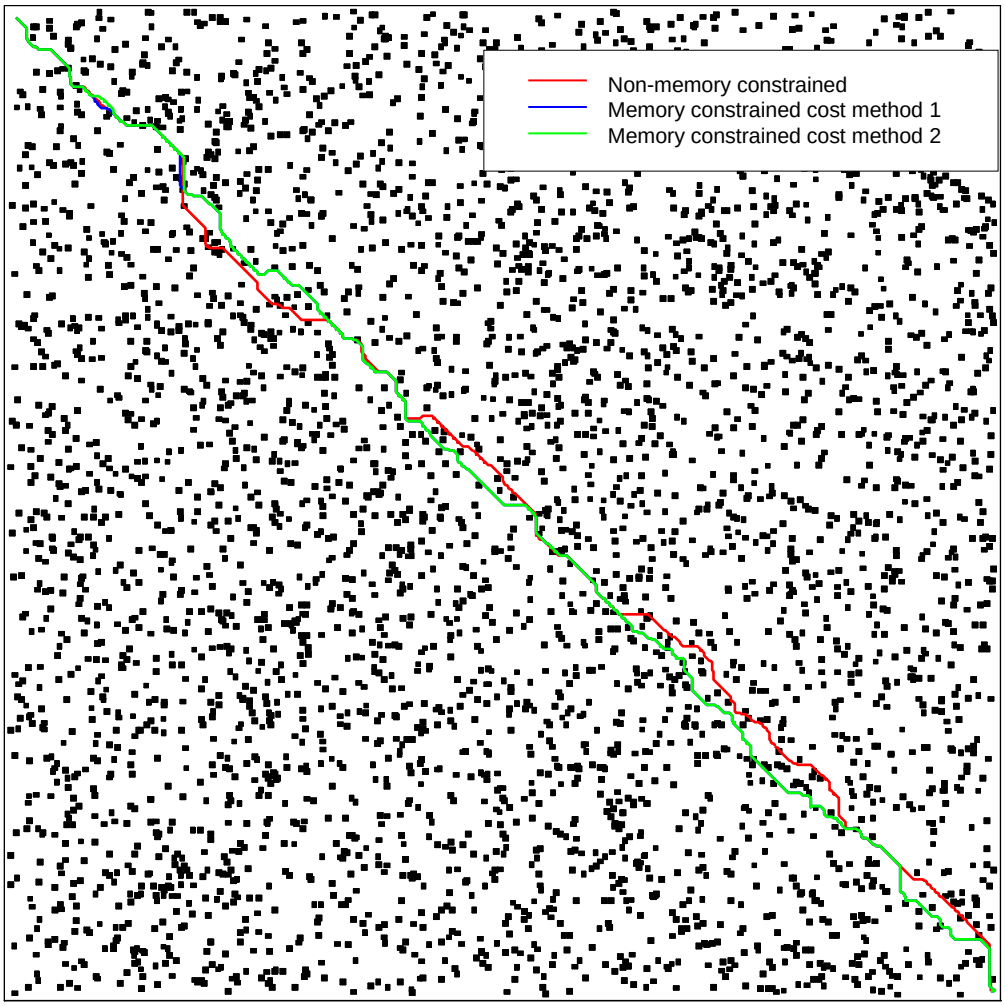


Figure 4.24: Path comparison in a 15% obstacle density 7.63 MB global world employing 64 KB local maps.

4.7 Alternative Techniques

The main purpose of the work presented in this chapter has been to devise a strategy that allows limited memory robots to plan paths in large environments. A regular grid map representation is utilised as an example because of its simplicity and usability in a range of environments. Additionally, it is possible to maintain a single grid map that can be utilised by multiple heterogeneous robots. In section 4.6, the aim is to compare memory constrained and non-memory constrained path planning based on adaptations of the A* algorithm. Planning time and the volume of data transmitted (which can be translated to a component of the overall planning time) are the main performance metrics.

In this chapter, the heuristic cost $d_n(x,x_{goal},x')$ (4.7) is represented by Manhattan distance. Alternatively, this could be represented using Euclidean distance. Manhattan distance is less computationally complex than Euclidean as is does not rely on square and square root calculations. A disadvantage of using Manhattan distance is visible in Figure 4.23, where the non-memory constrained path approaches the bottom of the global map and then travels right to reach the goal locations. A closer path to the diagonal of the global map can be achieved using Euclidean distance. This disadvantage of using Manhattan distance is less visible in the memory constrained methods due to division of the global environment. Regardless of using Manhattan or Euclidean distance, the ratios of memory constrained planning to non-memory constrained planning times are expected to be similar.

The use of the developed two-tiered approach is not limited to regular grid map environments or the A* search algorithm. Other node based map representations such as topological and quadtree maps [7] do not have fixed memory requirements (albeit consume less memory than a grid map in equivalent environments). They may also become large when compared with the available memory on a limited robot and can utilise a method similar to that presented in this chapter. By using equivalent replacements to the A* algorithm cost functions, alternative path planning algorithms such as Dijkstra's algorithm [8], wave propagation (such as NF1) [9], or spreading activation [10] can be used with the grid map node based representation employed in this chapter.

Dijkstra's algorithm has a greater search space than A* algorithm. Hence, the planning time of the A* algorithm will generally be less than Dijkstra's algorithm. However, Dijkstra's algorithm can compute the optimal path from all grid elements to the goal in a single run. This enables it to find the optimal path to all exit points in a single iteration although utilising more memory than the A* algorithm for storing this information. The ratios of memory constrained planning to non-memory constrained planning times are expected to be similar to or better than an A* version. This is due to the memory constrained and non-memory constrained versions of Dijkstra's algorithm having similar search spaces. The path lengths of A* and Dijkstra's algorithms are expected to be the same with the selection of appropriate (monotone) heuristics.

Similar to Dijkstra's algorithm, wave propagation and spreading activation methods can find the optimal path to all exit points in a single iteration. These methods generally have a greater search space than the A* algorithm resulting in greater planning time. It is expected that the ratios of memory constrained planning to non-memory constrained planning for these methods will be similar to or less than the results presented in this chapter. This is due to the memory constrained and non-memory constrained versions of the algorithms having similar search spaces and the computing of paths to exit points in a single iteration. Similar path lengths to the A* algorithm are expected using the wave propagation and spreading activation methods.

The D* algorithm [11] is a variant of the A* algorithm. It computes the optimal path from every location to the goal. Hence, the D* algorithm converts the A* algorithm from a single-source shortest path algorithm into an all-paths algorithm. The D* algorithm is designed for continuous replanning. However, this is computationally expensive and time consuming. It is not practical for robots with processing and memory limitations. It is also impractical to use the D* algorithm with the two-tiered strategy presented in this chapter since local memories are continuously overwritten with new data from other parts of the global world.

None of the methods reviewed in section 2.5 have addressed the problem of global path planning utilising limited memory robots that cannot store an entire global map locally (on the robot). These methods cannot be used to plan paths on limited memory robots unlike the method presented in this chapter which takes advantage of hierarchical heterogeneous multi-robot systems. The general approach taken in robotics has been to use a more computationally powerful robot when the need arises. However, this may not be ideal in a heterogeneous system that comprises many limited robots.

An alternative to the decentralised path planning method proposed in this chapter is to rely on a computationally powerful robot (manager) for centralised planning. Such a technique will outperform the memory constrained method developed in this chapter. However, as highlighted in section 4.1, it may not be possible for the computationally powerful robot to perform global path planning and maintain full communication with all limited memory robots. An additional computationally powerful robot dedicated to

path planning could be employed. However, this costs more money and this additional robot could malfunction. In such scenarios, the memory constrained technique developed in this chapter can be utilised to enable limited memory robots to plan global paths.

4.8 Summary

This chapter has presented and evaluated a novel method for path planning that utilises memory constrained robots in hierarchical heterogeneous multi-robot systems. Rather than relying on a single computationally powerful robot, global path planning can be decentralised by allowing memory restricted robots to utilise the memory of a computationally powerful robot. Since the global map cannot be completely stored in a memory constrained robot, it is divided into smaller local maps based on the memory constrained robot's memory capacity. The local maps are sequentially searched using a two-tiered A* algorithm that executes entirely on the memory constrained robot. However, if a limited memory robot is capable of storing the entire global map locally, the path planner becomes identical to a non-memory constrained approach. This decentralised global path planning approach relies on communication. Thus it incurs data transmission delays and has a finite range of operation.

Utilising 802.11g wireless communication devices for managers and 802.11b devices for workers, memory constrained path planning is capable of achieving superior or comparable execution times to non-memory constrained planning if the local map size is much smaller than the global map size. Employing 64 KB or 128 KB local maps in a 38.15 MB global map produced superior or comparable execution times to non-memory constrained planning. At a resolution of 10 cm, a 38.15 MB grid map corresponds to an area of approximately 225 m × 225 m. This would require additional communication relay nodes or the computationally powerful robot storing the global map to be mobile and relocate if path replanning is needed. By utilising 802.11n devices, communication range and latency can be improved.

Regardless of the global map size, memory constrained path planning time can be reduced if smaller sized local maps are utilised. This can be attributed to a reduction in search space for planning under the guidance of exit points. Employing multiple

smaller sized local map memories potentially improves memory constrained planning. This means that a memory constrained robot can segment its available memory into smaller portions to facilitate improved planning time.

At higher obstacle densities ($\geq 20\%$), memory constrained path planning yields significantly lower execution times than non-memory constrained planning. This indicates a reduced search space and can affect obstacle clearance. However, path length is not adversely affected at these higher obstacle densities. Most real environments generally have overall obstacle densities less than 5% depending on obstacle arrangement. Hence, smaller regions of higher obstacle densities may be confined to a few local maps reducing obstacle clearance issues in memory constrained planning. If obstacles are sufficiently large or the obstacle density is sufficiently high to block some local map boundaries, the back tracking algorithm can find a path through alternative local maps as long as exit points exist in them.

5 Task Allocation and Feedback Coordination Mechanism

5.1 Overview

In some multi-robot applications, predefined task allocation and coordination can fail to function adequately. This failure is attributed to the inability to completely and accurately predict a robot's interactions with the environment before task execution. Robots with limited capabilities (such as restricted processing, sensing and actuation abilities) can also fail to fulfil the requirements of a group task if they do not use their resources effectively. In some applications the system may also want to minimise losses by restricting the quantity of robots committed towards a group task if multiple and/or hazardous group tasks are present.

This chapter presents a task allocation and coordination approach for limited capability mobile robots that can address the shortfalls of predefined task allocation and coordination. To illustrate this approach, an exploration task (fully defined in Chapter 6) is used as a specific example. The algorithm proposed is generic and not limited purely to exploration, but such an example serves to demonstrate how this algorithm functions on a real task. An overview of the proposed system is presented in Figure 5.1.

A global task is specified by a human user in terms of the resources required (section 5.2). These resource requirements are represented using vector of task requirements (VOTR) data. The robots available for task assignment are specified using vector of merit (VOM) data that encodes their capabilities (section 5.3). Fuzzy Inference Systems (FISs) are employed to simplify detailed robot capability information for comparison with the human user's simplified task specification. An example of global task and robot specifications for a multi-robot exploration task is presented in section 5.4.

A team of robots comprising managers and workers is selected during an initial task allocation (or task devolution) process (section 5.5). The managers generally comprise

the most computationally powerful robots while the workers usually have limited computing abilities. A hierarchy can exist within the workers depending on the specified global task. For example, in a multi-robot exploration and mapping task (chapter 6) the worker robots can consist of planners and explorers (Figure 6.2). Numerical vector of task suitability (VOTS) data are employed by the task allocation algorithm to represent a robot's task eligibility. VOTS data are determined from a robot's VOM data and a task's VOTR data.

After initial task allocation a feedback coordination mechanism executes periodically on the manager robot(s) (section 5.6). This feedback mechanism monitors the individual and group performance of the worker robots. If the performance of a worker robot is unsatisfactory, a task reallocation algorithm adjusts the task-robot combinations of the team. Three cases of unsatisfactory robot performance that can be detected by the feedback mechanism include: complete failure, partial failure, and poor performance.

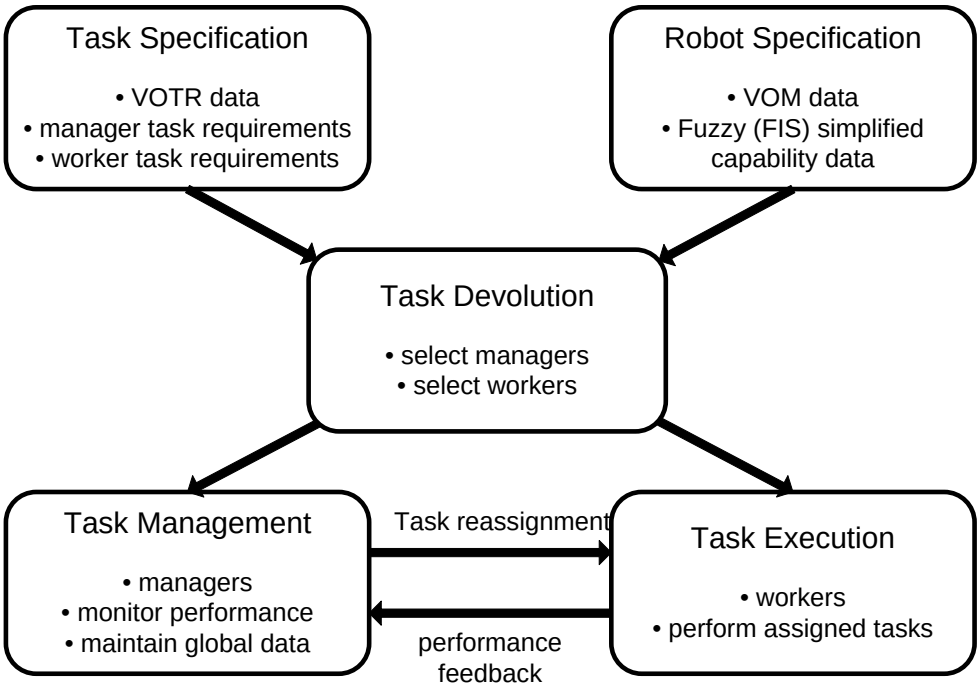


Figure 5.1: Overview of task allocation and coordination mechanism.

5.2 Task Specification

A global task is partially or fully specified by a human user via a remote base station (computer). This global task consists of a set of *n* tasks that are represented by a number of criteria specifying the resources required and conditions for that particular task. Figure 5.2 illustrates the criteria. A task can be partially specified by a basic user with a minimal set of criteria (marked with # in Figure 5.2). In such a case, default values are employed for all other criteria. On the other hand, an advanced user can specify all the task criteria of Figure 5.2 if necessary.

Two categories of tasks are specified: n_1 management tasks and n_2 worker tasks. Management and worker tasks are assigned to appropriate task manager and task worker robots respectively. Each task type can either be one-off or continuous.

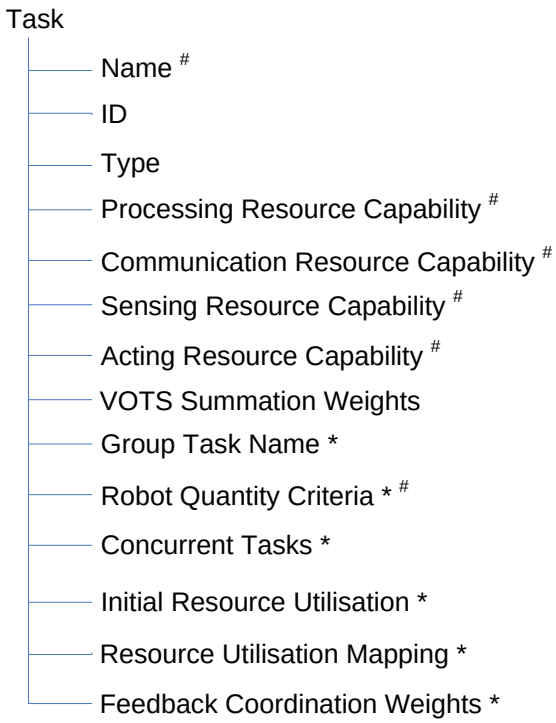


Figure 5.2: Summary of task specification criteria.

There are four divisions of resources: processing, communication, sensing, and actuation. For each task t_i, a minimum capability requirement score t_iRCS_{type} is specified for each resource *type*. To simplify user input, these scores are specified as ‘low’, ‘medium’ or ‘high’ These values are converted to unit interval data where ‘low’ corresponds to zero, ‘medium’ corresponds to 0.35 and ‘high’ corresponds to 0.65, respectively. This enables comparison with the outputs of Fuzzy Inference

Systems (FISs) [136] employed to combine robot resource capability data (section 5.3).

Worker robot tasks employ additional criteria marked with an asterisk in Figure 5.2. The *robot quantity* criteria is employed to determine initial, minimum, and maximum number of worker robots required for the task. For example, in a multi-robot map-building task (chapter 6) the robot quantity criteria can include exploration area size and an explorer-planner task quantity ratio. A worker robot that executes a planner task is called a planner. Similarly, a worker robot executing an explorer task is denoted as an explorer. An FIS (section 5.4) can be employed to map the quantity criteria data to appropriate planner and explorer task quantities. In an object pushing task, robot quantity criteria data can include the mass and size of an object. Using an FIS, this can be mapped to a desired size (or type) and number of robots to move the object.

Tasks that can be simultaneously executed with the current task are listed in the concurrent tasks field. For example, a robot executing a planner task can also execute an explorer task simultaneously if it has sufficient resources. Since there can be multiple identical tasks, there are m_1 management and m_2 worker tasks that are unique.

All tasks require control algorithms that are executed by the robot's processor. Thus, resource utilisation is represented by the control algorithm execution rate toward each physical resource type for a particular task. There are four resource utilisation categories representative of each resource type: planning, communication, sensing, and actuation. Hence, a *resource utilisation mapping* vector (generally not specified or modified by a normal user) is employed to encode each task-robot combination (5.1). A default *initial resource utilisation* is applied for each task-robot combination if a human user has not specified it.

The resource utilisation of each category for each task-robot combination RU_{ijcat} is denoted as:

$$RU_{ijcat} = \left[ru_{ijcat1}, ru_{ijcat2}, \ldots, ru_{ijcats}, \ldots, ru_{ijcatz} \right] \tag{5.1}$$

where:

$cat \in [plan, comm, sense, act]$,

ru_{ijcats} is the s^{th} task dependent resource utilisation sub-category, and

z is the number of sub-categories for resource utilisation RU_{ijcat}.

For the map-building and exploration task of chapter 6, resource utilisation can be represented by the following control algorithms:

- Planning (Processing): global (path planning between local environments and local environment assignment) and local (path planning within local environment and waypoint generation).
- Communication: no sub-categories required.
- Sensing: local map update and obstacle and pose detection.
- Actuation: motion control and motor commands.

Motion control is included in actuation since its output directs the movement of a robot's actuators. Note that motion control includes obstacle avoidance and path tracking but not path planning. Resource utilisation for planning and communication tasks can be expressed as either enabled or disabled since they are generally event-based and non-periodic. On the other hand, sensing and actuation tasks tend to be periodic and can be represented by execution rates.

As the global task progresses, a set of *feedback coordination weights* (section 5.6.1) is employed to monitor the performance of each task-robot combination. If the performance of a task-robot combination is unsatisfactory (section 5.6.1), the resource utilisation parameters are adjusted to enable or disable certain task-robot combinations. A set of rules *RL* (section 5.6.2) is employed by the feedback coordination mechanism to adjust resource utilisation. The feedback coordination weights and rules *RL* can be modified by advanced users if needed.

In this thesis, task reallocation (i.e. enabling or disabling task-robot combinations) is considered equivalent to adjusting a robot's resource utilisation. However, resource utilisation adjustment can also optimise the current task that a robot executes. This can be achieved if the task-resource utilisation mapping vector data are gradually adjusted by the feedback coordination mechanism.

During task devolution (section 5.5), a set of VOTS summation weights VSW_i for task i is employed to convert VOTS data to a single number (5.2).

$$VSW_i = \left[vsw_{iproc}, vsw_{icomm}, vsw_{isense}, vsw_{iact} \right] \tag{5.2}$$

5.3 Robot Specification Description

Similar to the global task specifications, the p robots available for the global task are specified with a number of criteria that represent the resources they possess as shown in Figure 5.3.

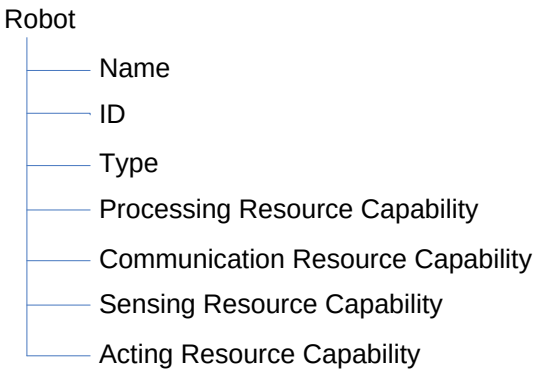


Figure 5.3: Brief description of robot specification criteria.

Numerical VOM data represent the capabilities of robots. For each robot r_j the capability of resource *type* RC_{type} is specified as:

$$RC_{type} = \left[rc_{type1}, rc_{type2}, \ldots, rc_{typek}, \ldots, rc_{typeq} \right] \tag{5.3}$$

where:

$type \in \left[proc, comm, sense, act \right]$,

rc_{typek} is the k^{th} sub-resource type, and

q is the number of sub-resources for resource RC_{type}.

The sub-resources for each resource type are:

- Processing: processor benchmark and memory.
- Communication: bandwidth and range.
- Sensing: quantity, range and distribution for each type of sensor present on

the robot.

- Actuation: operation time, base size, base performance (speed and terrain traversability) and manipulator.

At the task specification stage, a human user provides simplified inputs representing the minimum processing, communication, sensing and actuation resource requirements for a task. As mentioned, the minimum requirements are specified as 'low', 'medium', or 'high' for each resource type. An additional input is provided by the user for the processing resource to select robots with microcontroller based processing ('mc') or desktop PC equivalent based processing ('pc'). Similarly, a robot size input is also supplied by the user for the actuation resource if robots of a particular size are required ('small', 'medium' or 'large'). If robot size is irrelevant, the default value of 'any' is employed.

Unlike the simplified human user inputs, the robots are specified with detailed information. This information has a variety of data values and measurement units for each resource and sub-resource type. In this chapter, the robot resource data values are based on real robots such as those in the VUW fleet and other robot types in the literature (chapter 2). A mechanism is required to combine the variety of robot information into a single value for each resource type for comparison with the simplified human user input. Fuzzy systems [136] are favoured as they employ fuzzy sets and rules to permit graded (such as 'low', 'medium' and 'high') outputs. This provides a convenient way to map the detailed robot specification to a simplified form. Hence, FISs are employed to produce a simplified unit interval value for each resource type.

MATLAB® 2007a Fuzzy Logic Toolbox has been used to develop the FISs. All FISs described in this section are Mamdani type [136] (MATLAB's default FIS type). Two basic types of membership functions are used for simplicity. These are triangular membership functions and trapezoidal membership functions. Other membership function types such as Gaussian or sigma can be utilised. During initial experiments, the triangular and trapezoidal functions were able to produce results similar to Gaussian or sigma types.

A maximum of three membership functions are employed for all inputs and outputs to reduce the complexity of the fuzzy systems. All membership function ranges, fuzzy rules and their corresponding weights have been empirically tuned to select "good" parameter values. The outputs of the various input combinations (produced using membership functions, fuzzy rules and weights) have been viewed with MATLAB's surface viewer during empirical tuning. With three membership functions a reasonably smooth transitioning surface has been generated for the various input combinations. Incorporating more than three membership functions results in more rules and combinations to consider. The surface generated with a five membership function system was found to be similar to that of a three membership function system during initial testing.

Triangular membership functions are specified with three parameters [tri_1 tri_2 tri_3]. tri_1 is the minimum data value at which the degree of membership is zero, tri_2 is the data value at which the degree of membership is unity and tri_3 is the maximum data value at which the degree of membership is zero. For example, in Figure 5.4 the 'medium' membership function (in green) is specified as [0.3 0.5 0.7]. The 'low' membership function is a half triangle in Figure 5.4. This is achieved by setting tri_1 to a value less than the global minimum data value (i.e. less than zero in Figure 5.4). Similarly, a half triangle is obtained for the 'high' membership function by setting tri_3 to a value greater than the global maximum data value (i.e. greater than one in Figure 5.4)

Trapezoidal membership functions are specified with four parameters [$trap_1$ $trap_2$ $trap_3$ $trap_4$]. $trap_1$ and $trap_4$ are similar to tri_1 and tri_3. They represent the minimum and maximum data values at which the degree of membership is zero. $trap_2$ is the minimum data value at which the degree of membership is unity, while $trap_3$ is the maximum data value at which the degree of membership is unity. An example of trapezoidal membership functions is illustrated in Figure 5.5. The 'medium' membership function (in green) is specified as [0.2 0.4 0.6 0.8]. A partial trapezoid is obtained for the 'low' membership function by placing parameters $trap_1$ and $trap_2$ below the global minimum data value. Similarly, by setting $trap_3$ and $trap_4$ above the global maximum data value, a partial trapezoid is obtained for the 'high' membership function.

The empirically selected three output membership functions employed by each FIS are illustrated in Figure 5.4. The ranges for ‘low’, ‘medium’ and ‘high’ are [0 0.4], [0.3 0.7] and [0.6 1], respectively. Consequently, the numerical threshold values for ‘low’, ‘medium’ and ‘high’ in the task specification are intuitively selected as 0, 0.35 and 0.65, respectively.

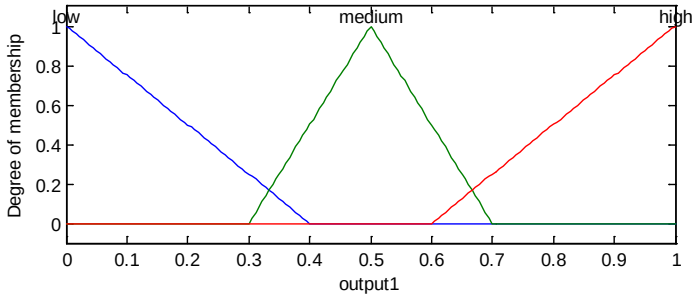


Figure 5.4: FIS output membership functions.

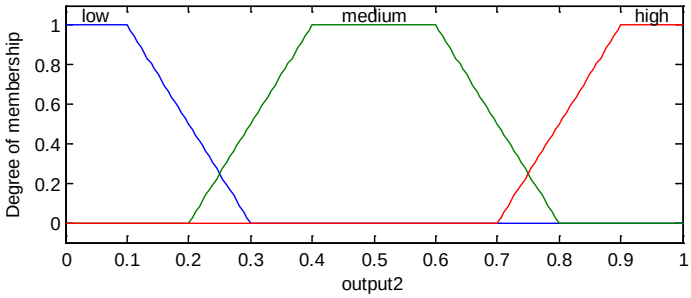


Figure 5.5: Trapezoidal membership functions.

Table 5.1 shows the parameter settings for the fuzzy inference functions of all implemented FISs. MATLAB default settings are employed for the ‘And’, ‘Or’, ‘Implication’ and ‘Defuzzification’ functions. The ‘sum’ method has been selected as the ‘Aggregation’ function to produce a gradual change in output as the inputs are varied.

Table 5.1: Fuzzy inference function settings.

Function	Method
And	min
Or	max
Implication	min
Aggregation	sum
Defuzzification	centroid

The input details for each resource type’s FIS span a variety of robots and values are consistent with other robot types in the literature.

5.3.1 Processing Fuzzy Inference System

Two FISs are employed to combine the processing resources. One is used to combine the processing resources of a robot with microcontroller based processing. The other is employed for combining desktop PC based equivalent processing resources.

Figure 5.6(a) illustrates the microcontroller (MC) processing FIS. It has two inputs: MC processor benchmark and MC memory (Table 5.2). The Embedded Microprocessor Benchmark Consortium (EMBC) (http://eembc.org) produces performance benchmarks for embedded systems. It employs a CoreMark 1.0 test to evaluate a processor's benchmark. A selected range of microcontrollers have been tested. It is possible to obtain additional benchmark data by testing a wider range of microcontrollers.

Benchmark data can be scaled to the range of [0 10] where 10 corresponds to the most powerful microcontroller. Generally, 8-bit microcontrollers fall into the 'low' benchmark category. The 'medium' and 'high' benchmark categories generally comprise of 16-bit and 32-bit microcontrollers, respectively. For MC memory, the 'low', 'medium' and 'high' divisions have been arbitrarily selected. The upper limit for the memory input has been determined based on powerful microcontrollers such as the ARM7 which can typically support at least 16 MB external memory. Some of the latest ARM7 microcontrollers (such as STR750XX) can support up to 64 MB of flash memory.

Table 5.3 details the fuzzy rules employed to combine the MC processing benchmark and MC memory inputs into the processing score RCS_{proc}. So for example, if Processor Benchmark is low then Processing Score will be low.

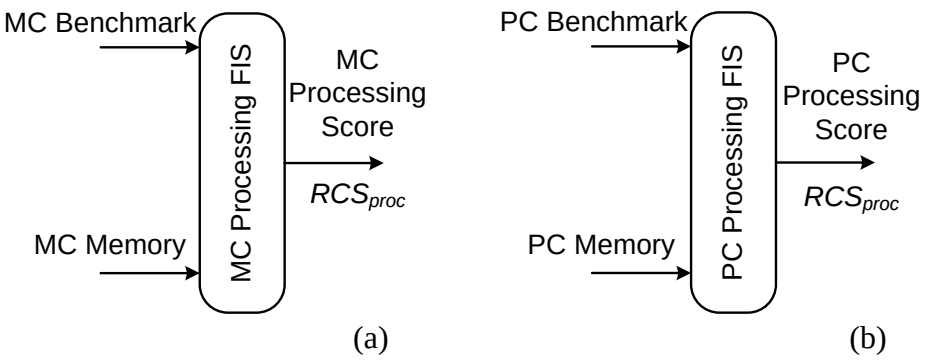


Figure 5.6: Diagrams of Microcontroller (MC) and PC Processing FISs.

Table 5.2: Microcontroller processing FIS inputs.

Input 1	Name		Processor Benchmark		
	Data Range		[0 10]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–4 0 4]	[2.5 5 7.5]	[6 10 14]
Input 2	Name		Memory		
	Data Range (MB)		[0 16]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–6 0 6]	[4 8 12]	[10 16 32]

The desktop PC based equivalent processing FIS is illustrated in Figure 5.6(b). A PC processor benchmark and memory (RAM) are input to the FIS (Table 5.4). MATLAB's 'bench' function data has been utilised to estimate a processor's benchmark. Several MATLAB users have posted benchmark test results of their computers on MATLAB Central (http://www.mathworks.com/matlabcentral/).

Table 5.3: Rule table for MC and PC processing FISs.

Rule No.	**Processor Benchmark**	**Memory**	**Inputs Connection**	**Processing Score**	**Rule Weight**
1	low	–	–	low	1
2	medium	–	–	medium	1
3	high	–	–	high	1
4	–	low	–	low	1
5	–	medium	–	medium	1
6	–	high	–	high	1

Table 5.4: Desktop PC based equivalent processing FIS inputs.

Input 1	Name		Processor Benchmark		
	Data Range		[0 5]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–2 0 2]	[1.25 2.5 3.75]	[3 5 7]
Input 2	Name		Memory		
	Data Range (GB)		[0 2]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	trapezoidal
		Parameters	[–0.6 0 0.6]	[0.2 0.75 1.3]	[0.9 1.75 2.08 2.72]

Based on these data, it is estimated that the processor benchmark of a computer employed for mobile robot applications can vary from zero to five. 'Low' benchmark processors generally include Pentium III, Atom, Celeron and Pentium M. Pentium M, Pentium IV, Core Solo and Core Duo processors are likely to have a 'medium' benchmark. 'High' benchmark processors generally include Pentium D and Core 2

Duo. For memory (RAM) the ‘low’ category generally includes 128 MB and 256 MB RAM. The ‘medium’ classification can include 512 MB and 1 GB RAM. A computer with at least 1.5 GB memory has a ‘high’ memory rating.

Similar to the MC processing FIS, Table 5.3 details the fuzzy rules used to determine the processing score RCS_{proc} for PC processing.

5.3.2 Communication Fuzzy Inference System

Figure 5.7 illustrates the communication FIS. It has two inputs: bandwidth and range (Table 5.5). ‘High’ bandwidth communication is typically 802.11a or 802.11g wireless communication (54 MBits/sec). ‘Medium’ bandwidth communication corresponds to 802.11b wireless communication (11 MBits/sec) or enhanced versions of the 802.11b capable of 22 MBits/sec data rates. ‘Low’ bandwidth communication includes Bluetooth communication (1–3 MBits/sec) and infrared (IrDA) communication (2.4 Kbits/sec to 4 MBits/sec).

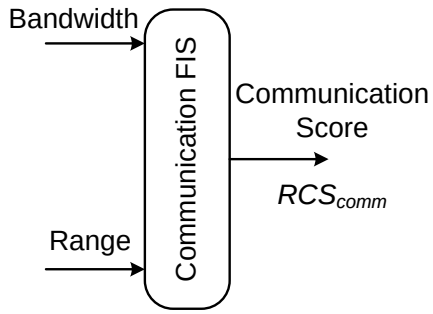


Figure 5.7: Diagram of Communication FIS.

Table 5.5: Communication FIS inputs.

Input 1	Name		Bandwidth		
	Data Range (MBits/sec)		[0 54]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–8.5 0 8.5]	[3.5 11 36]	[16.5 54 75.6]
Input 2	Name		Range		
	Data Range (m)		[0 100]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–40 0 40]	[25 50 75]	[60 100 140]

The communication range input corresponds to the typical maximum range at which maximum bandwidth is possible. High range (~100 m) communication is possible with 802.11b and 802.11g devices. Medium range communication of approximately

30 m is possible with 802.11a. Bluetooth and infrared are low range communication devices with typical ranges of 10 m and 1–2 m, respectively. Table 5.6 details the fuzzy rules to combine communication bandwidth and range into a communication score RCS_{comm}.

Table 5.6: Communication FIS fuzzy rules.

Rule No.	Bandwidth	Range	Inputs Connection	Communication Score	Rule Weight
1	low	–	–	low	1
2	medium	–	–	medium	1
3	high	–	–	high	1
4	–	low	–	low	1
5	–	medium	–	medium	1
6	–	high	–	high	1

5.3.3 Sensing Fuzzy Inference System

A general overview of an FIS that can be employed to combine the capabilities of multiple sensors is illustrated in Figure 5.8. The inputs to the multi-sensor FIS are the overall scores for each type of sensor present on the robot RCS_{sensek}. Overall scores for each sensor type are computed from individual FISs. Depending on the type of task, the fuzzy rules and corresponding weights of the multi-sensor FIS can be empirically selected. The output of the FIS is an overall multi-sensing score RCS_{sense} representing the combined sensing ability of the robot.

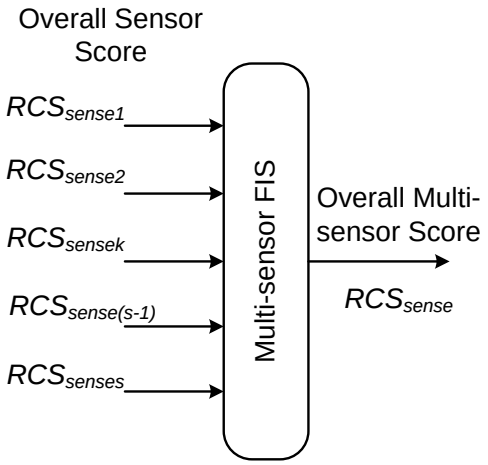


Figure 5.8: Diagram of Multi-sensor FIS.

This section only describes the FIS implementation for IR sensors as they are the primary sensors employed by the map-building and exploration task presented in this thesis. Thus, the multi-sensor FIS only gives weighting to the IR sensors. By altering

the data ranges and fuzzy rules, similar FISs can be produced for other types of sensors such as ultrasonic and laser. Sensor distribution and range inputs can be substituted with field of vision (FOV) and depth data for cameras.

A block diagram of the IR sensing FIS is shown in Figure 5.9. It consists of four FISs that determine a sensor score, an obstacle avoidance score, a mapping/exploration score and an overall sensor score, respectively. The inputs to the IR sensing FIS are the IR sensing sub-capabilities of sensor distribution, sensor quantity and sensing range. A sensor score FIS takes the sensor quantity and sensing range as input to determine a sensor score for the IR sensors. Next, the sensor score and sensor distribution data are input to two FISs that determine obstacle avoidance and mapping/exploration scores. Finally, the obstacle avoidance and mapping/exploration scores are combined in an overall sensor score FIS to calculate an overall score for the IR sensors.

Table 5.7 and Table 5.8 detail the sensor score FIS inputs and fuzzy rules, respectively. Sensor quantity data range and divisions have been selected based on experience with robots in the VUW fleet. Commercially available IR sensors from manufacturers such as Sharp presently have a maximum sensing (‘high’) range of approximately three metres. Sharp also manufactures ‘medium’ range (1.5 m) and ‘low’ range (0.3–0.8 m) sensors.

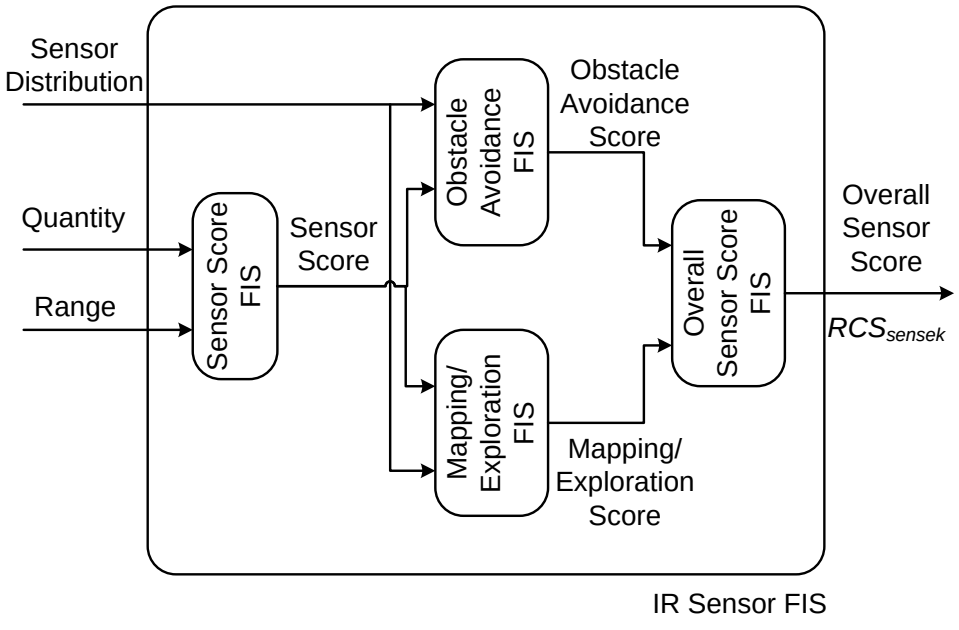


Figure 5.9: Block Diagram of IR Sensing FIS.

Table 5.7: Sensor score FIS inputs.

Input 1	Name		Sensor Quantity		
	Data Range		[0 15]		
	Membership Function Details	Name	low	medium	high
		Type	Triangular	triangular	triangular
		Parameters	[–7.5 0 7.5]	[0 7.5 15]	[7.5 15 21]
Input 2	Name		Range		
	Data Range (m)		[0 3]		
	Membership Function Details	Name	low	medium	high
		Type	Triangular	triangular	triangular
		Parameters	[–1.5 0 1.5]	[0 1.5 3]	[1.5 3 4.5]

Table 5.8: Sensor score FIS fuzzy rules.

Rule No.	Sensor Quantity	Range	Inputs Connection	Sensor Score	Rule Weight
1	low	–	–	low	1
2	medium	–	–	medium	1
3	high	–	–	high	1
4	–	low	–	low	1
5	–	medium	–	medium	1
6	–	high	–	high	1

The obstacle avoidance FIS sensor distribution and sensor score inputs are detailed in Table 5.9. Sensor distribution is classified into three categories: ‘low’, ‘medium’ and ‘high’. The range of each category has been selected based on experience with robots in the VUW fleet.

Table 5.9: Obstacle avoidance FIS inputs.

Input 1	Name		Sensor Distribution		
	Data Range (degrees)		[0 360]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–90 0 90]	[45 135 280]	[180 360 540]
Input 2	Name		Sensor Score		
	Data Range		[0 1]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–0.5 0 0.5]	[0 0.5 1]	[0.5 1 1.5]

Sensors covering less than a quarter of a robot’s circumference (< 45°) have ‘low’ distribution and will generally be poor (‘low’) at obstacle avoidance. If approximately half (usually front half) of the robot’s circumference is covered (~180° distribution), obstacle avoidance is usually good (‘high’). Full coverage of a robot’s perimeter (~360° distribution) generally yields average (‘medium’) obstacle avoidance for IR sensing. An average grading is given to full coverage since there is likely to be a less

dense sensor distribution than half coverage for an identical number of sensors. The sensor score input incorporates sensor quantity information from the sensor score FIS. Table 5.10 details the six fuzzy rules employed to calculate the obstacle avoidance score.

Table 5.10: Obstacle avoidance FIS rules.

Rule No.	Sensor Distribution	Sensor Score	Inputs Connection	Obstacle Avoidance Score	Rule Weight
1	low	–	–	low	1
2	medium	–	–	high	1
3	high	–	–	medium	1
4	–	low	–	low	1
5	–	medium	–	medium	1
6	–	high	–	high	1

Table 5.11 details the inputs to the mapping/exploration FIS. For simplicity, only two classifications (‘low’ and ‘high’) are made for sensor distribution. Based on experience with the VUW robot fleet, a robot should have at least one quarter of its perimeter covered with sensors to qualify for membership into the ‘high’ category. Table 5.12 lists the seven fuzzy rules employed by the mapping/exploration FIS. The sensor distribution input biases the sensor score input data in the mapping/exploration score calculation. A robot that does not have a ‘high’ sensor distribution will have a reduced mapping/exploration score.

Table 5.11: Mapping/exploration FIS inputs.

Input 1	Name		Sensor Distribution		
	Data Range (degrees)		[0 360]		
	Membership Function Details	Name	low	high	
		Type	triangular	triangular	
		Parameters	[–270 0 270]	[90 360 540]	
Input 2	Name		Sensor Score		
	Data Range		[0 1]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–0.5 0 0.5]	[0 0.5 1]	[0.5 1 1.5]

Table 5.12: Mapping/exploration FIS fuzzy rules.

Rule No.	Sensor Distribution	Sensor Score	Inputs Connection	Mapping/Exploration Score	Rule Weight
1	–	low	–	low	1
2	high	medium	and	medium	1
3	high	high	and	high	1
4	not high	–	–	low	1
5	not high	medium	and	low	0.6
6	not high	high	and	medium	0.25
7	high	low	and	medium	0.1

The obstacle avoidance and mapping/exploration scores are combined by the overall sensor score FIS. Since the obstacle avoidance and mapping/exploration scores are both outputs of other FISs, their membership functions are as illustrated in Figure 5.4 (p.119). Table 5.13 lists the seven fuzzy rules utilised to determine the overall sensor score RCS_{sense1} for the IR sensors. Rule 1 has full weighting (unity) since poor ('low') obstacle avoidance can adversely affect a robot's movement. Rules 2–4 have low weights (0.1) because the global task example in this thesis is multi-robot exploration and map building. The overall multi-sensor score RCS_{sense} is represented by the individual sensor score RCS_{sense1} since the robots in this thesis employ only IR sensors.

Table 5.13: Overall sensor score FIS fuzzy rules.

Rule No.	Obstacle Avoidance Score	Mapping/ Exploration Score	Inputs Connection	Overall Sensor Score	Rule Weight
1	low	–	–	low	1
2	medium	–	–	low	0.1
3	medium	–	–	medium	0.1
4	high	–	–	high	0.1
5	–	low		low	1
6	–	medium	–	medium	1
7	–	high	–	high	1

5.3.4 Actuation Fuzzy Inference System

Figure 5.10 illustrates the actuation FIS. It comprises three FISs that determine a base performance score, a base size score and an overall actuation score, respectively. The inputs to the actuation FIS are four actuation sub-capabilities. These are operation time, speed, difficult terrain traversability and base size. The base performance FIS takes speed and difficult terrain traversability as inputs to calculate a base performance score for a robot. A base size FIS maps the size of a robot into a preference score. Next, the base performance score, base size score and robot operation time are input to the overall actuation FIS to compute an overall actuation score.

Base performance score FIS inputs are presented in Table 5.14. The range and divisions for the speed input have been selected based on robots in the VUW fleet. A safe autonomous navigation speed near or above 1 m/sec is considered to be 'high'. Robots with safe navigation speeds of approximately 0.5 m/sec have 'medium' speed.

A robot with a safe navigation speed of below 0.3 m/sec is likely to be regarded as a slow (‘low’ speed) robot.

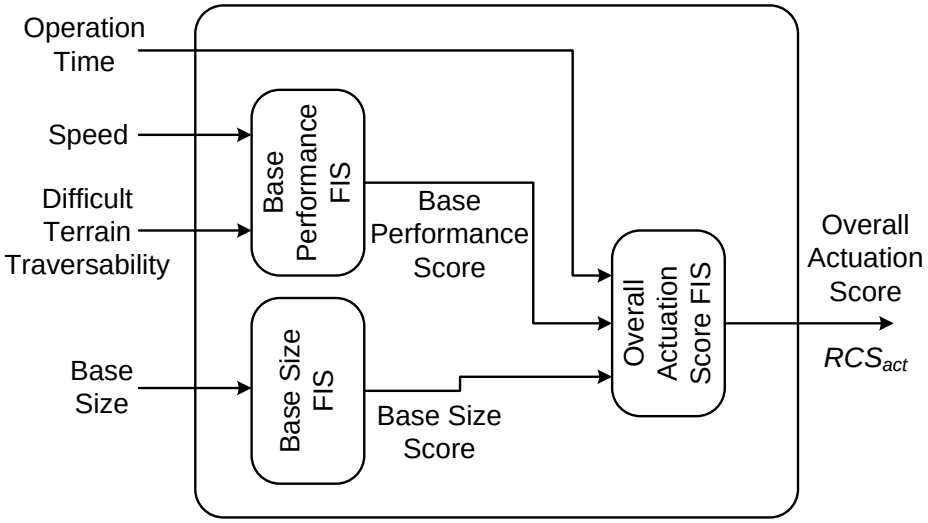


Figure 5.10: Block Diagram of Actuation FIS.

The difficult terrain traversability input is unit interval. At the very basic level terrain traversability can be represented with binary data. A value of zero can represent poor (‘low’) traversability while a value of one represents good (‘high’) traversability. It is possible to represent difficult terrain traversability with better precision by evaluating the movement of a robot in difficult terrain, such as uneven or boggy surfaces. The latter representation is employed for robots in this thesis but reasonable terrain traversability approximations are made for each robot drive type. Movement is not evaluated for real robots.

Table 5.14: Base performance FIS inputs.

Input 1	Name		Speed		
	Data Range (m/sec)		[0 1]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–0.5 0 0.5]	[0.3 0.5 0.65]	[0.5 1 1.5]
Input 2	Name		Difficult Terrain Traversability		
	Data Range		[0 1]		
	Membership Function Details	Name	low	high	
		Type	triangular	triangular	
		Parameters	[–0.4 0 1]	[0 1 1.4]	

Table 5.15 lists the fuzzy rules employed to compute the base performance score. Lower weightings are given to the difficult terrain traversability input rules since terrain acts as a modifier to the speed input.

Table 5.15: Base performance FIS fuzzy rules.

Rule No.	Speed	Difficult Terrain Traversability	Inputs Connection	Base Performance Score	Rule Weight
1	low	–	–	low	1
2	medium	–	–	medium	1
3	high	–	–	high	1
4	–	low	–	low	0.1
5	–	high	–	high	0.3

Base size score FIS inputs and outputs are listed in Table 5.16. The base size input represents the robot's radius (when a circle is prescribed from the robot's centre to encompass the robot). 'Small', 'medium' and 'large' divisions have been arbitrarily selected assuming a maximum robot radius of one metre. The base size score output is unit interval and is employed by the overall actuation score FIS as a modifier. Table 5.17 lists three rules that can be employed to give high preference to a robot of any size. These rules can be altered if a particular sized robot is desired.

Table 5.16: Base size FIS input and output.

Input 1	Name		Size		
	Data Range (m)		[0 1]		
	Membership Function Details	Name	small	medium	large
		Type	triangular	triangular	triangular
		Parameters	[–0.5 0 0.5]	[0 0.5 1]	[0.5 1 1.5]
Output 1	Name		Size Score		
	Data Range		[0 1]		
	Membership Function Details	Name	low	high	
		Type	trapezoidal	trapezoidal	
		Parameters	[–0.36 -0.04 0.35 0.55]	[0.45 0.65 1.04 1.36]	

Table 5.17: Base size FIS fuzzy rules.

Rule No.	Size	Base Size Score	Rule Weight
1	not low	high	1
2	not medium	high	1
3	not high	high	1

Table 5.18 lists the three inputs to the overall actuation FIS. Operation time input details are based on the variety of robot types found in the literature (chapter 2) and reasonable estimates for different types of tasks. A robot with less than one hour of operation has 'low' operation time. Robots with one to two hours of operation time are considered to be 'medium'. A robot that can operate for longer than two hours has a 'high' operation time. The base performance score input is a standard FIS output

(Figure 5.4, p.119). Similar to the base size FIS output, the base size score input uses two trapezoidal rules.

Table 5.18: Overall actuation score FIS inputs.

Input 1	Name		Operation Time		
	Data Range (hours)		[0 3]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–1.2 0 1.2]	[0.9 1.5 2.1]	[1.8 3 4.2]
Input 2	Name		Base Performance Score		
	Data Range		[0 1]		
	Membership Function Details	Name	low	medium	high
		Type	triangular	triangular	triangular
		Parameters	[–0.4 0 0.4]	[0.3 0.5 0.7]	[0.6 1 1.4]
Input 3	Name		Base Size Score		
	Data Range		[0 1]		
	Membership Function Details	Name	low	high	
		Type	trapezoidal	trapezoidal	
		Parameters	[–0.36 -0.04 0.35 0.55]	[0.45 0.65 1.04 1.36]	

The set of fuzzy rules employed by the overall actuation score FIS to compute the overall actuation score RCS_{sense} is listed in Table 5.19. These rules were empirically formulated. Rule 3 is employed to give low overall actuation scores to robots that do not meet the base size requirement. If the base size score is good (‘high’), rules 4 to 7 permit medium and high overall actuation scores.

Table 5.19: Overall actuation score FIS fuzzy rules.

Rule No.	Operation Time	Base Performance Score	Base Size Score	Inputs Connection	Overall Actuation Score	Rule Weight
1	low	–	–	–	low	1
2	–	low	–	–	low	1
3	–	–	low	–	low	1
4	medium	–	high	and	medium	1
5	–	medium	high	and	medium	1
6	high	–	high	and	high	1
7	–	high	high	and	high	1

5.4 Task and Robot Specifications for a Multi-Robot Map Building and Exploration Task

As an example, the manager and worker task requirements for a multi-robot map building and exploration task are shown in Table 5.20 and Table 5.21 respectively. Similarly, the capabilities of eight candidate robots for the multi-robot map building task are shown in Table 5.22. The capability data of the robots are based on mobile

robots in the VUW fleet (Figure 5.11, p.136). Some of the robots in the fleet have obsolete processing hardware and are in the process of being upgraded.

Table 5.20: Manager task specifications.

Task ID	Criteria	Value
M1	Name	maintain global info
	Type	continuous
	Processing Capabilities $t_{M1}RCS_{proc}$	[pc,low]
	Communication Capabilities $t_{M1}RCS_{comm.}$	[medium]
	Sensing Capabilities $t_{M1}RCS_{sense}$	[low]
	Actuation Capabilities $t_{M1}RCS_{act}$	[medium,any]
	VOTS Summation Weights [proc,comm,sense,act] VSW_{M1}	[0.5,0.5,0,0]
M2	Name	secondary task devolution & feedback system
	Type	Continuous
	Processing Capabilities $t_{M2}RCS_{proc}$	[pc,medium]
	Communication Capabilities $t_{M2}RCS_{comm.}$	[high]
	Sensing Capabilities $t_{M2}RCS_{sense}$	[low]
	Actuation Capabilities $t_{M2}RCS_{act}$	[medium,any]
	VOTS Summation Weights [proc,comm,sense,act] VSW_{M2}	[0.5,0.5,0,0]

Processing capability data represents the processor benchmark and the available memory. A manager robot executing task M1 (Table 5.20) needs to have at least ‘low’ PC based processing . In Table 5.22, Robot1 has PC based processing with an overall score of 0.48 (‘medium’).

Communication capability data comprises bandwidth and range. For example, in Table 5.21 worker task W1 requires an overall communication capability of ‘medium’. A communication bandwidth of 11 MBit/sec and range of 100 m is available on Robot2 (Table 5.22). Thus, Robot 2 has an overall communication score of 0.62 (‘medium’).

The sensing capability data represents infrared sensing ability as infrared sensors are primarily employed for the map-building and exploration task presented in this thesis. Worker task W2 (Table 5.21) requires a robot with at least ‘medium’ sensing capabilities. Robot5 (Table 5.22) has ten 1.5 m range infrared sensors that are evenly distributed (360°) around the robot. The overall sensing score for Robot5 is 0.49 (‘medium’).

Table 5.21: Worker task specifications.

Task ID	Criteria	Value
W1	Name	planner
	Type	continuous
	Processing Capabilities $t_{W1}RC_{proc}$	[mc,medium]
	Communication Capabilities $t_{W1}RC_{comm.}$	[medium]
	Sensing Capabilities $t_{W1}RC_{sense}$	[low]
	Actuation Capabilities $t_{W1}RC_{act}$	[low,any]
	Group Task Name	multi-robot map-building
	VOTS Summation Weights [proc,comm,sense,act] VSW_{W1}	[0.4,0.4,0.1,0.1]
	Robot Quantity Criteria	[1600,medium] → [1,1,4]
	Concurrent Tasks	[W2]
	Initial Resource Utilisation	[on]
	Resource Utilisation Mapping	
	[on]	
	Planning	[-1,0]
	Communication	[-1]
	Sensing	[0,0]
	Actuation	[0,0]
	[off]	
	Planning	[0,0]
	Communication	[-1]
	Sensing	[0,0]
	Actuation	[0,0]
W2	Name	explorer
	Type	continuous
	Processing Capabilities $t_{W2}RC_{proc}$	[mc,low]
	Communication Capabilities $t_{W2}RC_{comm}$	[low]
	Sensing Capabilities $t_{W2}RC_{sense}$	[medium]
	Actuation Capabilities $t_{W2}RC_{act}$	[medium,any]
	Group Task Name	multi-robot map-building
	VOTS Summation Weights [proc,comm,sense,act] VSW_{W2}	[0.05,0.05,0.45,0.45]
	Robot Quantity Criteria	[1600,medium] → [4,1,4]
	Concurrent Tasks	[W1]
	Initial Resource Utilisation	[on]
	Resource Utilisation Mapping	
	[on]	
	Planning	[0,-1]
	Communication	[-1]
	Sensing	[10,10]
	Actuation	[10,10]
	[off]	
	Planning	[0,0]
	Communication	[-1]
	Sensing	[0,0]
	Actuation	[0,0]

Table 5.22: Capability data of eight heterogeneous robots.

Robot ID	Criteria	Value	Score ($r_i RCS_{type}$)
1	Name	Robot1	–
	Type	tricycle-pentagon	–
	Processing Capabilities RC_{proc}	[3.2,0.5,pc]	[pc,0.48]
	Communication Capabilities $RC_{comm.}$	[54,100]	[0.87]
	Sensing Capabilities RC_{sense}	[11,1.5,360]	[0.51]
	Actuation Capabilities RC_{act}	[2.04,0.50,0.50,0.30]	[0.60]
2	Name	Robot2	–
	Type	tricycle-pentagon	–
	Processing Capabilities RC_{proc}	[6,10,mc]	[mc,0.52]
	Communication Capabilities $RC_{comm..}$	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[11,1.5,360]	[0.51]
	Actuation Capabilities RC_{act}	[2.14,0.50,0.50,0.30]	[0.65]
3	Name	Robot3	–
	Type	differential-circular	–
	Processing Capabilities RC_{proc}	[2.1,0.25,pc]	[pc,0.45]
	Communication Capabilities $RC_{comm.}$	[54,100]	[0.87]
	Sensing Capabilities RC_{sense}	[10,1.5,360]	[0.49]
	Actuation Capabilities RC_{act}	[2.15,0.40,0.50,0.1]	[0.66]
4	Name	Robot4	–
	Type	differential-rectangular	–
	Processing Capabilities RC_{proc}	[10,16,mc]	[mc,0.87]
	Communication Capabilities $RC_{comm.}$	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[14,1.5,360]	[0.55]
	Actuation Capabilities RC_{act}	[2.53,0.60,0.40,1]	[0.67]
5	Name	Robot5	–
	Type	differential-circular	–
	Processing Capabilities RC_{proc}	[6,4,mc]	[mc,0.37]
	Communication Capabilities	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[10,1.5,360]	[0.49]
	Actuation Capabilities RC_{act}	[1.25,0.40,0.50,0.1]	[0.5]
6	Name	Robot6	–
	Type	differential-circular	–
	Processing Capabilities RC_{proc}	[6,2,mc]	[mc,0.33]
	Communication Capabilities $RC_{comm.}$	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[10,1.5,360]	[0.49]
	Actuation Capabilities RC_{act}	[1.40,0.40,0.50,0.1]	[0.5]
7	Name	Robot7	–
	Type	differential-circular	–
	Processing Capabilities RC_{proc}	[6,4,mc]	[mc,0.37]
	Communication Capabilities $RC_{comm.}$	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[10,1.5,360]	[0.49]
	Actuation Capabilities RC_{act}	[1.19,0.40,0.50,0.1]	[0.48]
8	Name	Robot8	–
	Type	differential-rectangular	–
	Processing Capabilities RC_{proc}	[6,2,mc]	[mc,0.33]
	Communication Capabilities $RC_{comm.}$	[11,100]	[0.62]
	Sensing Capabilities RC_{sense}	[14,1.5,360]	[0.55]
	Actuation Capabilities RC_{act}	[0.95,0.60,0.40,1]	[0.30]

Actuation capabilities include operation time, base size, and base performance. Worker task W1 (Table 5.21) requires a robot of any size with at least ‘low’ actuation ability. Additionally, worker task W2 (Table 5.21) is an explorer task requiring a robot of any size with at least ‘medium’ actuation capabilities. Table 5.22 details the actuation capabilities of three potential robots with different drive types (Robot2, Robot7 and Robot8). These robots have overall actuation scores of 0.65 (‘high’), 0.48 (‘medium’) and 0.30 (‘low’), respectively.

In each task, a set of weights with a sum of unity is specified for each resource type. The processing and communication weights of tasks M1 and M2 add up to unity in Table 5.20. Similarly, the sensing and actuation weights of tasks W1 and W2 add up to unity in Table 5.21. A higher weight for a sub-resource gives it greater preference in the selection process. On the other hand, a lower weight gives reduced preference.

Table 5.23 and Table 5.24 provide details of the robot quantity criteria FIS employed by the multi-robot map-building and exploration task. Exploration area size can be precise or specified as ‘small’, ‘medium’, or ‘large’ to simplify user input. Similarly, explorer-planner task quantity ratio can be specified as ‘low’, ‘medium’ or ‘high’ if precise data cannot be given. The two outputs from the FIS are the quantities of explorers and planners for the global task.

Table 5.23 details the input and output settings of the robot quantity criteria FIS. The data ranges and membership function ranges have been empirically selected. They can be customised if required. If a human user is unable to provide precise data, the area size reference points for ‘small’, ‘medium’ and ‘large’, are arbitrarily selected as 500 m^2 (10%), 2500 m^2 (50%) and 4500 m^2 (90%), respectively. Similarly, the explorer-planner ratio reference points for ‘low’, ‘medium’, and ‘high’, are arbitrarily selected as 1 (0%), 5.5 (50%) and 10 (100%), respectively. A data range of [0 12] and a trapezoidal type ‘high’ membership function is employed for the outputs to achieve output values of ten from the FIS. Table 5.24 lists the twelve fuzzy rules that are employed by the robot quantity criteria FIS. The rules and their weights have been empirically selected.

Table 5.23: Input/output details of robot quantity criteria FIS for a multi-robot map-building task.

Input 1	Name		Area Size		
	Data Range (m^2)		[0 5000]		
	Membership Function Details	Name	small	Medium	Large
		Type	triangular	Triangular	triangular
		Parameters	[–2000 0 2000]	[500 2500 4500]	[3000 5000 7000]
Input 2	Name		Explorer-Planner Ratio		
	Data Range		[1 10]		
	Membership Function Details	Name	low	medium	High
		Type	triangular	triangular	triangular
		Parameters	[–3 1 5]	[2 5.5 9]	[6 10 14]
Output 1	Name		Explorer Quantity		
	Data Range		[0 12]		
	Membership Function Details	Name	low	medium	High
		Type	triangular	triangular	trapezoidal
		Parameters	[–4 0 4]	[1 5 9]	[6 10 12 16]
Output 2	Name		Planner Quantity		
	Data Range		[0 12]		
	Membership Function Details	Name	low	medium	High
		Type	triangular	triangular	trapezoidal
		Parameters	[–4 0 4]	[1 5 9]	[6 10 12 16]

Table 5.24: Robot quantity criteria FIS fuzzy rules for a multi-robot map-building task.

Rule No.	**Area Size**	**Explorer-Planner Ratio**	**Inputs Connection**	**Explorer Quantity**	**Planner Quantity**	**Rule Weight**
1	small	low	and	low	low	1
2	small	medium	and	low	low	0.9
3	small	medium	and	medium	low	0.1
4	small	high	and	low	low	0.75
5	small	high	and	medium	low	0.25
6	medium	low	and	medium	medium	1
7	medium	medium	and	medium	low	1
8	medium	high	and	medium	low	0.5
9	medium	high	and	high	low	0.5
10	large	low	and	high	high	1
11	large	medium	and	high	medium	1
12	large	high	and	high	low	1

The robot quantity criteria in the worker task specification (Figure 5.2, p.113) specifies an area of approximately 1600 m^2 with a 'medium' setting for the explorer-planner task ratio ([1600,medium]). A second three element data set (e.g. [1,1,4] for task W1 in Table 5.21) is derived from the robot quantity criteria FIS output. It represents the initial, minimum and maximum quantities of robots for the task. Four explorers and one planner are initially required based on the robot quantity criteria

FIS output. The minimum planner and explorer quantities are set to unity by default. For the maximum quantity, the largest initial allocation of both tasks is selected. In the map-building and exploration experiments (chapter 7), the initial, minimum and maximum values are arbitrarily selected to investigate the effect of varying the quantity of robots deployed.

The default initial resource utilisation for each worker task is set to “on”. This represents the task as being fully enabled. A minimum ([off]) and maximum ([on]) range of resource utilisation values for each task is specified by the resource utilisation mapping data. Negative one (–1) values of resource utilisation indicate that a non-periodic or event driven task is enabled. A task is disabled if its resource utilisation data is set to zero. Feedback coordination mechanism weights are omitted from Table 5.21. These weights are presented in Table 5.29 (p. 150).

(a) (b) (c)

Figure 5.11: Three mobile robots in the VUW fleet.

Pentagon shaped tricycle robots (Robot1 and Robot2) are similar to the tricycle robot Scratchy (Figure 5.11(a)) [15]. Scratchy’s capabilities are the same as Robot1. It has an AMD Athlon 64 3000+ processor with 512 MB RAM. The estimated benchmark of the processor obtained from MATLAB’s ‘bench’ function is 3.2. Scratchy has eleven 1.5 m range infrared sensors that are evenly spaced (360° distribution) around the robot. A 54 MBit/sec 802.11g (~ 100 m) wireless communication module is also present on Scratchy. For actuation, Scratchy has just over two hours of operation time and is 0.5 m in radius (when a circle is prescribed from the robot’s centre). It has a maximum speed of 0.5 m/sec on normal flat terrain with a 30% rating (0.3) for

difficult terrain traversability. Robot2 is microcontroller based with a benchmark of six and 10 MB memory. It has similar sensing and actuation capabilities as Robot1.

The circular shaped differential drive robots (Robot3, Robot5, Robot6 and Robot7) are based on the humanoid differential drive robot MARVIN (Figure 5.11(b)) [51]. MARVIN's base (lower half of Figure 5.11(b)) contains the processing, communication, sensing and actuation capabilities of the robots described in Table 5.22.

Robot3 has identical processing, communication and sensing capabilities to MARVIN's base. It has an AMD Athlon XP 2000+ processor with 256 MB RAM. Based on data from MATLAB's 'bench' function, the processor benchmark is approximately 2.1. MARVIN has ten evenly spaced (360° distribution) 1.5 m range infrared sensors and a 54 MBit/sec 802.11g (~ 100 m range) wireless module. The actuation capabilities of Robot3 include an operation time of over two hours with a base size radius of 0.4 m (similar to MARVIN). It has maximum speed of 0.5 m/sec on normal flat terrain with a low rating of 10% (0.1) for difficult terrain traversability. Unlike MARVIN (Robot3), Robot5, Robot6 and Robot7 are microcontroller based with 2–4 MB memory. These robots possess 11 MBit/sec 802.11b wireless communication with approximately 100 m range. Additionally, these robots have twelve 1.5 m range infrared sensors that are evenly distributed around the robot.

Rectangular shaped differential track robots (Robot4 and robot 8) are similar to the tank robot (Figure 5.11(c)) [137]. The tank robot has fourteen evenly spaced (360° distribution) 1.5 m range infrared sensors and a 54 MBit/sec 802.11g (~ 100 m range) wireless module .

In Table 5.22, Robot4 has fourteen 1.5 m range infrared sensors that are evenly distributed around the robot. Robot8 has sixteen infrared sensors. Unlike the tank robot, Robot4 and Robot8 have 11 MBit/sec 802.11b (~ 100 m range) wireless communication. While the tank robot possesses an AMD Athlon XP based computer for processing, Robot4 is microcontroller based with a benchmark of ten and 16 MB memory. Robot8 is also microcontroller based with a benchmark of six and 2 MB memory. For actuation, Robot4 has an operation time of over 2.5 hours and a base size radius of 0.6 m (when a circle is prescribed from the robot's centre). On normal

flat terrain Robot4 can achieve a maximum speed of 0.4 m/sec. It has a high rating for difficult terrain traversability (100% or 1). Robot8 has similar actuation capabilities to Robot4.

5.5 Task Devolution

5.5.1 Task Devolution Description

Task devolution performs two main functions. Firstly, it transfers the control of the global task from the base station to the (manager) robots. Secondly, an initial allocation of tasks to the robots is performed.

A key element of the task devolution process is the Vector of Task Suitability (VOTS). $VOTS_{ij}$ is the VOTS for a task-robot pair and represents the j^{th} robot's suitability for the i^{th} task. The VOTS data for each resource *type* $VOTS_{typeij}$ is a function of the resource capability score of the robot r_jRCS_{type} and the minimum resource capability score required for the task t_iRCS_{type} (5.4). Robot r_j is considered capable of performing task t_i if $r_jRCS_{type} \geq t_iRCS_{type}$ for all resource types.

$$VOTS_{typeij} = r_jRCS_{type} - t_iRCS_{type} \tag{5.4}$$

There are two stages in the task devolution process. At the first stage, the base station performs primary task devolution to identify the task manager robots and assign tasks to them. The task specification and robot capability details are then transferred to the task manager robots. In the second stage, the task manager robots execute a secondary task devolution process, identifying and assigning tasks to the worker robots. The primary and secondary task devolution stages are both greedy assignment processes.

Since r_jRCS_{type} and t_iRCS_{type} are unit interval data, the $VOTS_{ij}$ data of all capable robots are also unit interval. Two additional parameters, a VOTS weighted sum ($VOTSWS_{ij}$) and value (V_{ij}), are computed from the $VOTS_{ij}$ data during task devolution. An FIS (similar to section 5.3) is utilised to compute these two values. The fuzzy inference function settings are identical to those specified in Table 5.1. Table 5.25 details the settings for all the inputs and outputs of the FIS.

Four inputs to the FIS include the VOTS data for each resource type ($VOTS_{procij}$, $VOTS_{commij}$, $VOTS_{senseij}$ and $VOTS_{actij}$). These four inputs are combined with fuzzy rules and VOTS summation weights VSW_i to compute $VOTSWS_{ij}$. A fifth input, task diversity capability TD_{ij}, is also applied to compute a robot task execution value V_{ij} in the secondary task devolution process. TD_{ij} and V_{ij} are explained in steps 3 and 4 of the secondary task devolution, respectively. The set of fuzzy rules and corresponding weights employed to determine $VOTSWS_{ij}$ and V_{ij} from the five inputs is shown in Table 5.26. A unity weight value is employed for rule 15 to favour robots with high TD_{ij} values. On the other hand, lower weight values are employed for rules 13 and 14 to only slightly alter V_{ij}.

Table 5.25: $VOTSWS_{ij}$ and V_{ij} FIS input/output settings.

Inputs 1–5 & Outputs 1–2	Name		$VOTS_{procij}$, $VOTS_{commij}$, $VOTS_{senseij}$, $VOTS_{actij}$, TD_{ij}, $VOTSWS_{ij}$ and V_{ij}		
	Data Range (m)		[0 1]		
	Membership Function Details	Name	small	medium	large
		Type	triangular	triangular	triangular
		Parameters	[–0.5 0 0.5]	[0 0.5 1]	[0.5 1 1.5]

Table 5.26: Set of fuzzy rules to determine $VOTSWS_{ij}$ and V_{ij}.

Rule No.	$VOTS_{procij}$	$VOTS_{commij}$	$VOTS_{senseij}$	$VOTS_{actij}$	TD_{ij}	**Conn–ection**	$VOTSWS_{ij}$	V_{ij}	**Rule Weight**
1	low	–	–	–	–	–	low	low	vsw_{iproc}
2	medium	–	–	–	–	–	medium	medium	vsw_{iproc}
3	high	–	–	–	–	–	high	high	vsw_{iproc}
4	–	low	–	–	–	–	low	low	vsw_{icomm}
5	–	medium	–	–	–	–	medium	medium	vsw_{icomm}
6	–	high	–	–	–	–	high	high	vsw_{icomm}
7	–	–	low	–	–	–	low	low	vsw_{isense}
8	–	–	medium	–	–	–	medium	medium	vsw_{isense}
9	–	–	high	–	–	–	high	high	vsw_{isense}
10	–	–	–	low	–	–	low	low	vsw_{iact}
11	–	–	–	medium	–	–	medium	medium	vsw_{iact}
12	–	–	–	high	–	–	high	high	vsw_{iact}
13	–	–	–	–	low	–	–	low	0.1
14	–	–	–	–	medium	–	–	medium	0.2
15	–	–	–	–	high	–	–	high	1

The main steps of the primary task devolution are as follows:

1. Identify a subset of all the robots that are capable of performing at least one management task.
2. Rank the capable robots in descending order based on $VOTSWS_{ij}$ for all tasks

that they are capable of performing.

3. Consider the highest ranked robot. Determine the capability of this robot to perform combinations of management tasks that have not been assigned. A combined management task is obtained by adding the resources required for the individual management tasks. VOTS data are calculated for the combined management task to determine the robot's ability to execute the tasks.
4. Assign a combined management task to the highest ranked robot. The goal is to maximise the robot's resource utilisation such that the weighted sum of the VOTS data for the combined task approaches zero. This permits the selection of a minimal number of manager robots for the global task.
5. Remove the highest ranked robot from the selection process.
6. If all management tasks have not been assigned and all ranked robots have not been considered then go to 3.

When steps 3 to 6 are executed, all management tasks may not be assigned. The greedy nature of the algorithm tends to favour combinations of smaller tasks over individual larger tasks. To account for this imbalance, an additional iterative step has been included. The weighted sum of VOTS data for each multiple task combination is overrated by a factor *CO*. Weight w_{CO} is incremented in steps of 0.1 from 0 to 1 when all management tasks are not assigned successfully. The number of tasks in the combination is n_{1c} and the maximum number of tasks possible in a combination is n_{1cp}. Alternative task allocation strategies that employ other optimisation strategies are reviewed in chapter 2.

$$CO = 1 + w_{CO} \times \frac{n_{1c}}{n_{1cp}} \tag{5.5}$$

Stage 2 of the task devolution process, secondary task devolution is outlined below:

1. Identify a subset of robots that are able to execute each worker robot task.
2. Determine the weighted sum $VOTSWS_{ij}$, of the $VOTS_{typeij}$ data for each capable task-robot combination. This weighted sum represents a robot's load handling capacity for that task.

3. Count the number of other tasks that each capable robot can execute. Divide this value by the number distinct of worker robot task types. The resulting value represents the robot's task diversity capability TD_{ij}.
4. Determine each robot's value V_{ij} of performing each task using the $VOTSWS_{ij}$ and TD_{ij} values from steps 2 and 3 respectively.
5. For each task, sort the capable robots in descending order of value V_{ij} data.
6. Based on the number of capable robots for each task, sort the tasks in ascending order. This enables tasks with fewer capable robots to be given higher priority for allocation.
7. Store the sorted capable robots and tasks in a robot-task capability matrix for task reallocation use.
8. Using the worker task requirements, select and assign the quantity of robots needed for each type of task. The selection process checks the capability of a robot to execute the current task together with any other tasks that have already been assigned to it.
9. For each robot that is allocated a task, initialise the corresponding resource utilisation values.
10. Initialise task score values TS_{ij} for the assigned task-robot combinations (5.6). $VOTSWS_{i\max}$ is the maximum *VOTSWS* value amongst all robots that have been assigned task *i*. The task score values are updated by the feedback coordination mechanism (section 5.6).

$$TS_{ij} = \frac{VOTSWS_{ij}}{VOTSWS_{i\max}} \tag{5.6}$$

5.5.2 Multi-Robot Map Building and Exploration Task Devolution

A team of robots can be selected from Table 5.22 for the multi-robot map building task described in Table 5.20 and Table 5.21. Following the execution of the primary and secondary task devolution algorithms presented in this section, the resulting initial team and initial task allocations are shown in Table 5.27. After primary task devolution, only two of the eight robots (Robot1 and Robot3) are isolated as capable of executing at least one manager task. Robot3 is capable of executing M1, while

Robot1 can execute M2. Hence, Robot3 and Robot1 are assigned tasks M1 and M2 respectively.

Table 5.27: Resulting initial team and initial task allocations for the task and robot specifications presented in Table 5.20-Table 5.22.

Task ID	Robot ID	TS_{W1}	TS_{W2}
M1	3	–	–
M2	1	–	–
W1	4	1.0000	–
W2	4	–	1.0000
W2	2	–	0.8692
W2	5	–	0.5949
W2	7	–	0.5768

Table 5.28 details the sorted capable robots in the robot-task capability matrix. Robot8 is omitted since it does not meet the minimum resource requirements for executing any worker task. Based on the robot value data rankings, task W1 is enabled on Robot4, while Robot4, Robot2, Robot5 and Robot7 have task W2 enabled. Tasks W1 and W2 are enabled on Robot4 since it is the highest ranked robot and capable of executing both tasks. W1 and W2 are executed in parallel on Robot4. Preference is given to the planner task (W1). Each worker robot has the default initial resource utilisation of its assigned task(s) enabled after task allocation. The initial task score for each selected worker task-robot combination is given in the third and fourth columns of Table 5.27.

Table 5.28: Robot-task capability matrix for worker robots.

Task					
W1			W2		
Robot ID	*V*	*VOTSWS*	Robot ID	*V*	*VOTSWS*
4	0.6618	0.5859	4	0.4835	0.2901
2	0.5646	0.4495	2	0.4587	0.2548
5	0.5344	0.4178	5	0.4171	0.2126
7	0.5336	0.4164	7	0.4155	0.2166
			6	0.2433	0.2077

5.6 Feedback Coordination Mechanism

Feedback coordination is performed by the manager robots and executes periodically with time interval T_m after initial task allocation. If the feedback mechanism detects that the performance of a worker robot is not satisfactory, a task reallocation process

adapted from the worker task devolution is called. The task reallocation algorithm applies a set of group task specific rules that adjusts a robot's suitability for a particular task.

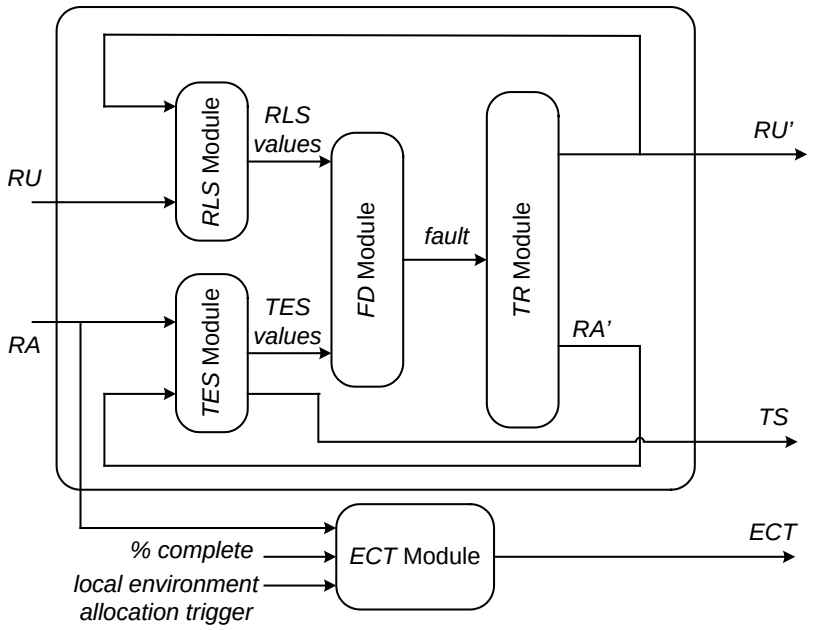


Figure 5.12: Feedback coordination mechanism block diagram.

A block diagram of the feedback coordination mechanism is shown in Figure 5.12. The current resource achievement *RA* and corresponding current resource utilisation *RU* (or task allocation) for each worker robot are input to the feedback mechanism. Each robot's *RA* data are compared with expected (best-case scenario) resource achievement *RA'* data to compute the robot's task execution success *TES* value.

The best-case scenario resource achievement can be determined from simulation data or a simple real life experiment. This would need to be carried out by an expert user initially as a form of "calibrating" the robot. A normal user is not expected to alter or provide this data. For an exploration task, the best case scenario is the exploration of an environment comprising normal terrain traversable by any robot and no obstacles. It is relatively easy for an expert user to implement a 2D simulator using MATLAB or a 3D simulator using software such as Microsoft Robotics Studio.

A robot's planning, communication, sensing and actuation achievement can be logged to data files for analysis after the global task is complete. However, robot models may

not always be accurate in simulations, and similarly simulation results do not always reflect real-world results. Thus a robot or team of robots would ideally have an initial test sequence in a simple real environment to determine ideal achievement data. This process is essential in the currently developed feedback system. If the ideal achievement data is incorrectly determined, it will affect failure detection. A good estimate is required. A future improvement to the system could be to dynamically adjust the failure detection thresholds (for poor performance, partial failures, and complete failures) in a tuning process (chapter 9).

The *RU* data of each robot are compared with expected resource utilisation *RU'* data to compute the robot's load success *RLS* value. A failure detection *FD* module then takes the *RLS* and *TES* values as input. If a failure is detected, the task reallocation *TR* module is triggered and the resource utilisation and achievement targets of the worker robots are updated.

It is assumed that the selected robots have sufficient processing resources for their task allocations. However, if a robot does not have sufficient processing resources, *RLS* data can be employed to gradually adjust resource utilisation data for improving task execution. This scenario is not investigated in this thesis. In the multi-robot exploration application (chapter 6) the feedback mechanism is also triggered to replace or remove explorer worker robots that become idle when no suitable unexplored areas remain.

5.6.1 Performance Monitoring

The performance monitoring components of the feedback coordination mechanism include the *TES*, *RLS* and *FD* modules. All performance monitoring variables are normalized to unit interval values. The resource utilisation achievement of each resource category for each task-robot combination RA_{ijcat} is specified as:

$$RA_{ijcat} = \left[ra_{ijcat1}, ra_{ijcat2}, ..., ra_{ijcats'}, ..., ra_{ijcatz'} \right] \tag{5.7}$$

where:

$$cat \in \left[plan, comm, sense, act \right],$$

$ra_{ijcats'}$ is the s'th task dependent resource utilisation achievement sub-category, and

z' is the number of sub-categories for resource utilisation achievement RA_{ijcat}.

Achievement data are a record of robot activity. As mentioned previously, resource achievement data (*RA*) are compared with expected resource achievement data (*RA'*) to determine success. Planning achievement is usually represented by the number of local and/or global plans made. Communication achievement can be represented by the volume of messages transmitted and received successfully. The accuracy of covered or explored area can represent sensing achievement. Actuation achievement may include criteria such as average speed, distance travelled or objects moved.

By integrating the instantaneous values of current and expected target achievements over N task monitoring time intervals, overall values of achievement ORA_{ijcat} and expected achievement ORA'_{ijcat} can be determined (5.8)–(5.9).

$$ORA_{ijcat} = \sum^{N} RA_{ijcat} \tag{5.8}$$

$$ORA'_{ijcat} = \sum^{N} RA'_{ijcat} \tag{5.9}$$

A set of achievement weights KA_{icat}, comprising achievement sub-category weights $ka_{icats'}$, is also specified for each task (5.10). These weights are used in conjunction with RA_{ijcat} and RA'_{ijcat} to bias sub-category achievements $ra_{ijcats'}$ and $ra'_{ijcats'}$ when determining success.

$$KA_{icat} = \left[ka_{icat1}, ka_{icat2}, ..., ka_{icats'}, ..., ka_{icatz'} \right] \tag{5.10}$$

To combine the success values of each resource category into a single *TES* value, a second set of achievement success bias weights WS_i is used (5.11). The individual robot *TES* value TES_{ijind} is determined using only the individual robot's instantaneous achievement data (5.12). On the other hand, a robot's team *TES* value TES_{ijtm} is determined as a combination of the instantaneous achievements of all n robots in the team executing a task (5.13).

$$WS_i = \left[ws_{iplan}, ws_{icomm}, ws_{isense}, ws_{iact} \right] \tag{5.11}$$

$$TES_{ijind} = \sum ws_{icat} \left(\sum ka_{icats'} \frac{ra_{ijcats'}}{ra'_{ijcats'}} \right) \tag{5.12}$$

$$TES_{ijtm} = \sum ws_{icat} \left(\sum ka_{icats'} \frac{ra_{ijcats'}}{\sum_{j=1}^{n} ra'_{ijcats'}} \right) \tag{5.13}$$

Replacing RA_{ijcat} and RA'_{ijcat} with ORA_{ijcat} and ORA'_{ijcat} respectively in (5.12), yields an overall individual robot success value $OTES_{ijind}$ (5.14). Making the same replacements in (5.13) produces a robot's overall team success value $OTES_{ijtm}$ (5.15).

$$OTES_{ijind} = \sum ws_{icat} \left(\sum ka_{icats'} \frac{ora_{ijcats'}}{ora'_{ijcats'}} \right) \tag{5.14}$$

$$OTES_{ijtm} = \sum ws_{icat} \left(\sum ka_{icats'} \frac{ora_{ijcats'}}{\sum_{j=1}^{n} ora'_{ijcats'}} \right) \tag{5.15}$$

Individual robot RLS_{ij} values are determined in a similar manner to $OTES_{ijind}$ values. The major difference is that RA_{ijcat} and RA'_{ijcat} data are replaced with RU_{ijcat} and RU'_{ijcat} data respectively (5.16). A set of resource utilisation weights KU_{icat} that combines resource utilisation data of the sub-resources for each task replaces KA_{icat} (5.17). The *RLS* data of each resource type is combined into a single number using a set of load success bias weights WL_i. Generally, the weights in WL_i and WS_i can have similar values since they are employed by the same task. *RLS* values verify that a robot has the capacity to process resource data in a timely manner.

$$RLS_{ij} = \sum wl_{icat} \left(\sum ku_{icats} \frac{ru_{ijcats}}{\sum_{j=1}^{n} ru'_{ijcats}} \right) \tag{5.16}$$

$$KU_{icat} = \left[ku_{icat1}, ku_{icat2}, ..., ku_{icats'}, ..., ku_{icatz'} \right] \tag{5.17}$$

$$WL_i = \left[wl_{iplan}, wl_{icomm}, wl_{isense}, wl_{iact} \right] \tag{5.18}$$

Calculating the *RLS* data for planning and communication resources can be troublesome since planning and communication control algorithms are usually non-periodic. However, the *TES* of planning and communication resources may be used to approximate the *RLS* values. Hence, planning achievement can be employed as an approximation of the corresponding control algorithm's execution rate. Similarly, communication achievement may be used as an indication of the communication control algorithm's execution rate.

The *FD* module detects three types of robot faults when success values are below threshold. A complete failure CF_{ij} is detected when no achievement data are received from a robot and it has no pulse signal. The pulse signal is a message sent by each robot to the manager at regular intervals to indicate activity. Partial failures PF_{ij} occur when a robot fails at its current task due to faulty hardware but is not dead (i.e. achievement data are received from the robot). Faulty hardware can include sensor or actuator (motor) failures. Partial failures can be detected when a robot's instantaneous task execution success value TES_{ijind} is below a threshold value close to zero TES_T. When a robot's overall task execution success value $OTES_{ijind}$ is below a second non-zero threshold value $OTES_T$, it is identified as a poor performance robot PP_{ij}.

$$PF_{ij} = \begin{cases} 1; & \text{if } TES_{ijind} < TES_T \\ 0; & \text{otherwise} \end{cases} \tag{5.19}$$

$$PP_{ij} = \begin{cases} 1; & \text{if } OTES_{ijind} < OTES_T \\ 0; & \text{otherwise} \end{cases} \tag{5.20}$$

The multi-robot mapping and exploration task (chapter 6) employs task scores TS_{ij} when assigning local environments to explorers and jobs to planners. Explorer task scores TS_{ej} are employed to adjust the utility values of neighbouring local environments around an explorer's current local environment. This facilitates an increment or reduction in utility values based on the performance of an explorer robot. An explorer's task score TS_{ej} is determined from the overall team success value

$OTES_{ejtm}$, where e corresponds to the index of the explorer task (5.21). Planner robot task scores TS_{pj} (5.22) are also calculated in a similar manner. The index of the planner task is represented by p in (5.22). $OTES_{etm\max}$ and $OTES_{ptm\max}$ correspond to the maximum $OTES$ values amongst all worker robots for the explorer and planner tasks respectively.

$$TS_{ej} = \frac{OTES_{ejtm}}{OTES_{etm\max}} \tag{5.21}$$

$$TS_{pj} = \frac{OTES_{pjtm}}{OTES_{ptm\max}} \tag{5.22}$$

Exploration ECT data (5.23) are computed using exploration time t_{ek}, unexplored area in the current local environment $\overline{A}_{ck}$, unexplored area in the new local environment $\overline{A}_{n'k}$, total area explored A_o and predicted exploration rate E_o. Explored area and exploration rate data are obtained from sensing achievements. ECT data are employed by the multi-robot mapping and exploration task (chapter 6) for local environment assignment. The ECT data can be computed since it is assumed that the local environments and thus global area to be explored has specified boundaries.

$$ECT_k = \begin{cases} t_{ek}(\overline{A}_{ck} + \overline{A}_{n'k})/A_o & ; \text{if } t_{ek} > T_m \\ (\overline{A}_{ck} + \overline{A}_{n'k})/E_o & ; \text{otherwise} \end{cases} \tag{5.23}$$

5.6.2 Task Reallocation

During the task reallocation process a robot's VOTS derived value in the robot-task capability matrix is updated with overall task execution success $OTES_{ijind}$ data. The robots are then re-ranked in descending order using this updated value. After re-ranking, a set of rules RL is applied to further adjust robot rankings and remove faulty robots. These rules are outlined below and listed in order of priority:

1. Remove completely failed robots in CF from all tasks in the robot-task capability matrix.
2. Delete the currently assigned task i from the robot-task capability matrix for partially failed robots in PF.

3. Shift partially failed robots in *PF* to the bottom row of the robot-task capability matrix for all other tasks.
4. Move poor performance robots in *PP* to the bottom row of the robot-task capability matrix for the currently assigned task *i*.
5. Promote robots with any task specific capability requirements to the top of the robot-task capability matrix. An example is exploring an environment with varied terrain types. In some situations a very specific actuation type may be needed to explore some local environments.
6. Adjust the quantity of tasks to be assigned based on the progress of the group task. For example, robots can gradually be removed from the team as the exploration task nears completion.

Hysteresis loops to avoid false detections due to noise are not required for complete failures (step 1) and partial failures (step 2) since reallocations are essential. In these two situations, a robot is unable to execute its allocated task and cannot be assigned the same task again. Hysteresis loops can be employed for poor performance (step 3). However, this has not been incorporated since poor performance is appropriately identified and corrected without hysteresis (section 8.5.1).

To complete the task reallocation process, steps 6 to 9 of the worker task devolution procedure (section 5.5.1) are executed to adjust the resource utilisation of the worker robots.

Partially failed and poor performance robots are given lower rankings to place them below non-faulty task capable robots. Depending on their new ranking and group task requirements, partially failed and poor performance robots may be assigned an alternative task, keep their current task (poor performance robots only) or be removed from the team.

5.6.3 Multi-Robot Map Building and Exploration Task Feedback Example

Table 5.29 – Table 5.34 detail the application of the feedback coordination mechanism in a multi-robot map building and exploration task. The worker robot tasks and robots available for the multi-robot map building and exploration task are

specified in Table 5.21 and Table 5.22 respectively. The feedback time interval T_m is set to 60 seconds.

The feedback coordination weights for worker tasks are specified in Table 5.29. Task IDs W1 and W2 represent the planner and explorer tasks respectively. In the resource achievement weights, no weights, (denoted by '–'), are required to combine the sub-categories of planning, communication and actuation achievements. While communication and actuation have only one achievement sub-category each, planning has two achievement sub-categories. The maximum value of the two planning sub-categories is selected to represent planning achievement. This can be permitted in the map building and exploration task since the two planning sub-categories are related.

Table 5.29: Feedback weights for worker tasks.

Task ID	Criteria	Value
W1	Resource Achievement Weights	
	Planning	–
	Communication	–
	Sensing	[0.5,0.5]
	Actuation	–
	Achievement Success Bias Weights	[0.5,0.5,0,0]
	TES Threshold	-1
	OTES Threshold	-1
	Resource Utilisation Weights	
	Planning	–
	Communication	–
	Sensing	[0.5,0.5]
	Actuation	[0.5,0.5]
	Load Success Bias Weights	[0.5,0.5,0,0]
W2	Resource Achievement Weights	
	Planning	–
	Communication	–
	Sensing	[0.5,0.5]
	Actuation	–
	Achievement Success Bias Weights	[0.1,0.05,0.45,0.4]
	TES Threshold	0.12
	OTES Threshold	0.65
	Resource Utilisation Weights	
	Planning	–
	Communication	–
	Sensing	[0.5,0.5]
	Actuation	[0.5,0.5]
	Load Success Bias Weights	[0,0,0.5,0.5]

Complete failure of a planner robot can be detected when no achievement data are received from the robot. Negative one (–1) threshold values in Table 5.29 indicate that poor performance and partial failures are not detected for the planning task. A partial failure of the planner task does not arise since a processing or communication failure generally renders the robot useless for other tasks in multi-robot applications.

In the explorer task, TES threshold TES_T is set to a value close to zero (0.12) such that sensing and actuation failures (or partial failure) can be detected. Poor performance in the explorer task can be detected when the overall performance is below 65% of the best-case scenario.

When combining resource utilisation data to determine robot load success values, planning and communication utilisation data correspond to planning and communication resource achievement data. Hence, no sub-category weights are required for planning and communication resource utilisation data as well. The load success bias weights favour planning and communication for the planner task. On the other hand, sensing and actuation are favoured for the explorer task's load success calculation.

Table 5.30: Instantaneous achievement data for two worker robots.

Robot ID	Achievement Category		Task ID					
			W1			W2		
	Resource	Sub-Category	*N* = 1	*N* = 2	*N* = 3	*N* = 1	*N* = 2	*N* = 3
4	Planning (n Plans)	*Global*						
		Local Environment Assignment	3	0	0	0	0	0
		Path Plan	3	0	0	0	0	0
		Local						
		Waypoint Generation	0	0	0	0	1	0
		Path Plan	0	0	0	0	3	2
	Communication (bytes)		59999	0	0	8283	4576	3192
	Sensing (n Cells)	Area Explored	0	0	0	705	339	158
		Area Explored Accurately	0	0	0	645	319	146
	Actuation	Distance Travelled (m)	0	0	0	14.58	14.17	13.42
2	Planning (n Plans)	*Global*	0	0	0			
		Local Environment Assignment	0	0	0	0	0	0
		Path Plan	0	0	0	0	0	0
		Local						
		Waypoint Generation	0	0	0	1	0	0
		Path Plan	0	0	0	3	2	3
	Communication (bytes)		0	0	0	8555	8192	0
	Sensing (n Cells)	Area Explored	0	0	0	252	317	289
		Area Explored Accurately	0	0	0	233	308	272
	Actuation	Distance Travelled (m)	0	0	0	15.18	14.6	16.6

Table 5.30, Table 5.32 and Table 5.33 illustrate the performance of two worker robots over three monitoring intervals (*N*) while exploring local environments in a large environment (chapter 6). The instantaneous achievements (actual achievement during exploration) of Robot4 and Robot2 are presented in Table 5.30. Robot4 is a planner (task ID W1) and performed global planning over the first monitor interval. Since

Robot2 is not a planner, it has zero achievement for W1. Robot4 and Robot2 are both explorers and their achievement data are listed under task W2 in Table 5.30. The best-case scenario achievement data over a single monitor period for tasks W1 and W2 are listed in Table 5.31. These data are obtained from prior simulations in a similar sized environment with normal terrain and no obstacles. By integrating the achievement data of Table 5.31 over several monitor intervals, overall expected achievement can be calculated.

Table 5.31: Instantaneous expected achievement data for two worker robots.

Robot ID	Achievement Category		Task ID	
			W1	W2
	Resource	Sub-Category	T_m	T_m
4	Planning (n Plans)	*Global*		
		Local Environment Assignment	0.4420	0
		Path Plan	0.4420	0
		Local		
		Waypoint Generation	0	0.1473
		Path Plan	0	2.8000
	Communication (bytes)		10380	5270
	Sensing (n Cells)	Area Explored	0	349
		Area Explored Accurately	0	349
	Actuation	Distance Travelled (m)	0	15.26
2	Planning (n Plans)	*Global*		
		Local Environment Assignment	0	0
		Path Plan	0	0
		Local		
		Waypoint Generation	0	0.1584
		Path Plan	0	3.0100
	Communication (bytes)			8580
	Sensing (n Cells)	Area Explored	0	382
		Area Explored Accurately	0	381
	Actuation	Distance Travelled (m)	0	18.27

Table 5.32: Instantaneous resource utilisation of two worker robots.

Robot ID	Resource Utilisation		Task ID					
			W1			W2		
	Resource	Sub-Category	$N = 1$	$N = 2$	$N = 3$	$N = 1$	$N = 2$	$N = 3$
4	Sensing	Obstacle & Pose Detection	0	0	0	8.86	9.93	9.96
	(Freq. [Hz])	Local Map Update	0	0	0	8.86	9.93	9.96
	Actuation	Motion Control	0	0	0	8.86	9.93	9.96
	(Freq. [Hz])	Motor Commands	0	0	0	8.86	9.93	9.96
2	Sensing	Obstacle & Pose Detection	0	0	0	9.7	9.96	9.78
	(Freq. [Hz])	Local Map Update	0	0	0	9.7	9.96	9.78
	Actuation	Motion Control	0	0	0	9.7	9.96	9.78
	(Freq. [Hz])	Motor Commands	0	0	0	9.7	9.96	9.78

Table 5.32 details the sensing and actuation resource utilisation for tasks W1 and W2. Since no sensing and actuation is necessary for the planner task (W1), it has zero resource utilisation for these resources. The target (expected) utilisation for sensing and actuation resources in the explorer task (W2) is 10 Hz (Table 5.21). During task execution, the actual resource utilisation for task W2 is slightly reduced due to path planning computation overheads.

Task execution success and robot load success data for Robot4 and Robot2 are given in Table 5.33. Since Robot2 is not a planner, it has zero success for task W1. However, Robot4 is a planner and has *OTES* and *RLS* values of unity over all three monitor periods for task W1. It is unsuitable to detect partial failure in the planning task using TES_{ind}. While Robot4 has zero TES_{ind} data over the second and third monitor interval, it has not failed. Instead, as suggested by the *OTES* data, Robot4 has performed extra plans over the first monitor interval. Robot4 and Robot2 are both explorers (task W2) and their success data do not indicate any partial failure or poor performance.

Table 5.33: TES and RLS data over three monitor intervals for two worker robots.

Robot ID	Parameter	Task ID					
		W1			W2		
		***N* = 1**	***N* = 2**	***N* = 3**	***N* = 1**	***N* = 2**	***N* = 3**
4	TES_{ind}	1.0000	0	0	0.9000	0.9556	0.6887
	TES_{tm}	1.0000	0	0	0.3665	0.2689	0.1678
	$OTES_{ind}$	1.0000	1.0000	1.0000	0.9000	0.9535	0.9236
	$OTES_{tm}$	1.0000	1.0000	1.0000	0.3665	0.3215	0.2729
	RLS	1.0000	1.0000	1.0000	0.8866	0.9400	0.9589
2	TES_{ind}	0	0	0	0.7675	0.8019	0.7936
	TES_{tm}	0	0	0	0.2066	0.2119	0.1931
	$OTES_{ind}$	0	0	0	0.7675	0.7847	0.7877
	$OTES_{tm}$	0	0	0	0.2066	0.2092	0.2039
	RLS	0	0	0	0.9700	0.9833	0.9816

The task score values of all worker robots over three monitor intervals are shown in Table 5.34. TS_{W2} data indicate that initial task assignment based on robot capabilities may not always rank robots in order of descending superiority. The initial team and task allocations (Table 5.27) shows that Robot2 is superior to Robot5 and Robot7 in the explorer task. Over the three monitor intervals during task execution, Robot2 is inferior to Robot7 and Robot5 in the explorer task. Task reallocation is not necessary since all robots are performing satisfactorily. The $OTES_{ind}$ values for all explorers

(task W2) is above the poor performance threshold of 0.65. Also, the TES_{ind} values for all explorers (task W2) is above the partial failure threshold of 0.12. Hence, all robots continue executing their assigned tasks.

Table 5.34: Task score data over three monitor intervals for all worker robots.

Robot ID	Task Score							
	TS_{W1}				TS_{W2}			
	Initial	$N = 1$	$N = 2$	$N = 3$	Initial	$N = 1$	$N = 2$	$N = 3$
4	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0	0	0	0	0.8692	0.5637	0.6507	0.7471
6	0	0	0	0	0.5949	0.7071	0.7371	0.7804
5	0	0	0	0	0.5768	0.6754	0.7501	0.8153

5.7 Scalability of Task Allocation and Feedback Coordination

Task allocation consists of two stages. These are primary task devolution and secondary task devolution. Both processes are centralised and execute sequentially. After the selection of a manager robot from the primary stage, all information regarding the available robots is forwarded to it from the base station. Using the robot capability data presented in Table 5.22 with numerical values for robot IDs and drive types, each robot's description is approximately 112 bytes in double precision. Latency is expected to be small (≤ 0.5 sec) when information for up to thirty robots is transmitted using 802.11g wireless communication.

Secondary task allocation utilises a centralised greedy algorithm. This is expected to function appropriately if the number of robots is increased. More complex meta heuristic algorithms, such as genetic algorithms, Tabu search, branch and bound, or pattern search [138] could be utilised to search the space of robot fleet permutations. However, these methods are likely to have greater search times and still do not guarantee an optimal selection. Moreover, these methods would not produce adequate solutions if tasks are incorrectly specified by a non-expert human user. This research favours the use of a feedback system to detect and correct suboptimal situations during task execution.

Feedback coordination is periodic and requires worker robots to send achievement data to managers. Chapter 8 evaluates feedback coordination at monitor time intervals

of 60 sec, 180 sec and 300 sec. Using the achievement data presented in Table 5.30, the feedback data sent by each robot will be approximately 64 bytes in double precision. Assuming the worker robots employ 802.11b wireless devices, a maximum bandwidth of 1408 KB/sec is available. Hence, the volume of feedback data is small relative to the bandwidth of communication. Up to 22528 robots can be monitored. However, in reality worker robots will be executing tasks and sharing other data as well. Chapter 6 evaluates the scalability for a multi-robot mapping and exploration task.

5.8 Summary

This chapter has presented a task allocation and feedback coordination mechanism for a hierarchical heterogeneous multi-robot system that consists of limited capability mobile robots. By employing fuzzy inference systems (FISs), the system can accept simplified human user data (graded inputs) for task specification. This enables non-expert human users to specify tasks to a team of robots. However, such an approach requires additional effort from an expert in designing the system. The use of graded inputs is subjective and a non-expert human user may need to be educated via user manuals. Task allocation matches the resources required for a task with resources available on prospective robots.

By employing feedback coordination, the system has the potential to address sub-optimal task allocations and robot failures. Feedback coordination can also be used to rectify errors in tasks specified by non-expert users. Ideal (best-case scenario) achievement data is required to benchmark the performance of robots during task execution. Hence, an expert user is required to “calibrate” a robot’s performance prior to task execution. It is envisioned that this calibration will only be required to be done when the task type is altered.

Data presented in this chapter demonstrates the application of the task allocation and feedback coordination mechanism to a multi-robot map building and exploration task. Results of extensive experiments with the task allocation and feedback coordination applied to a multi-robot map building and exploration task (chapter 6) are presented in chapter 7 and chapter 8.

6 Multi-Robot Map Building and Exploration Task

6.1 Overview

In multi-robot systems comprising robots with limited processing and sensing capabilities, mapping a large environment may be performed efficiently with good task allocation and coordination strategies [132]. Chapter 5 presented a task allocation and feedback coordination mechanism that can be employed for limited capability robots. This chapter focuses on a hierarchical multi-robot map building and exploration strategy that utilises the task allocation and feedback mechanism of chapter 5. The strategy enables limited capability (in terms of processing capability or memory capacity) robots to map a large environment that may contain numerous scattered obstacles. Since this is a customised map building and exploration strategy that is not similar to other approaches, it is difficult to compare it with the techniques reviewed in section 2.9.

A global environment is divided into local environments (similar to chapter 4) for the limited capability robots (worker robots as described in section 5.3) to explore (Figure 6.1). This approach is an example of territorial division [139, 140] where separate areas are assigned to each robot for performing tasks. The division is based on the sensing and processing capabilities of the robots in addition to their physical size. By storing only a local map, robots with limited processing and sensing abilities can update map data while surveying the local environment. Territorial division also has the potential advantage of decreasing robot interference during task execution [140].

Two types of tasks are performed by the limited capability robots. Firstly, a planner task enables robots to navigate between the local environments. Secondly, to navigate and explore within a local environment, an explorer task is required. Ideally a robot would be able to perform both of these tasks. However, due to (say) processing or memory limitations it may only be able to execute one in real time. Hence, the multi-robot map building and exploration task is expressed as multi-task robots performing single-robot tasks. The planning and exploring tasks of the limited robots can be

coordinated with the assistance of a computationally powerful manager robot (Figure 6.2).

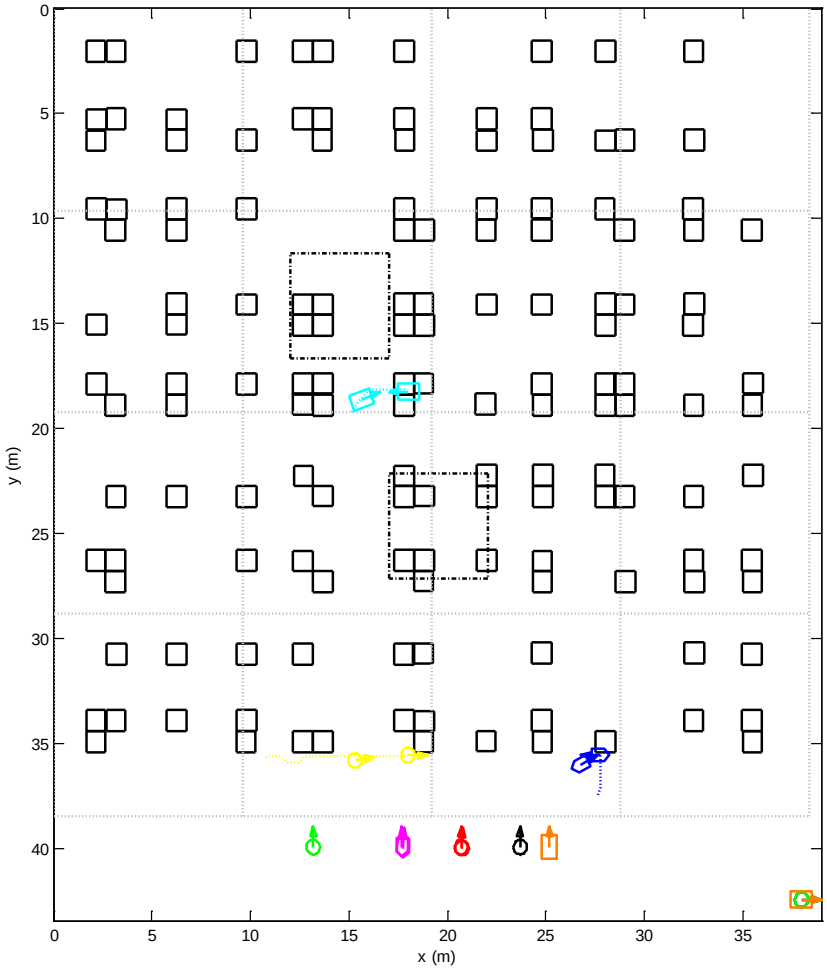


Figure 6.1: A team of limited capability robots exploring a large environment.

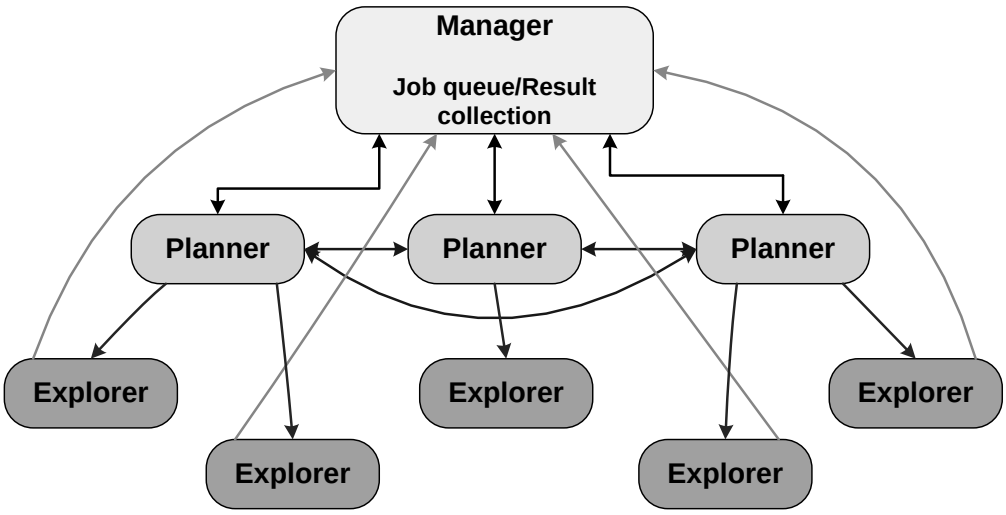


Figure 6.2: Multi-robot map building and exploration task overview.

A major role played by the manager robot is the maintenance of the map-building task's global information. It maintains a queue of explorer requests for new local environments and updates the status of the local environments as the global task progresses. Planners rely on the manager for path planning data and new local environment allocation requests. The planners can communicate amongst themselves to address concurrency issues. Asynchronous two-way communication is assumed to exist between the manager, planners and explorers. Messages are received via callback functions.

Figure 6.1 shows an example of a global environment divided into local environments in a MATLAB simulation of the multi-robot map building and exploration task. There are sixteen local environments and their boundaries are represented by grey dotted lines. Three limited capability robots (light blue, blue and yellow in Figure 6.1) are employed to explore the environment. Scattered obstacles present in the environment are represented by black squares. Also present in the environment are regions of boggy terrain whose boundaries are denoted by the black dashed-dotted lines. Only robots with a particular drive type, such as caterpillar-like treads, can navigate through boggy terrain. For clarity of explanation, the environment is characterised as either being flat and easily accessible (normal) or difficult to traverse (for example bogs, rubble, steep slopes). Throughout this chapter, such difficult terrain will just be labelled as boggy.

A complementary pair of global maps stored on the manager robot represents obstacle and terrain information. The obstacle map consists of occupancy probability data and is stored at the required resolution (section 3.2). Data in the obstacle map represent obstacles independent of terrain.

Unlike obstacle data, terrain data represents terrain traversable by any robot and difficult terrain. In this thesis, terrain traversable by any robot is represented as non-boggy terrain while boggy terrain represents difficult terrain. A value of zero represents non-boggy terrain whereas boggy terrain is represented by value of unity.

This thesis arbitrarily assumes that the area of a bog is not insignificant in comparison to the area of a local environment. Under this assumption terrain data are generalised

for a local map. Hence, when part of a local map consists of a bog, the entire local map is marked as boggy. This assumption is a function of available robot memory and the division of the global environment into local environments. Marking the entire local environment as boggy will save memory as it can be problematic to maintain multiple global maps at the required resolution (obstacle occupancy grid map resolution) for limited robots. If the sizes of bogs are small in comparison to the local environment, the terrain map can be maintained at full resolution (similar to the obstacle map). This permits superior paths to be planned at the expense of additional memory.

Initially, it is assumed that all local environments consist of non-boggy terrain. However, as boggy terrain is encountered by explorers during exploration, the terrain data are updated to a value of unity. Explorers communicate the discovery of bogs to managers and this information is shared with the planner robots.

A multi-robot map building and exploration task has been simulated using MATLAB. Results of performing this multi-robot task with and without the feedback coordination system (section 5.4) are presented in chapter 8.

6.2 Explorer Task

As previously mentioned, there are two major functions (or sub-tasks) carried out by the explorers. One function is navigation to an assigned local environment. The other function is to explore and map the assigned local environment. Figure 6.3 illustrates the explorer control sequence and the relationship between the two major sub-tasks. Initially, the explorer requests for a new exploration area and/or path. It then receives new instructions from a planner about where to go and explore. Following this, the two major sub-tasks are sequentially executed to achieve the objectives of the explorer task.

Navigation to an assigned local environment involves travelling along a planned path provided by a planner robot (Figure 6.3 and Figure 6.6) from the current location to the centre of the assigned local environment. An explorer may travel through partially explored regions of the global environment to reach an unexplored local environment. Therefore, it cannot be assumed that any edge or corner of the local environment is

better than others. Planning to the centre of the assigned local environment permits equal entry opportunities from all edges or corners. Navigation to an assigned local environment is complete once the explorer is inside its assigned local environment. The navigation strategy employed is described in chapter 3.

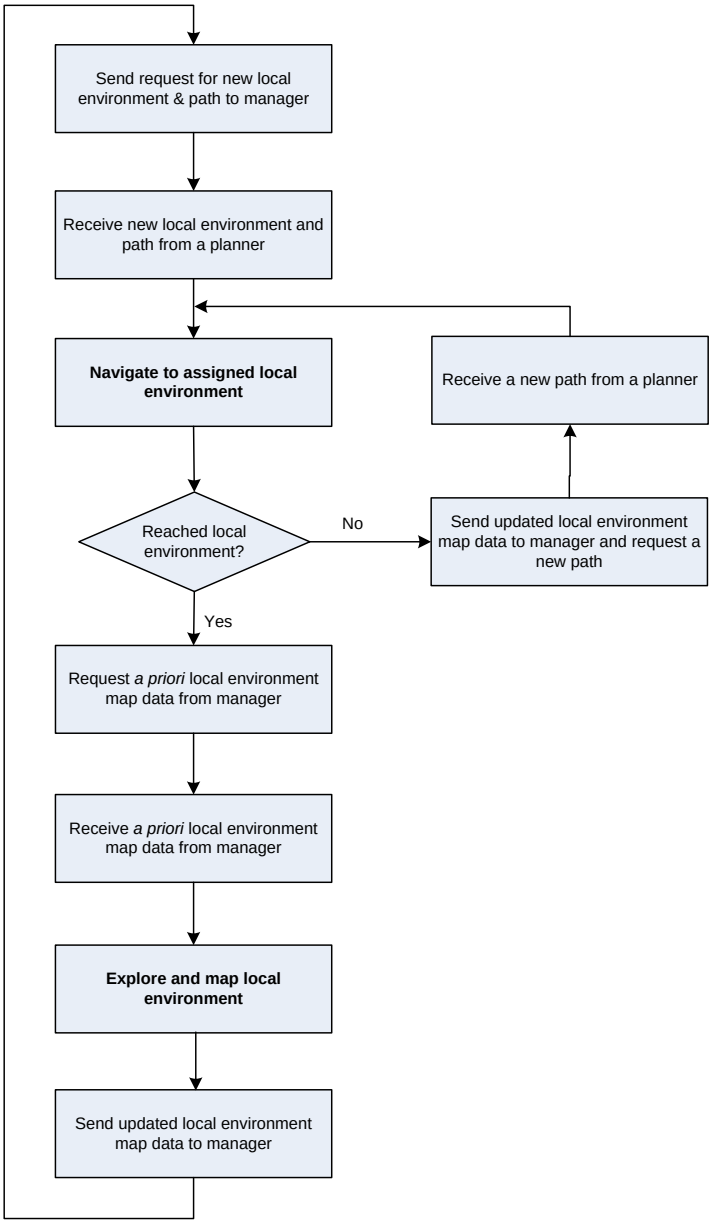


Figure 6.3: Explorer control flowchart.

While navigating to its assigned local environment, the explorer will pass through other local environments. Some of these local environments may be unexplored or partially explored. Hence, explorers maintain a small quantity of local map memories and send updated map information to the managers as navigation to the assigned local

environment progresses. Exploration and mapping of the assigned local environment is discussed in the following sections.

6.2.1 Exploring the Assigned Local Environment

The robots employed for exploration have limited range sensors mounted at fixed positions and orientations. For initial simulation purposes, straight beam sensors such as infrared sensors are assumed for obstacle detection since they are in common use, being inexpensive and reasonably accurate depending on the grid map resolution. Cone-shaped beams such as ultrasonics could equally be modelled, and are briefly discussed at the end of section 6.2.2.

However, employing infrared sensors requires a robot to traverse most of the environment to update map data. To aid traversal (hence exploration) of the assigned local environment, a set of waypoints is generated (Figure 6.4). The waypoint generation technique is inspired by the trapezoidal decomposition coverage method [141]. Waypoint generation accounts for a robot's sensing characteristics (range and noise) and the size of the local environment. Sensor noise is assumed to be linearly proportional to range.

The first two blocks of Figure 6.4 get the dimensions of the local environment and identify the corner from which the waypoint list begins. The next step for generating waypoints involves determining a threshold sensor noise NS_t from an expected occupancy probability threshold value P_s and map resolution parameters (X_{res},Y_{res}) (6.1). This threshold sensor noise is converted to a threshold sensing range SR_t by employing the minimum sensor noise NS_{min}, maximum sensor noise NS_{max}, minimum sensing range SR_{min} and maximum sensing range SR_{max} characteristics (6.2).

$$NS_t = \frac{0.5 X_{res} Y_{res} (1 - P_s)}{2 P_s (X_{res} + Y_{res})} \tag{6.1}$$

$$SR_t = SR_{\min} + (SR_{\max} - SR_{\min}) \frac{NS_t - NS_{\min}}{NS_{\max} - NS_{\min}} \tag{6.2}$$

Using the information of the first three blocks (Figure 6.4), a minimum number of evenly spaced lines required to scan the environment is determined. The start and end

points of these evenly spaced lines are ordered into a set of waypoints. A graphical layout of an ordered set of eight waypoints for a local environment is presented in Figure 6.5. Generally, the sensing radius threshold SR_t calculation is only necessary for the first local environment explored and its value is retained by the robot for subsequent local environment exploration. If the expected occupancy probability threshold P_s is set to a different value then the sensing range threshold is recalculated.

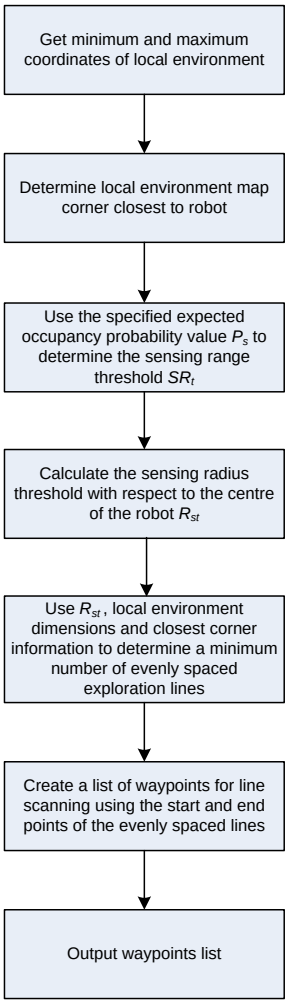


Figure 6.4: Waypoint generation flowchart.

After the waypoints have been generated and ordered into a list, they must be sequentially visited. The navigation strategy described in chapter 3 is utilised to achieve waypoint navigation. Path planning favours unexplored space while the local environment map data are updated at regular intervals. As each waypoint is successfully visited, the target is updated to the next waypoint on the list. If the progress towards a waypoint is unsatisfactory, path replanning occurs initially.

However, persistent unsatisfactory progress such as poor relative progress since the last replan or circling the goal location, results in the waypoint being considered inaccessible. This may occur if the waypoint is located inside a previously unmapped obstacle or within a narrow passage that was previously unknown. In such situations, the target is updated to the next waypoint on the list. When the last waypoint on the list has been visited or is deemed inaccessible, the exploration is complete and map data are forwarded to the manager for evaluation (section 6.4.2).

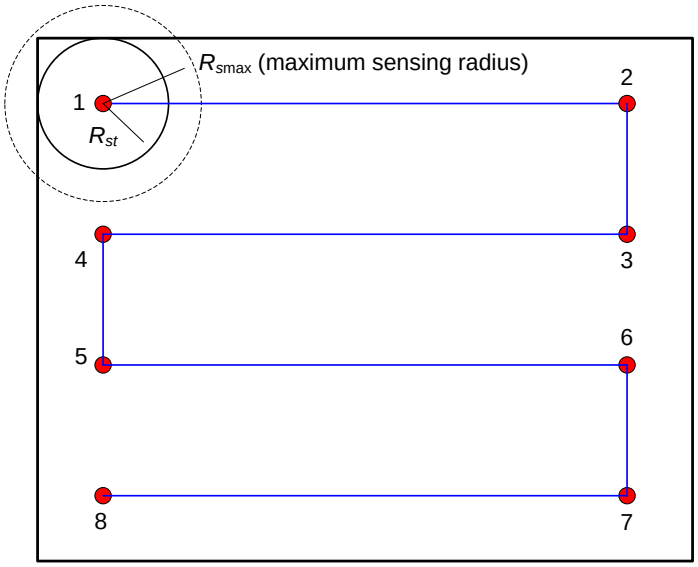


Figure 6.5: Graphical layout of waypoints.

Assuming the global environment has reasonably flat terrain, the robots employed for exploration are capable of detecting boggy terrain by monitoring the PWM control effort applied to their drive motors. If the PWM control effort is abnormally high at low velocities, then it is highly likely that a non-bog capable robot is in boggy terrain. In this situation, the explorer informs the manager to update the local environment as boggy. To attempt to get out of the bog, a non-bog capable robot reverses to the first node of its current path (i.e. the path the robot followed to enter the bog). Once successfully out of the bog, the non-bog capable explorer can request for an alternative non-boggy local environment to explore. If the robot is unsuccessful in getting out of the bog, the feedback mechanism (section 5.4) will detect robot failure.

If a robot is capable of traversing boggy terrain, it is assumed that a reduction in robot travel speed occurs in the absence of other obstacles when passing through a bog. An increase in the drive motor PWM control effort at this reduced speed is also expected.

In this situation, the bog capable explorer informs the manager to update the local environment as boggy and continues to explore the local environment.

6.2.2 Mapping the Assigned Local Environment

Each node in the local environment's occupancy grid map is assigned an occupancy probability value in the range [0.05,0.95] (section 3.2.1). Initially, each node within a previously unassigned local environment has an occupancy value of 0.5 to represent unexplored space.

Uncertainty in the robot's position is not modelled. For the purpose of this discussion it is assumed that the robots are able to localise themselves. However, imprecise robot localisation due to sensor uncertainty would cause cells of the occupancy grid map to be updated incorrectly resulting in an inaccurate map. This in turn presents difficulties for navigating to unexplored regions using the map data. Simultaneous localisation and mapping (SLAM) can be utilised to correct this problem [67]. However, this may present difficulties due to limited sensing, computing or memory on the robots. Instead, navigation aids such as beacons or GPS (DGPS) can be employed to correct imprecise localisation [67].

Since the infrared sensor beams are directional, they are simulated as a straight line originating from the sensor up to the maximum sensing range. As an example, the Sharp GP2Y0A02YK infrared sensors have a beam angle of approximately 5 degrees. The beam width is approximately 13 cm at the maximum range of 1.5 m. This is small in comparison to the occupancy grid resolution of 30 cm (section 3.2.1). Nodes that the sensor beams pass through need to have their probabilities updated as free space. Nodes that the robot itself occupies are also updated as free space. If an obstacle is detected by the sensor beam then the corresponding nodes are updated as occupied space.

Occupancy probabilities are updated in real time using Bayes' rule [130]. This rule produces an updated (or posterior) occupancy probability P_u using an initial (or prior) occupancy probability P_i and recently acquired sensor information (6.5). The new sensor information is translated into an expected probability that a sensed occupied

cell is occupied $P(occ_s|occ)$ or a sensed unoccupied cell is unoccupied $P(uocc_s|uocc)$. As it is only possible for a cell to be sensed as either occupied or unoccupied, only one of the two probabilities is employed in the occupancy probability update.

$P(occ_s|occ)$ data are approximated by employing the grid map resolution parameters (X_{res},Y_{res}) and sensor noise NS_{sen} to compute a ratio of areas (6.3). The numerator of (6.3) corresponds to the area of a single occupancy grid while the denominator is an enlarged area accounting for sensor uncertainty. $P(uocc_s|uocc)$ is approximated by a linear probability function that varies between 0.2 (at the minimum sensing range $SR_{\min}$) and 0.5 (at the maximum sensing range $SR_{\max}$) (6.4). The lower probability limit of 0.2 has been arbitrarily selected to control the free space probability saturation rate. On the other hand, the upper limit of 0.5 results in the prior and posterior probabilities being the same. Equation (6.4) thus permits cells closer to the robot to saturate faster towards the minimum occupancy probability value of 0.05 representing free space (section 3.2).

$$P(occ_s \mid occ) = \frac{X_{res}Y_{res}}{X_{res}Y_{res} + 2NS_{sen}(X_{res} + Y_{res})} \tag{6.3}$$

$$P(uocc_s \mid uocc) = 0.2 + \frac{(0.5-0.2)(SR - SR_{\min})}{SR_{\max} - SR_{\min}} \tag{6.4}$$

$$P_u = \begin{cases} \dfrac{P_i P(occ_s \mid occ)}{P_i P(occ_s \mid occ) + (1-P_i)(1-P(occ_s \mid occ))} & \text{if } P(occ_s \mid occ) > 0.5 \\ \dfrac{P_i P(uocc_s \mid uocc)}{P_i P(uocc_s \mid uocc) + (1-P_i)(1-P(uocc_s \mid uocc))} & \text{if } P(uocc_s \mid uocc) < 0.5 \\ P_i & \text{otherwise} \end{cases} \tag{6.5}$$

If boggy terrain is detected during mapping and the explorer is non-bog capable, all cells that the explorer's sensors sweep through while inside the bog are reset to unexplored. This is done to prohibit global path planning through the bog when the non-bog capable explorer has successfully exited the bog and is assigned an alternative (non-boggy) local environment. It is necessary since the explorer is already within a local environment marked as boggy and needs to exit it successfully to reach

non-bogy terrain. The cells can be updated later when a bog capable robot is assigned to explore the local environment.

The node occupancy probability update method can be extended to sensors with cone-like beams such as ultrasonic sensors. The sensor beam simulation representation can be changed from a straight line to a triangular shaped approximation for a 2D simulation. Additionally, the sensor noise function and its relationship to the expected probabilities ($P(occ_s|occ)$ and $P(uocc_s|uocc)$) within the triangular sensing region may be different.

6.3 Planner Task

Similar to the explorers, the planners execute two major functions (or sub-tasks). One of these functions is to assign local environments to explorers (section 6.3.1). The other major function is to perform global path planning for explorers. Figure 6.6 shows the relationship of the two major sub-tasks in the planner control sequence.

Once the manager has a job such as an explorer's request for a new local environment assignment and/or path in the queue (section 6.4), it delegates the job to an available planner. With the assistance of the manager, planners maintain the status of each local environment (section 6.4.2). These status values are employed by the local environment assignment and global path planning sub-tasks. Hence, before executing the two sub-tasks, the planner checks its received messages for status updates from other planners. If communication fails, the planners act independently and it is possible for a local environment to be allocated to multiple robots simultaneously. However, a feedback coordination mechanism (section 5.6) can identify and remove or replace faulty (duplicate) planners.

If a job does not require a new local environment assignment for an explorer, a path is planned to its currently assigned local environment. However, if an explorer requires a new local environment assignment, a new local environment assignment is proposed by a planner (section 6.3.1). Before addressing any concurrency issues, the planner requests an estimated completion time (ECT) acceptability notification from the

manager (section 6.4.3). ECT is only computed when less available local environments remain than explorers.

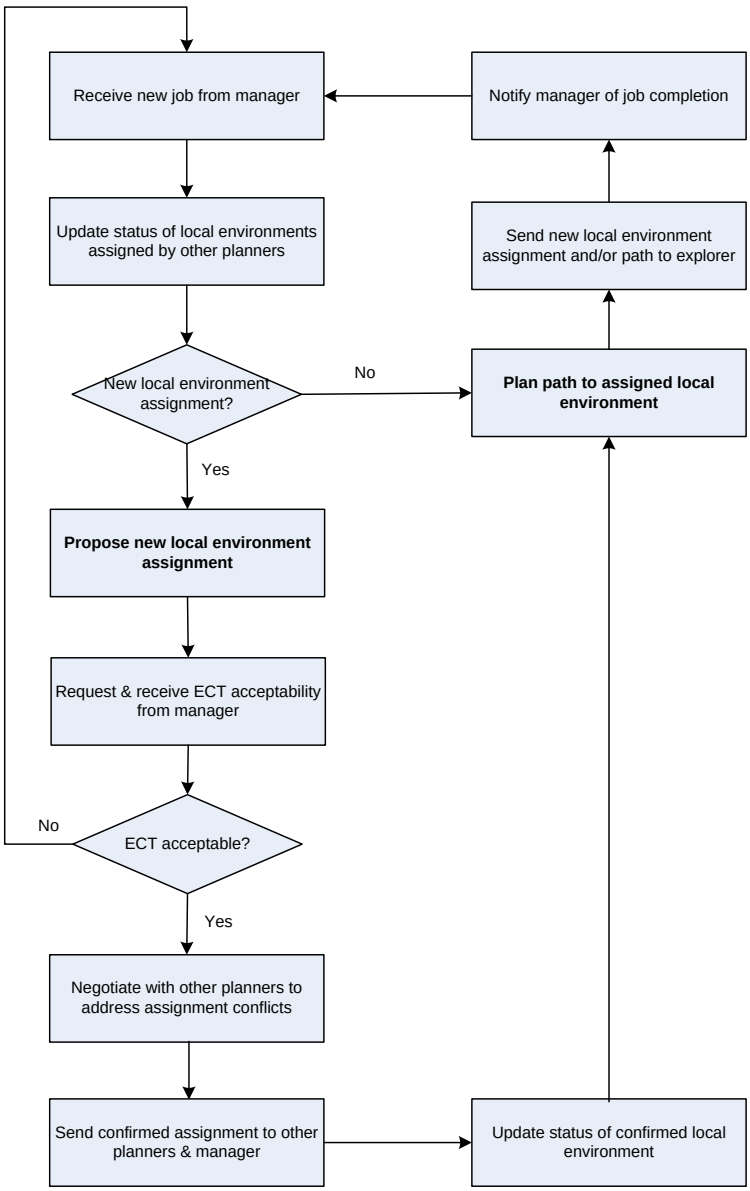


Figure 6.6: Planner control flowchart.

A planner needs to negotiate with other planners before confirming its proposed local environment assignment. This involves addressing concurrency issues such as assigning the same local environment to multiple explorers. To address concurrency issues, the planner requests other planners to provide their proposed assignments. A distributed mutual exclusion strategy [142] is applied to proposed assignments with identical local environments. Initially, proposed assignment(s) with the minimum

utility-cost tradeoff are isolated. Then, if there are more than one minimum tradeoff proposed assignments, the one with the lowest job ID is selected. This ensures that only one explorer is assigned to the conflicting local environment. After resolving any conflicts, a confirmed new local environment assignment is sent to other planners and the managers. Prior to global path planning, the status of the confirmed local environment is updated as assigned.

If the job does not require a new local environment assignment for the explorer, or if a confirmed assignment has been made for the explorer, a global path is planned to the assigned local environment. Global path planning involves planning a path from the explorer's current position to the centre of its newly assigned local environment. Planners employ the memory constrained path planning technique presented in chapter 4 to facilitate global path planning for explorers. If a particular local environment's status indicates that it has been assigned to another explorer, the path planner avoids planning through that local environment to minimise interference between robots. Additionally, the path planner employs terrain information to avoid planning paths through boggy local environments for non-bog capable explorers.

6.3.1 Local Environment Assignment

Planner robots employ a greedy algorithm inspired by [11] to propose a new local environment assignment for an explorer (Figure 6.7). The proposed new local environment assignment is checked for ECT acceptability and other planner conflicts (Figure 6.6) prior to confirmation. Initially, the planner obtains an update on the current local environment assignment of each explorer from the manager. Next, if the explorer requiring a new local environment is capable of exploring boggy terrain (determined from a robot's actuation capability data (chapter 5)), a set of all available boggy local environments is determined. This set of available local environments consists of all unexplored or partially explored boggy local environments.

Utility and cost values are computed for each local environment in the set of available environments for each explorer. The utility of a local environment is a dimensionless value that represents the gain from exploring the local environment. In this application it represents the unexplored space within a local environment. The cost of

a local environment is expressed in terms of travel distance and proximity to the other explorer robots. Utility and cost values are explained in some detail below.

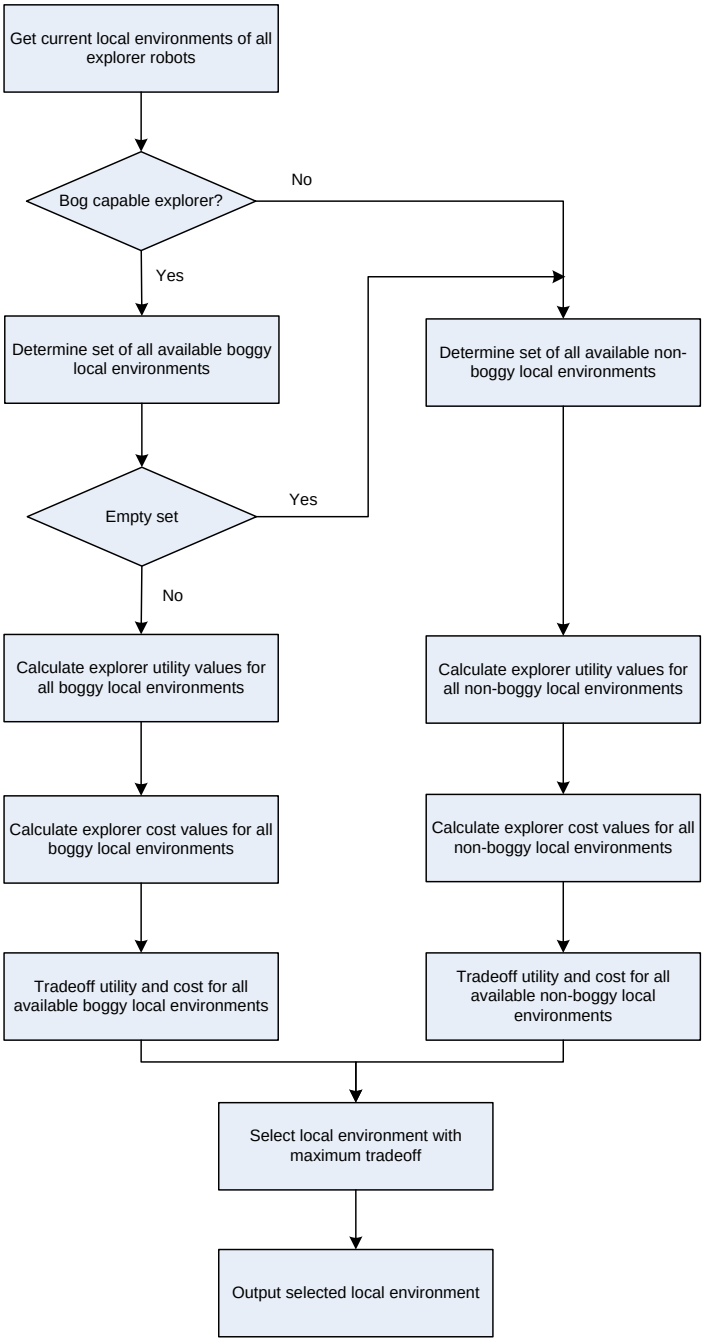


Figure 6.7: Flowchart to propose a new local environment assignment for an explorer.

Utility values are computed for each available boggy local environment if the set of available boggy local environments is non-empty. Each explorer – local environment

combination (*ej–lm*) has a utility value $U_{ej–lm}$. All initial utility values ($U_{ej–lm\text{-}init}$) are set to unity since all local environments are assumed to be unexplored (6.6).

$$U_{ej-lm-init} = 1 \tag{6.6}$$

A utility factor $UF_{ej\text{-}lm}$ influences the utility values of local environments adjacent to the current local environment assignments of the explorers (6.7). This parameter enables weaker explorers (slow robots) to favour local environments adjacent or close to their current location. Non-zero unit interval explorer task scores TS_{ej} (section 5.6.1) determine $UF_{ej\text{-}lm}$ (6.8). The *limit* function (6.7) maintains utility values in the range [0,2]. This facilitates adequate tradeoff between utility and unit interval cost values (6.10). If the utility of an adjacent local environment becomes zero due to utility factor reduction or being completely explored then it is less favoured (Table 6.1). However, if the utility of an adjacent local environment is two, it is highly favoured for exploration (Table 6.1).

$$U_{ej-lm} = limit(UF_{ej-lm}.U_{ej-lm-init}) \tag{6.7}$$

$$UF_{ej-lm} = \begin{cases} \dfrac{1}{TS_{ej}} & \text{; for explorer } j\text{'s neighour local environments} \\ TS_{ej'} & \text{; for other explorers } (j') \text{ neighbour local environments if } TS_{ej} \geq TS_{ej'} \\ 1 & \text{; otherwise} \end{cases} \tag{6.8}$$

After computing utility values for the local environments, corresponding cost values need to be determined. Each explorer – local environment combination (*ej–lm*) has a unit interval cost value $C_{ej–lm}$ (6.9). The cost function weighs the distance of the explorer to an available local environment $dist_{ej\text{-}lm}$ and the mean distance of the available local environment to other explorers $\overline{dist_{ej'\text{-}lm}}$. Distance $dist_{ej\text{-}lm}$ is minimised while $\overline{dist_{ej'\text{-}lm}}$ is maximised. Unit interval cost values are obtained by employing the maximum distance values (max($dist_{ej\text{-}lm}$) and max($\overline{dist_{ej'\text{-}lm}}$)) together with weights ($w_{ej\text{-}lm\text{-}1}$ and $w_{ej\text{-}lm\text{-}2}$) in the computation (6.9).

$$C_{ej-lm} = w_{ej-lm-1} \cdot \frac{dist_{ej-lm}}{\max(dist_{ej-lm})} + w_{ej-lm-2} \cdot \frac{\max(\overline{dist_{ej'-lm}}) - \overline{dist_{ej'-lm}}}{\max(\overline{dist_{ej'-lm}})} \quad (6.9)$$

Weight $w_{ej\text{-}lm\text{-}1}$ favours minimising the distance of the available local environment from the explorer robot. On the other hand, weight $w_{ej\text{-}lm\text{-}2}$ favours maximising the mean distance of the available local environment from other explorer robots. A value of 0.9 and 0.1 has been empirically determined for $w_{ej\text{-}lm\text{-}1}$ and $w_{ej\text{-}lm\text{-}2}$ respectively. These values provide adequate balance between the two cost components $dist_{ej\text{-}lm}$ and $\overline{dist_{ej'\text{-}lm}}$. If $w_{ej\text{-}lm\text{-}1}$ is reduced and $w_{ej\text{-}lm\text{-}2}$ is increased then the cost calculation can favour the maximisation of distance between explorers. However, minimising the distance of travel to an available local environment is preferred in the exploration task. This avoids excessive travel through unexplored terrain to reach a new local environment.

Once the utility and cost values have been determined, a tradeoff value for each available local environment can be computed. Each explorer – local environment combination (*ej–lm*) has a tradeoff value T_{ej-lm} (6.10). An available local environment with maximum tradeoff is selected as the proposed local environment for each explorer. Table 6.1 details the range of tradeoff values as utility and cost are varied.

$$T_{ej-lm} = U_{ej-lm} - C_{ej-lm} \quad (6.10)$$

Table 6.1: Utility and cost combination tradeoff data range.

U_{ej-lm} Value	C_{ej-lm} Value	T_{ej-lm} Range
2	(0,1]	[1,2)
(1,2)	(0,1]	(0,2)
1	(0,1]	[0,1)
(0,1)	(0,1]	[-1,1)
0	(0,1]	[-1,0)

If the set of available boggy environments is empty, a set of all available non-boggy local environments is determined. Similarly, if the explorer is not capable of exploring boggy terrain, the set of all available non-boggy local environments is also determined. All unexplored or partially explored non-boggy local environments are contained in the set of available local environments. At this stage of planning, the set of non-boggy local environments is not empty since the manager always checks this

condition before sending a job to a planner. The utility and cost values of the available non-boggy local environments are calculated in a similar manner to the procedure described above ((6.6)–(6.9)). Each non-boggy local environment's tradeoff value is also computed as described previously (6.10), and the local environment with maximum tradeoff is selected as the proposed assignment.

6.4 Manager Responsibilities

The primary purpose of the manager robot is to coordinate the planning and exploring activities of the limited capability robots. It initially assigns planner and explorer tasks to the limited robots, and then monitors their performance while the map-building and exploration task executes (chapter 5). During task execution, the manager also maintains global information related to the map-building and exploration task.

While the manager robot is computationally more powerful than the explorers, it may not be able to execute all the functions required for successful coordination. Hence, there may be multiple managers sharing the workload. However, in this chapter it is assumed (to simplify the system explanation) that the maintenance of the map-building task's global information can be handled by a single manager robot. If a single manager is unable to execute the global information maintenance task, it can be subdivided and executed on multiple robots that communicate with one another.

There are three major components of global information maintenance for the multi-robot map-building and exploration task: job queue maintenance, global map data and local environment status updates, and estimated completion time (ECT) computation. Each of these components is presented in some detail below.

6.4.1 Job Queue Maintenance

A single queue is maintained by the manager to receive job requests from explorers and forward these requests to planners for processing. Maintaining a single queue avoids job duplication. The manager responds to messages and data received from explorers and planners. Initially, an unassigned job list *UJL* is empty (Figure 6.8(b)) and the manager waits for a job to arrive from an explorer (Figure 6.8(a)).

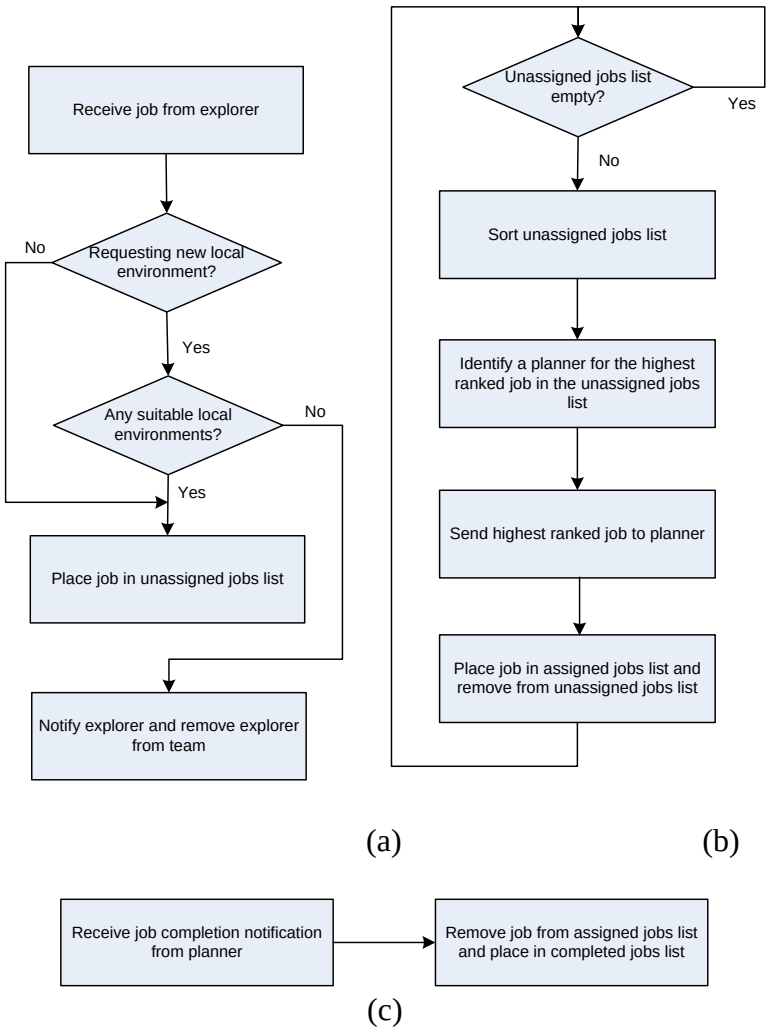


Figure 6.8: Job queue maintenance flowcharts.

The job received from an explorer JB_{ej} consists of the explorer's ID j and task type TT_j (6.11). Task type TT_j is a request for either a new path to an assigned local environment or a new local environment assignment and path. If explorer j is requesting a new local environment assignment and there are suitable local environments remaining, a unique task ID TI is merged with JB_{ej} to produce manager task M_{TI} (6.12). M_{TI} is then appended to UJL (6.13). However, if there are no suitable local environments remaining to explore, explorer j is notified and removed from the team by the feedback mechanism (section 5.6). If explorer j is requesting a new path then its job is appended to UJL.

$$JB_{ej} = \{j, TT_j\} \tag{6.11}$$

$$M_{TI} = \{TI, JB_{ej}\} \tag{6.12}$$

$$UJL = \{UJL; M_{TI}\} \tag{6.13}$$

It is possible for several explorers to send a job to the manager simultaneously. Hence, there can be several jobs in *UJL* before a job is forwarded to a planner for processing. Instead of forwarding jobs to planners in a first in first out (FIFO) manner, *UJL* is sorted to enable the most suitable explorers to have their jobs processed first.

Initially, the jobs in *UJL* are already ranked in ascending order of task ID. Next, the jobs in *UJL* are sorted into new local environment assignment jobs and path plan jobs. Priority is given to jobs requiring a new local environment assignment to reduce the number of idle robots during exploration. Following this, the new local environment assignment jobs are sorted in ascending order based on explorer task scores TS_{ej} (section 5.6.1). If there are more local environments available for exploration than new local environment assignment jobs, no further processing is necessary. However, if there are more new local environment assignment jobs than local environments available for exploration, the weakest explorer(s) assignment jobs are purged from *UJL*. The weakest explorer(s) are notified that their exploration task is complete and they are removed from the team by the feedback coordination mechanism (section 5.6).

After sorting and truncating *UJL*, the highest ranked job in *UJL*, M^*_{TT}, needs to be delegated to a planner for processing. All available planners (i.e. planners not processing any jobs) are ranked in descending order based on planner scores TS_{pj}. The highest ranked planner is then delegated M^*_{TT}. An initially empty assigned jobs list *AJL* has M^*_{TT} appended to it (6.14). Following this, M^*_{TT} is purged from *UJL*.

$$AJL = \{AJL; M^*_{TT}\} \tag{6.14}$$

After completing its delegated job, the planner sends a job completion notification to the manager. When the notification is received by the manager, M^*_{TT} is removed from *AJL* and placed in the completed jobs list *CJL* (Figure 6.8(c)).

During exploration, explorers can suffer failures in sensing, actuation, processing and communication. These failures can be detected by the feedback coordination

mechanism (section 5.6) and the exploration task can be updated accordingly (Figure 6.9(a)). Similarly, planner failures (mainly due to processing and communication faults) can also be detected and acted upon (Figure 6.9(b)).

When an explorer fault notification is received, the message is relayed to all planners and all pending jobs from the failed explorer are purged. Next, the manager attempts to recover any updated local environment map data from the failed explorer and integrate it with the global map. Following this, the status of the local environment assigned to the failed explorer is updated based on exploration quality (section 6.4.2).

If a planner fails, the notification is forwarded to all other planners by the manager. All jobs delegated to the failed planner are transferred back from *AJL* to *UJL*. Planner failures are generally caused by processor related faults on a robot. Without a functional processor it is difficult to assign any other role or task to a failed planner. Hence, such a robot is usually removed from the team and replaced with another robot (section 6.4).

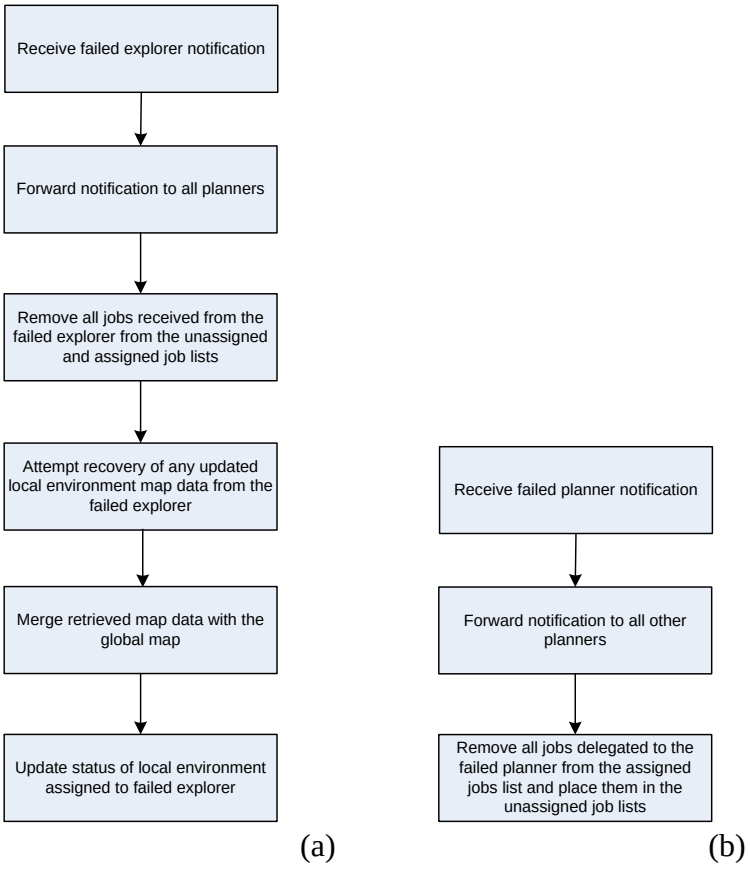


Figure 6.9: Updating the job queue when explorers and planners fail.

6.4.2 Global Map Data and Local Environment Status Updates

Explorers send local environment map data updates to the manager as exploration of the global world proceeds (Figure 6.10(a)). Updates may be sent while navigating to an assigned local environment to free up explorer local map memories for new data. Additionally, updates are sent when exploration of an assigned local environment is complete. Newly acquired local environment map data must be integrated with the existing map data on the manager. Similar to Yamauchi [8], Bayes' rule [130] integrates the new P_{new} and existing P_{old} occupancy probabilities of map nodes to produce a combined occupancy probability P_{com}.

$$P_{com} = \frac{P_{old}.P_{new}}{P_{old}.P_{new} + (1 - P_{old}).(1 - P_{new})} \tag{6.15}$$

Each local environment is assigned one of the following status values: unexplored *UE*, partially explored *PE*, explored *E* or being explored *BE*. Initially, the status of all local environments is set to *UE*. When a local environment is assigned to an explorer its status is updated to *BE*. The status of a local environment changes again after local environment map data received from an explorer is integrated with existing map data. Depending on the overall explored area in the combined map data (6.17), the status is changed to either *E* or *PE* (6.18), (6.19).

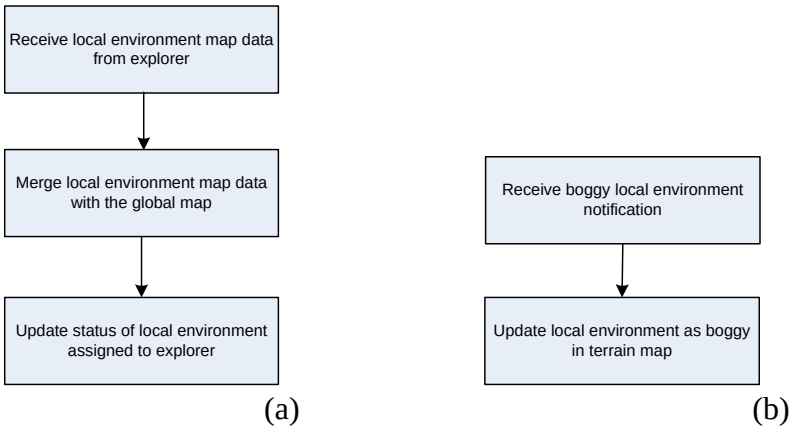


Figure 6.10: Global map data update flowcharts.

A flood-filled map is produced by applying a modified flooding algorithm [143] to the combined map data. The modified flooding algorithm has the ability to flood obstacles with discontinuous boundary nodes. An element-wise (or node-wise) logical

comparison is made between the combined map data *CMD* and flood-filled map data *FMD* to generate logical map data *LMD* (6.16).

$$LMD = \overline{(CMD - 0.5)}.\overline{(FMD)} \tag{6.16}$$

The sum of non-zero entries in *LMD* gives the unexplored nodes $Nodes_{unex}$ in the local environment. A per unit explored area value *EA* can be determined from $Nodes_{unex}$ and the nodes in the combined map data of the local environment $Nodes_{CMD}$. A threshold *EA* value EA_T can be employed to distinguish between *PE* and *E* status ((6.18) (6.19)).

$$EA = 1 - \frac{Nodes_{unex}}{Nodes_{CMD}} \tag{6.17}$$

$$E = EA \geq EA_T \tag{6.18}$$

$$PE = EA < EA_T \tag{6.19}$$

If the status of a local environment assigned to an explorer is changed to *PE* after receiving and integrating map data, the explorer should not be reassigned the same local environment. The manager maintains a record of each explorer's previously assigned *PE* local environments to avoid this scenario.

Explorers also send local environment terrain updates to the manager during exploration. Initially, all local environments are assumed to be non-boggy. When bogs are discovered, the explorer sends a notification to the manager to update the local environment's terrain data as boggy (Figure 6.10(b)).

6.4.3 Estimated Completion Time (ECT) Computation

The goal of ECT computation is to ensure that the best explorers are selected when there are less unexplored local environments than explorer robots. This computation assumes that the local environments and thus global area to be explored has specified boundaries. A planner requests for an ECT acceptability notification after proposing a new local environment assignment for explorer *j* (section 6.3). This request is received by the manager and processed as described in Figure 6.11.

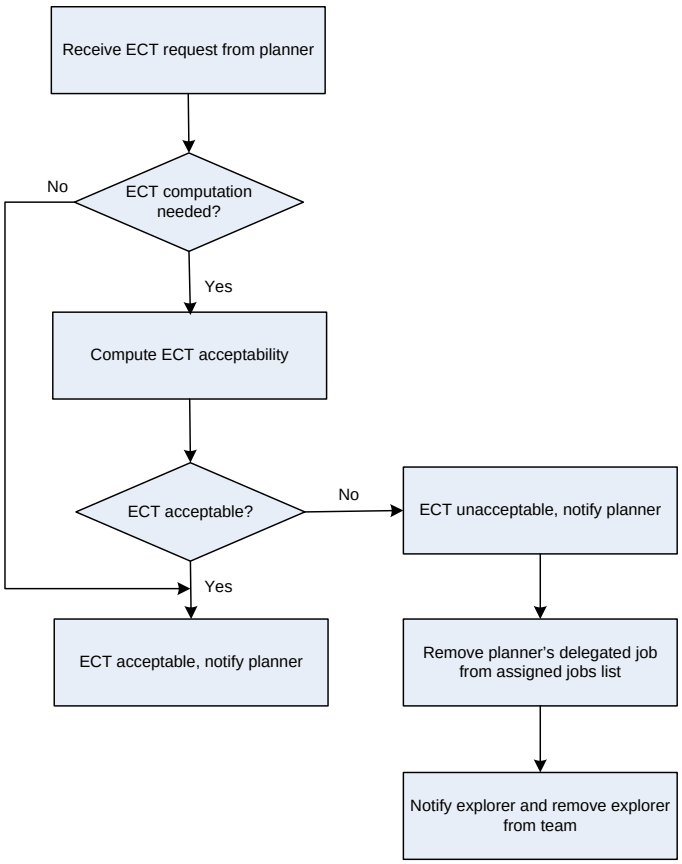


Figure 6.11: ECT flowchart.

ECT computation is required for all explorers if there are less local environments available for exploration than explorers. The feedback coordination performance data (section 5.4.1) are employed to estimate the completion time ECT_k of the k^{th} explorer if the proposed local environment is assigned to it. After computation, the ECT data of all explorers are ranked in ascending order. If explorer *j*'s rank is not below the quantity of local environments available for exploration then its ECT is acceptable. Otherwise, explorer *j*'s ECT is unacceptable. The outcome of ECT acceptability is relayed to the planner that requested the computation. If the explorer's ECT is unacceptable, the manager removes the planner's job from the assigned jobs list. The explorer is also notified and removed from the team by the feedback mechanism (section 5.4).

ECT computation is not required when there are at least as many local environments available as explorers. Each explorer's ECT is assumed to be acceptable and the

manager notifies the planner to continue with the local environment assignment process.

6.5 Scalability

Scalability of the map-building and exploration task is largely dependent on the volume of data that represents a local map. Local map data is frequently utilised by planners for memory constrained global path planning. Explorers receive local map data from managers and return updates to managers. Section 5.7 highlights that worker robots communicate 64 bytes of data to managers for feedback coordination. This is small in comparison to the local map sizes discussed in chapter 4. A key finding from chapter 4 is that local map sizes should be kept small to reduce memory constrained path planning time. Another potential advantage of using smaller local map sizes is that explorers (some of which may be poor) are confined to smaller areas of the global environment.

Assuming the worker robots employ 802.11b wireless devices, a maximum bandwidth of 1408 KB/sec is available. If a local map size of 64 KB is used for exploration with feedback coordination, a maximum of 21 robots can be controlled. This number can be further increased to 96 if 802.11g devices are employed by the workers.

6.6 Summary

This chapter has presented a map-building and exploration strategy that takes advantage of the benefits of hierarchical heterogeneous multi-robot systems. The presented strategy is well suited for three-tiered multi-robot systems such as [16] comprising robots with limited sensing and processing abilities. At the highest level of the hierarchy, manager robots coordinate lower tiered robots and maintain global information regarding the mapping and exploration task. Robots employed for planner and explorer tasks at the second and third tiers of the hierarchical system have limited processing capability or memory capacity. Hence, they utilise the map data stored on a manager robot for global path planning and map-building.

The presented technique also provides different levels of abstraction. Manager robots have a global view of the task. On the other hand, explorer robots are only aware of the section of the environment that they are required to explore. Planners are made aware of what plans they need to make and for which robots.

Moreover, the mapping and exploration task is designed to be employed by the task allocation and feedback mechanism presented in chapter 5. Chapter 8 details the results of performing the multi-robot mapping and exploration task with and without feedback coordination (section 5.4).

7 Task Allocation (Devolution) Experiments

7.1 Overview

This chapter presents experiments on the task allocation strategy described in chapter 5. The multi-robot map-building and exploration task presented in chapter 6 is employed as a model global task for the experiments. Section 7.2 presents the task allocation experiment configurations. It details the tasks, vector of task suitability (VOTS) summation weight sets, and robot sets that have been employed in the experiments. Section 7.3 and section 7.4 present the results of the primary and secondary task devolution processes, respectively. A summary of the experiments detailed in this chapter is given in section 7.6.

7.2 Task Allocation (Devolution) Experiment Configurations

The goal of the task allocation (devolution) process is to select the most suitable robots for the global task. Each robot possesses certain resources while each task requires a particular set of resources. Four categories of resources are employed to encode tasks and robots (section 5.2 and section 5.3). These categories include processing, communication, sensing and actuation. Each resource category can also have a number of sub-resource categories. As explained in chapter 5, resources present on a robot are expressed as numerical vectors of merit (VOM). Similarly, the resources required for a task are expressed as vectors of task requirements (VOTR).

The task devolution process compares the VOTR data for a task with the VOM data of the available robots to produce a VOTS for each task-robot combination (section 5.5). All valid robot-task combination VOTS data are stored in a robot-task capability matrix (RTCM). The robots in the RTCM need to be ranked in descending order of task suitability (based on VOTS data) to permit the selection of the most suitable

robots. This process can be simplified if the VOTS data can be combined into a single number.

A set of VOTS summation weights VSW_i (section 5.2) is employed to combine the resource VOTS scores into a single number (5.2). The VOTS summation weights can be varied depending on the nature of the task. For instance, if a task requires a lot of planning, a greater weighting can be given to processing when ranking the robots. On the other hand, sensing and actuation can be given higher weighting if the task requires a robot to explore an environment. Note that variation of the VOTS summation weights can affect the ranking of a capable robot in the RTCM. In this manner, certain robot(s) amongst those capable can be given higher preference for selection.

In the multi-robot map-building and exploration task presented in chapter 5, there are two manager tasks (MT1 and MT2 in Table 5.20, section 5.4) and two worker tasks (WT1 and WT2 in Table 5.21, section 5.4) that need to be allocated. Table 7.1 details the VOTS summation weight sets evaluated for the manager tasks. A similar set of weights is applied to test the planner (worker) task. Intuitively, the manager and planner (worker) tasks are processing and communication resource intensive. Hence, except for W1, the weights of Table 7.1 give these two resources greater preference than sensing and actuation. Table 7.2 shows the VOTS summation weights tested for the explorer (worker) task. Generally, the explorer task is sensing and actuation resource intensive. Thus, Table 7.2 gives these resources greater preference than processing and communication (except for W1).

Table 7.1: Manager and planner (worker) task VOTS summation weights.

Weight Set	vsw_{proc}	vsw_{comm}	vsw_{sense}	vsw_{act}
W1	0.25	0.25	0.25	0.25
W2	0.30	0.30	0.20	0.20
W3	0.40	0.40	0.10	0.10
W4	0.45	0.45	0.05	0.05
W5	0.50	0.50	0.00	0.00
W6	0.75	0.25	0.00	0.00
W7	1.00	0.00	0.00	0.00
W8	0.25	0.75	0.00	0.00
W9	0.00	1.00	0.00	0.00

Table 7.2: Explorer (worker) task VOTS summation weights.

Weight Set	vsw_{proc}	vsw_{comm}	vsw_{sense}	vsw_{act}
W1	0.25	0.25	0.25	0.25
W2	0.20	0.20	0.30	0.30
W3	0.10	0.10	0.40	0.40
W4	0.05	0.05	0.45	0.45
W5	0.00	0.00	0.50	0.50
W6	0.00	0.00	0.25	0.75
W7	0.00	0.00	0.00	1.00
W8	0.00	0.00	0.75	0.25
W9	0.00	0.00	1.00	0.00

Table 7.3: VOM data of five sets of eight heterogeneous mobile robots.

Robot ID	Resource	Value				
		Set (a)	Set (b)	Set (c)	Set (d)	Set (e)
R1	Proc. Cap.	[3.1,1,pc]	[2.1,1,pc]	[2.1,0.5,pc]	[2.1,0.5,pc]	[3.1,1,pc]
	Comm. Cap.	[54,100]	[54,100]	[54,100]	[54,100]	[54,100]
	Sens. Cap.	[12,1.5,360]	[13,1.5,360]	[13,1.5,360]	[11,1.5,360]	[14,1.5,360]
	Act. Cap.	[2.00,0.60,0.5,0.3]	[2.00,0.60,0.5,0.3]	[2.00,0.60,0.5,0.1]	[2.00,0.60,0.5,0.3]	[2.00,0.60,0.4,1]
R2	Proc. Cap.	[2.1,1,pc]	[3.1,0.5,pc]	[3.1,1,pc]	[3.1,1,pc]	[2.1,1,pc]
	Comm. Cap.	[54,100]	[54,100]	[54,100]	[54,100]	[54,100]
	Sens. Cap.	[13,1.5,360]	[13,1.5,360]	[13,1.5,360]	[13,1.5,360]	[12,1.5,360]
	Act. Cap.	[2.00,0.60,0.5,0.1]	[2.00,0.60,0.5,0.1]	[2.00,0.60,0.5,0.1]	[2.00,0.60,0.4,1]	[2.00,0.60,0.4,1]
R3	Proc. Cap.	[6,4,mc]	[10,16,mc]	[10,16,mc]	[10,16,mc]	[10,16,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[12,1.5,360]	[12,1.5,360]	[12,1.5,360]	[13,1.5,360]	[13,1.5,360]
	Act. Cap.	[1.06,0.25,0.4,1]	[1.50,0.40,0.5,0.1]	[1.19,0.25,0.5,0.3]	[1.19,0.25,0.5,0.3]	[1.19,0.25,0.5,0.3]
R4	Proc. Cap.	[10,16,mc]	[6,4,mc]	[6,4,mc]	[10,16,mc]	[6,2,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[14,1.5,360]	[13,1.5,360]	[12,1.5,360]	[12,1.5,360]	[13,1.5,360]
	Act. Cap.	[1.50,0.40,0.5,0.1]	[0.95,0.15,0.5,0.3]	[0.95,0.15,0.5,0.3]	[1.19,0.25,0.5,0.3]	[0.85,0.15,0.4,1]
R5	Proc. Cap.	[10,16,mc]	[10,16,mc]	[10,16,mc]	[10,16,mc]	[10,16,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[13,1.5,360]	[13,1.5,360]	[12,1.5,360]	[11,1.5,360]	[14,1.5,360]
	Act. Cap.	[1.50,0.40,0.5,0.1]	[1.43,0.40,0.5,0.3]	[1.25,0.25,0.5,0.1]	[1.25,0.25,0.5,0.1]	[1.50,0.40,0.5,0.1]
R6	Proc. Cap.	[6,4,mc]	[10,16,mc]	[10,16,mc]	[6,2,mc]	[6,4,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[12,1.5,360]	[11,1.5,360]	[14,1.5,360]	[11,1.5,360]	[13,1.5,360]
	Act. Cap.	[1.06,0.25,0.4,1]	[1.43,0.40,0.5,0.3]	[1.19,0.25,0.5,0.3]	[1.19,0.25,0.5,0.3]	[1.19,0.25,0.5,0.3]
R7	Proc. Cap.	[6,2,mc]	[6,2,mc]	[6,4,mc]	[6,4,mc]	[10,16,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[13,1.5,360]	[11,1.5,360]	[12,1.5,360]	[13,1.5,360]	[11,1.5,360]
	Act. Cap.	[1.43,0.40,0.5,0.3]	[1.43,0.40,0.5,0.3]	[1.06,0.25,0.4,1]	[1.25,0.25,0.5,0.1]	[1.25,0.25,0.5,0.1]
R8	Proc. Cap.	[6,4,mc]	[6,4,mc]	[10,16,mc]	[6,4,mc]	[10,16,mc]
	Comm. Cap.	[11,100]	[11,100]	[11,100]	[11,100]	[11,100]
	Sens. Cap.	[13,1.5,360]	[12,1.5,360]	[10,1.5,360]	[11,1.5,360]	[10,1.5,360]
	Act. Cap.	[1.25,0.25,0.5,0.1]	[1.19,0.25,0.5,0.3]	[1.19,0.25,0.5,0.3]	[0.95,0.15,0.5,0.3]	[1.06,0.25,0.4,1]

Table 7.3 details randomly generated VOM data for five sets of eight robots. Each robot set is independent and represents a situation where eight candidate robots are available for task allocation. The numerical values for each resource type have been randomly selected from a set of practical values. Section 5.3 explains the numerical data values presented in Table 7.3 and section 5.4 provides sample data from real robots.

Table 7.4: VOM data of Table 7.3 simplified with FISs.

Robot ID	Resource	Value				
		Set (a)	Set (b)	Set (c)	Set (d)	Set (e)
R1	Proc. Cap.	[2,0.54]	[2,0.53]	[2,0.46]	[2,0.46]	[2,0.54]
	Comm. Cap.	[0.87]	[0.87]	[0.87]	[0.87]	[0.87]
	Sens. Cap.	[0.52]	[0.54]	[0.54]	[0.51]	[0.55]
	Act. Cap.	[0.58]	[0.58]	[0.58]	[0.58]	[0.58]
R2	Proc. Cap.	[2,0.53]	[0.47]	[2,0.54]	[2,0.54]	[2,0.53]
	Comm. Cap.	[0.87]	[0.87]	[0.87]	[0.87]	[0.87]
	Sens. Cap.	[0.54]	[0.54]	[0.54]	[0.54]	[0.52]
	Act. Cap.	[0.58]	[0.58]	[0.58]	[0.58]	[0.58]
R3	Proc. Cap.	[1,0.38]	[1,0.86]	[1,0.86]	[1,0.86]	[1,0.86]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.52]	[0.52]	[0.52]	[0.54]	[0.54]
	Act. Cap.	[0.39]	[0.50]	[0.49]	[0.49]	[0.49]
R4	Proc. Cap.	[1,0.86]	[1,0.38]	[1,0.38]	[1,0.86]	[1,0.33]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.55]	[0.54]	[0.52]	[0.52]	[0.54]
	Act. Cap.	[0.50]	[0.30]	[0.30]	[0.49]	[0.26]
R5	Proc. Cap.	[1,0.86]	[1,0.86]	[1,0.86]	[1,0.86]	[1,0.86]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.54]	[0.54]	[0.52]	[0.51]	[0.55]
	Act. Cap.	[0.50]	[0.50]	[0.50]	[0.50]	[0.50]
R6	Proc. Cap.	[1,0.38]	[1,0.86]	[1,0.86]	[1,0.33]	[1,0.38]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.52]	[0.51]	[0.55]	[0.51]	[0.54]
	Act. Cap.	[0.39]	[0.50]	[0.49]	[0.49]	[0.49]
R7	Proc. Cap.	[1,0.33]	[1,0.33]	[1,0.38]	[1,0.38]	[1,0.86]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.54]	[0.51]	[0.52]	[0.54]	[0.51]
	Act. Cap.	[0.50]	[0.50]	[0.39]	[0.50]	[0.50]
R8	Proc. Cap.	[1,0.38]	[1,0.38]	[1,0.86]	[1,0.38]	[1,0.86]
	Comm. Cap.	[0.63]	[0.63]	[0.63]	[0.63]	[0.63]
	Sens. Cap.	[0.54]	[0.52]	[0.50]	[0.51]	[0.50]
	Act. Cap.	[0.50]	[0.49]	[0.49]	[0.30]	[0.39]

The detailed robot capability data presented in Table 7.3 is combined with fuzzy inference systems (section 5.3) to produce an overall score for each resource type (Table 7.4). Table 7.4 and the task descriptions presented in chapter 5 (Table 5.20 and Table 5.21) are utilised for task devolution experiments in this chapter.

The effect of applying the VOTS summation weights of Table 7.1 and Table 7.2 on the five sets of robots has been investigated. Results of applying the VOTS summation weights of Table 7.1 and Table 7.2 are presented in Figure 7.1 – Figure 7.15. The five graphs ((a)–(e)) in each figure illustrate results for each robot set (Table 7.3). All robots identified in the results figures are capable of task execution. If a robot is not listed in the graphs then it is incapable of task execution.

VOTS weighted sum (VOTSWS) data (Figure 7.1 and Figure 7.4) and Value data (Figure 7.7 and Figure 7.11) illustrate intermediate results of the task allocation process. The data presented in these figures is employed to rank the capable robots and select the most suitable robot(s). While VOTSWS data are computed using VOTS data and summation weights only, Value data calculation also includes a task diversity capability (TD) input (Table 5.26, section 5.5.1). VOTSWS data ranks candidate manager robots and Value data ranks candidate worker robots.

In Figure 7.1 and Figure 7.4, it is important to note that higher VOTSWS data values does not necessarily mean that the corresponding weight set is better. Instead, it is the relative VOTSWS data values within a weight set that is important. This is also valid for the Value data presented in Figure 7.7 and Figure 7.11. The sole purpose of varying the VOTS summation weights is to investigate change in robot rankings based on the calculated VOTSWS and Value data. Section 5.5.2 presents an example of VOTSWS and Value data calculation for the task and robot descriptions detailed in Table 5.20 – Table 5.22.

Figure 7.3, Figure 7.6, Figure 7.10, Figure 7.14 and Figure 7.15 display the robots that are assigned tasks. In these figures, an ‘x’ indicates that a robot has been selected for the respective task.

Since each robot set is independent and robot capabilities are randomly selected, comparisons between robots with the same IDs are invalid.

7.3 Primary Task Devolution Results

Figure 7.1 illustrates the variation in VOTS weighted sum (VOTSWS) data as the VOTS summation weights are varied according to Table 7.1 for manager task MT1. In all five robot sets, only robots R1 and R2 are capable of executing task MT1. The VOTSWS data varies in all five robot sets, but does not indicate a change in robot rankings (Figure 7.1 (a)–(e)).

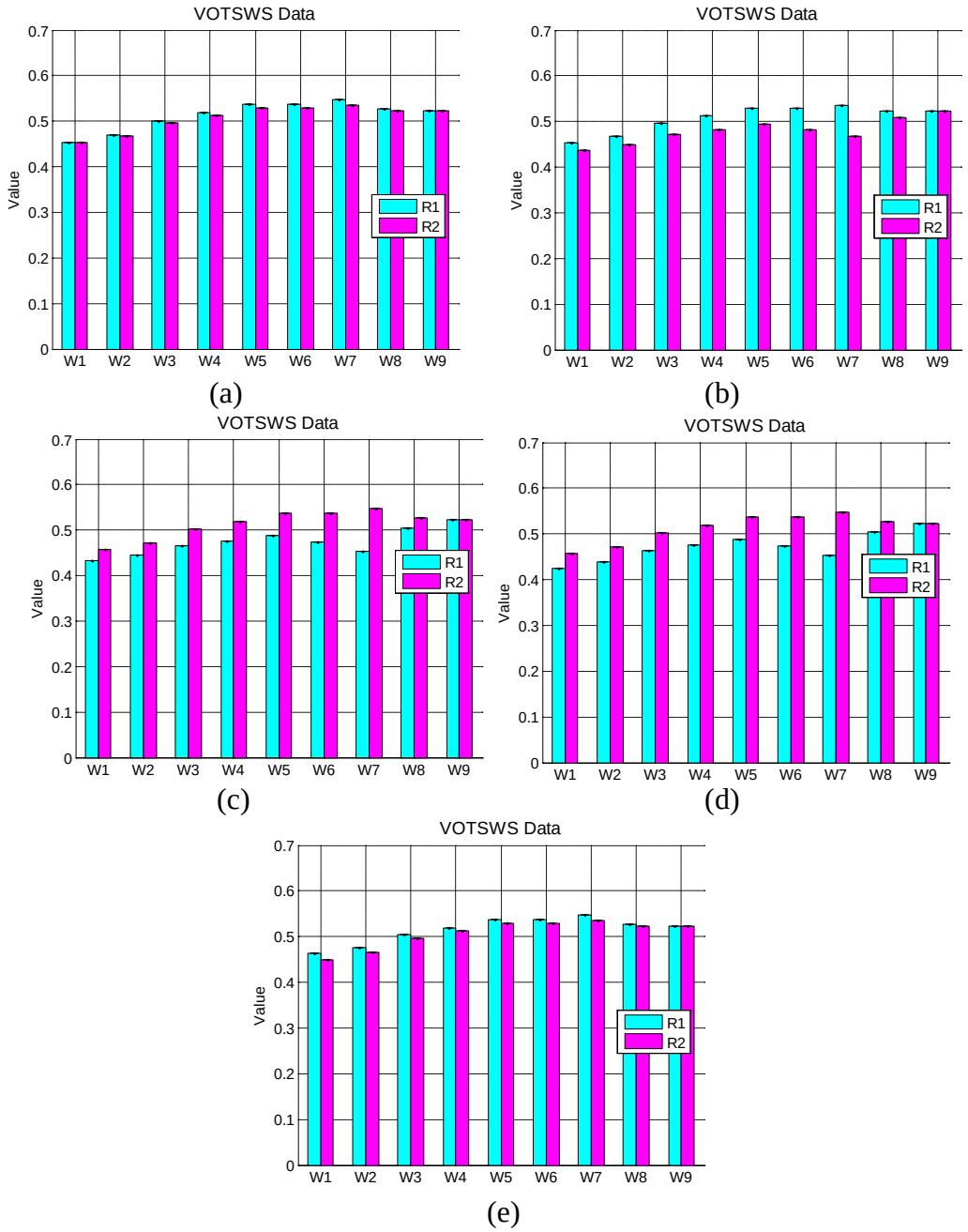


Figure 7.1: VOTSWS data for manager task MT1.

Applying non-zero weightings for sensing and actuation (W1–W4 in Table 7.1) can alter the difference between VOTSWS data of the candidate robots. Weight set W7 gives weighting only to processing for all robot sets. When the processing resources of robots R1 and R2 are compared to the low processing requirement of task MT1, large VOTS data is obtained for the processing resource. This results in higher VOTSWS values when weight set W7 is employed.

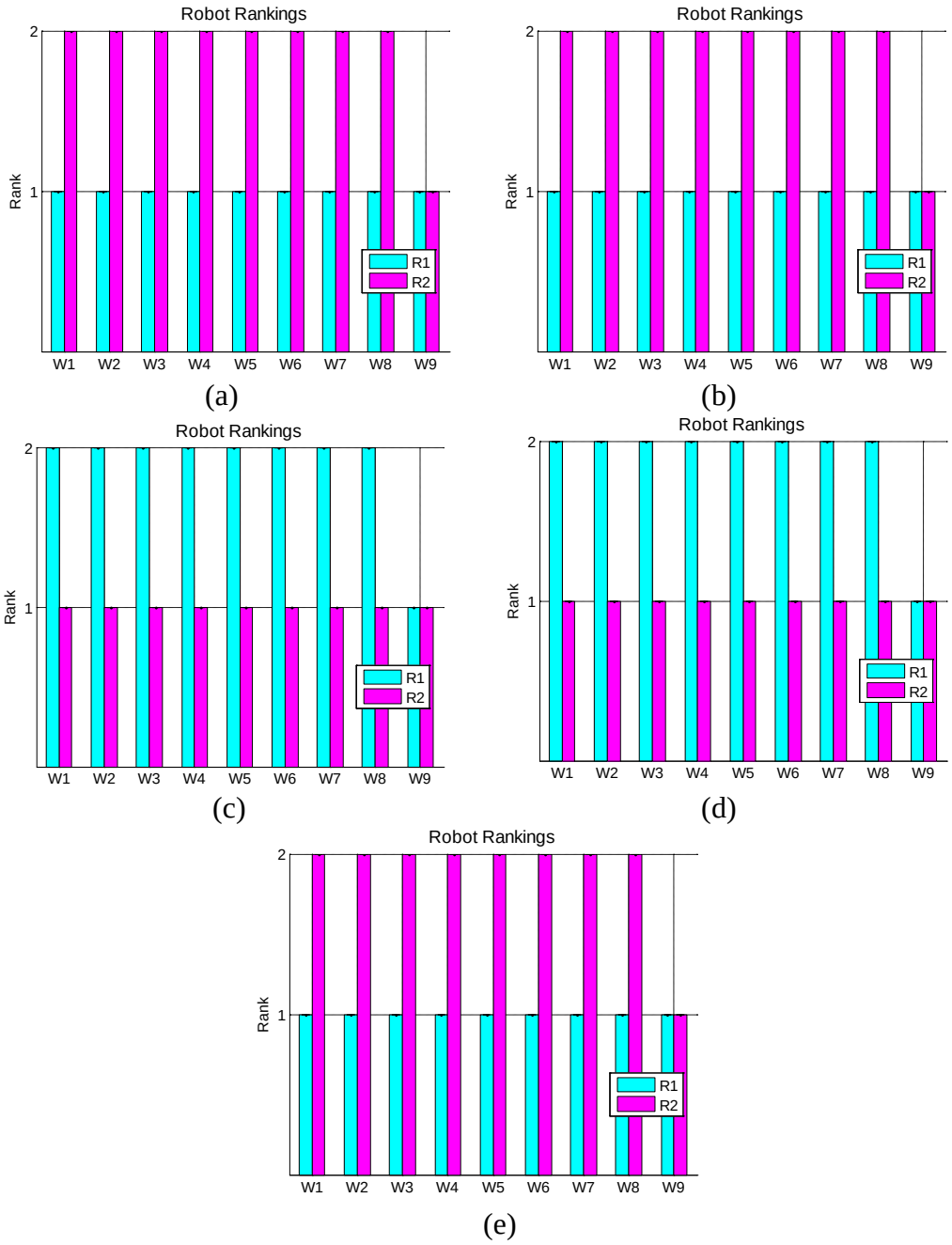


Figure 7.2: Robot rankings for manager task MT1.

For weight set W9, weighting is only given to communication. Since all capable robots possess the same communication resources they have equal VOTSWS data at

weight W9. Figure 7.2 (a)–(e) shows that the rankings of robots R1 and R2 remain the same for weight sets W1–W8. Thus, Figure 7.3 (a)–(e) shows that the same robot is assigned manager task MT1 within each robot set, for all the tested weight sets.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2	x	x	x	x	x	x	x	x	x
R3									
R4									
R5									
R6									
R7									
R8									

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2	x	x	x	x	x	x	x	x	x
R3									
R4									
R5									
R6									
R7									
R8									

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1	x	x	x	x	x	x	x	x	x
R2									
R3									
R4									
R5									
R6									
R7									
R8									

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1	x	x	x	x	x	x	x	x	x
R2									
R3									
R4									
R5									
R6									
R7									
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2	x	x	x	x	x	x	x	x	x
R3									
R4									
R5									
R6									
R7									
R8									

(e)

Figure 7.3: Robot(s) assigned to manager task MT1.

Intuitively, weight set W5 has been selected to give equal weighting to processing and communication capability data.

Figure 7.4, Figure 7.5 and Figure 7.6 show the VOTSWS data, robot rankings and selected robots, respectively, for manager task MT2. Similar to manager task MT1,

the VOTSWS data varies in all robot sets (Figure 7.4 (a)–(e)). Minimum VOTSWS data is obtained when weighting is given to processing only (weight set W7 in Table 7.1). A higher processing requirement for task MT2 results in lower VOTSWS values when weight set W7 is employed.

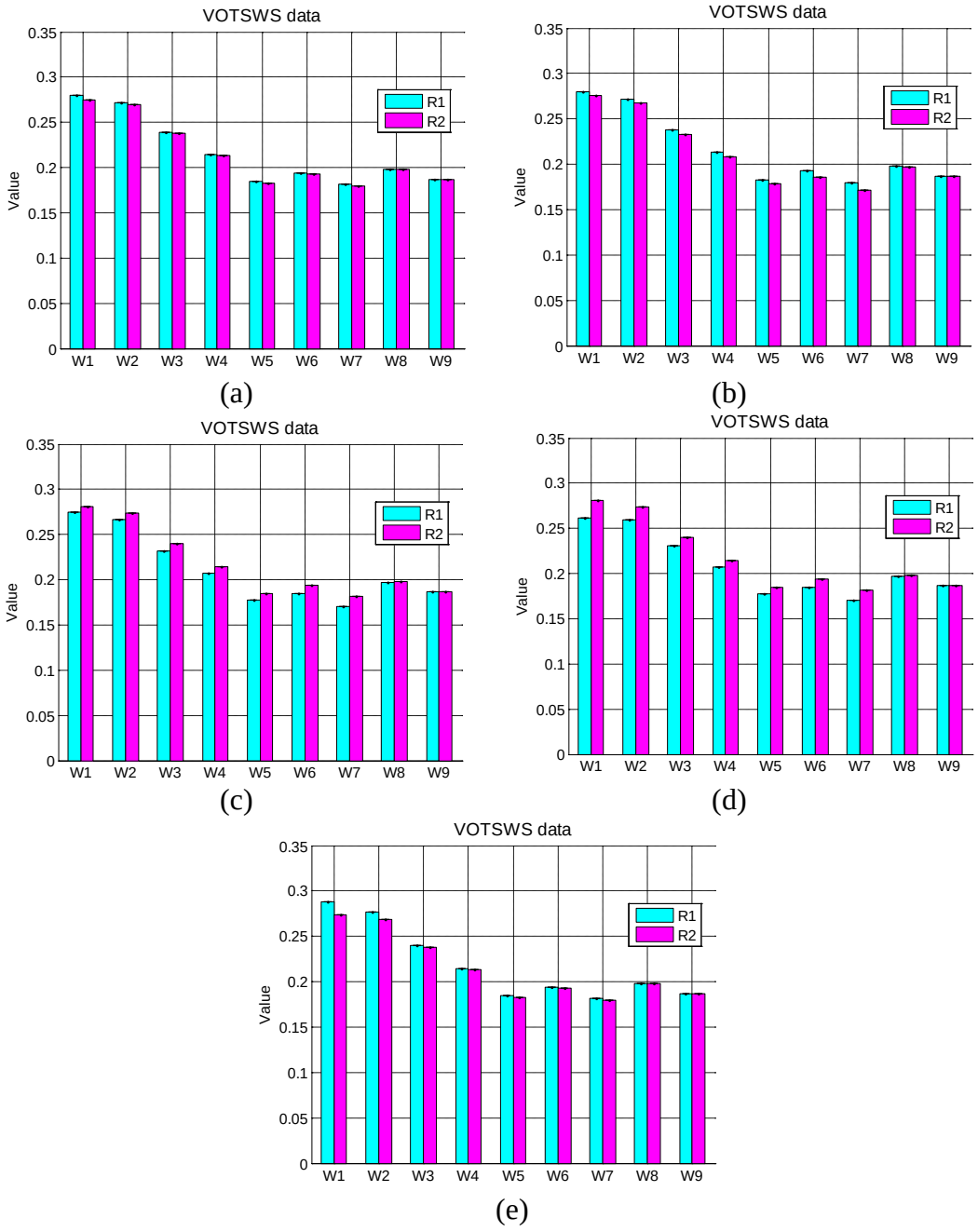


Figure 7.4: VOTSWS data for manager task MT2.

All capable robots have identical VOTSWS data for weight set W9 since they possess identical communication resources. Similar to task MT1, applying non-zero weightings to sensing and actuation resources (W1–W4 in Table 7.1) alters VOTSWS data. Weight sets W1–W8 do not alter the rankings of capable robots within each

robot set (Figure 7.5 (a)–(e)). Hence, in each robot set, all tested weight combinations assign the same robot to task MT2 (Figure 7.6 (a)–(e)).

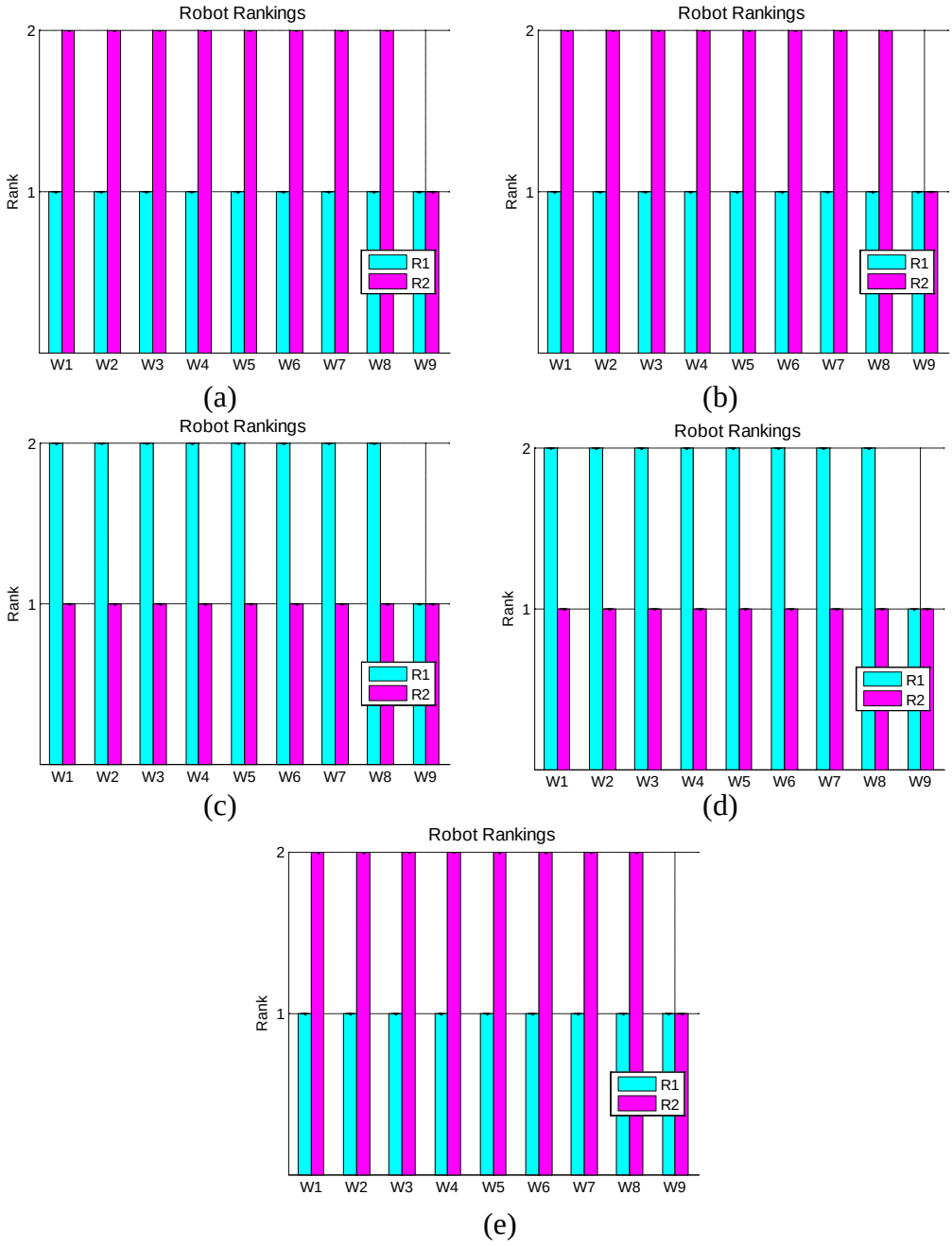


Figure 7.5: Robot rankings for manager task MT2.

It is important to point out that the second best robot is always selected for task MT1 (Figure 7.2 and Figure 7.3) while the best robot is selected for task MT2 (Figure 7.5 and Figure 7.6). Task MT2 is more difficult than task MT1 due to its higher processing and communication requirements. This gives task MT2 priority for assignment during task allocation. Figure 7.1 (task MT1) shows VOTSWS values

greater than 0.4 for all tested configurations. On the other hand, VOTSWS values are less than 0.3 for all tested configurations in Figure 7.4 (task MT2). Since the VOTSWS values for task MT1 are greater than task MT2, this verifies that task MT2 is more difficult than task MT1.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1	x	x	x	x	x	x	x	x	x
R2									
R3									
R4									
R5									
R6									
R7									
R8									

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1	x	x	x	x	x	x	x	x	x
R2									
R3									
R4									
R5									
R6									
R7									
R8									

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2	x	x	x	x	x	x	x	x	x
R3									
R4									
R5									
R6									
R7									
R8									

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2	x	x	x	x	x	x	x	x	x
R3									
R4									
R5									
R6									
R7									
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1	x	x	x	x	x	x	x	x	x
R2									
R3									
R4									
R5									
R6									
R7									
R8									

(e)

Figure 7.6: Robots assigned to manager task MT2.

7.4 Secondary Task Devolution Results

7.4.1 Worker Task WT1 (Planner)

Figure 7.7 – Figure 7.10 details the results of VOTS summation weight variation (Table 7.1) for worker task WT1 (planner). Weight sets W1–W4 give weighting to all four resource types. No weighting is given to sensing and actuation resources in weight sets W5–W9. In weight set W7, weighting is only given to processing. Communication is solely preferred in weight set W9.

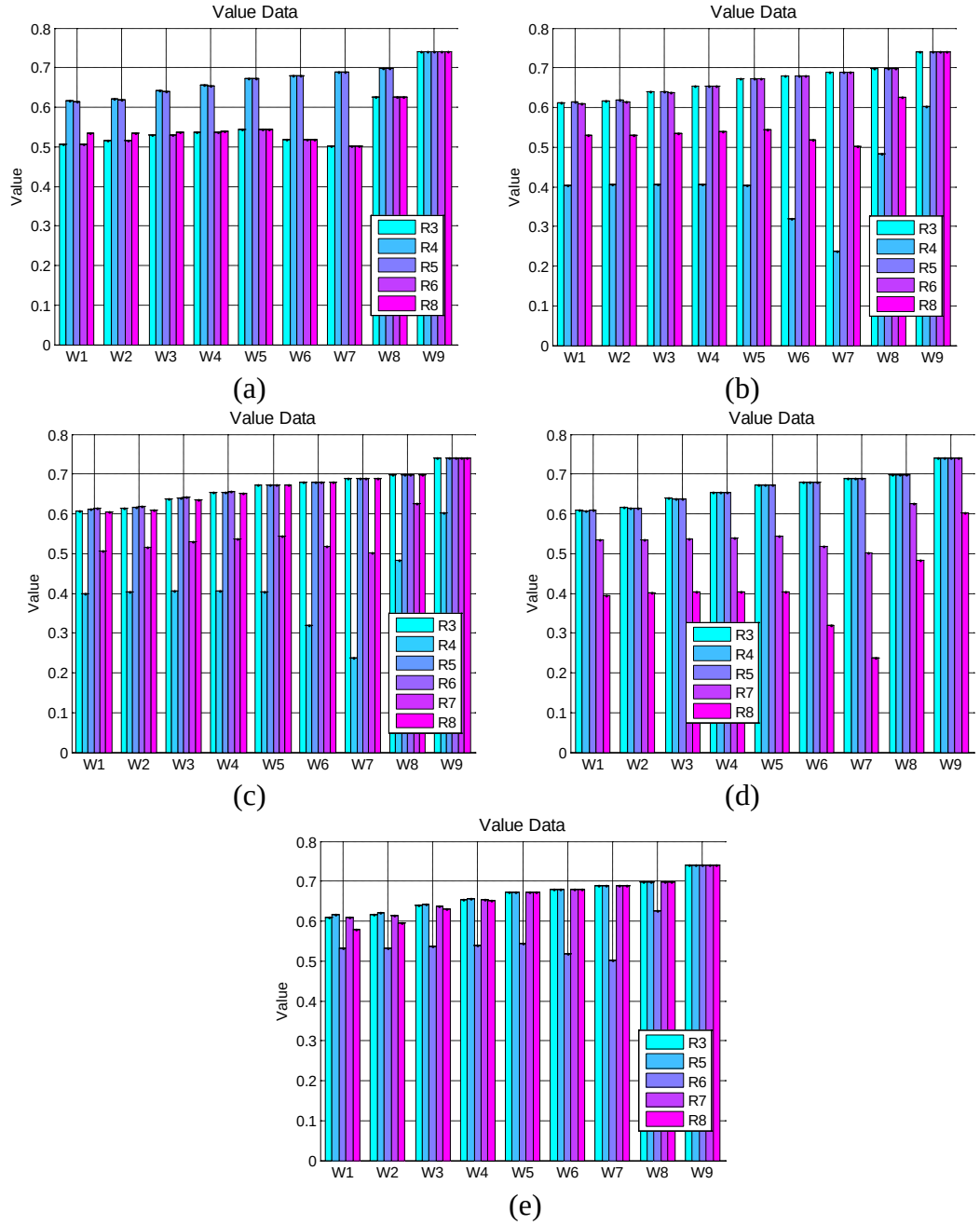


Figure 7.7: Value data for worker task WT1 (planner).

Figure 7.7 shows the variation in Value data as the weights are varied according to Table 7.1. The robots plotted in each robot set are capable of executing worker task WT1. In all five robot sets, robots R1 and R2 are not identified as candidate worker robots since they have already been categorised as manager robots in section 7.3. Robots with 'good' processing capabilities tend to show a gradual increase in Value data as the weight sets are varied from W1–W9. For example, R4 and R5 in robot set (a) (Figure 7.7 (a)) have processing capability scores of 0.86 (as illustrated in Table 7.4). A similar trend is visible in the other robot sets.

Value data is highest for all robots at weight set W9. This indicates that all robots have communication capabilities that are much greater than the task's minimum requirement when compared to other resource types (processing, sensing, or actuation). All candidate worker robots possess identical communication devices in the tested configurations. However, Value data can vary depending on the number of different tasks that a robot is capable of performing. For example, R4 in robot set (b) (Figure 7.7 (b)) has a lower Value than other robots at W9 since it is incapable of executing task WT2 (explorer).

Robots with processing capabilities close to the minimum requirement generally exhibit little variation in Value data as the weight sets are varied from W1–W5 (e.g. R8 in sets (a) and (d) (Figure 7.7 (a),(d)) or R4 in sets (b) and (c) (Figure 7.7 (b),(c))). However, there can be a slight increase until W5 (e.g. R3 in set (a) (Figure 7.7 (a)), or R8 in set (b) (Figure 7.7 (b)) or R7 in set (c) (Figure 7.7 (c)) attributed to the good communication capabilities of the robots. For weights sets W6 and W7, Value data is decreased due to the high weighting of processing capability. Weight sets W8 and W9 exhibit an increase in Value compared to W7 in all robot sets. This is due to the weighting for communication being greater than processing.

Based on the Value data of Figure 7.7, Figure 7.8 illustrates the robot rankings for the planner worker task. A robot with a rank of unity has the highest rank.

Figure 7.8 shows that four robot sets (Figure 7.8 (a),(b),(c),(e)) out of five produce identical rankings within their respective sets when weight sets W1–W4 are applied. Robot set (d) (Figure 7.8 (d)) shows a change in ranking for R4 and R5 when weight

sets W3 and W4 are employed. In weight sets W1 and W2, R4 and R5 are ranked third and second, respectively. But, in weight sets W3 and W4, R4 and R5 are ranked second and third, respectively. This change can be attributed to the fuzzy combination of VOTS data. It should also be noted that the Value data of R4 and R5 are almost identical for weight sets W2, W3, and W4 (Figure 7.7 (d)). A key point to note is that each robot has a unique rank when weight sets W1–W4 are applied to the tested robot sets.

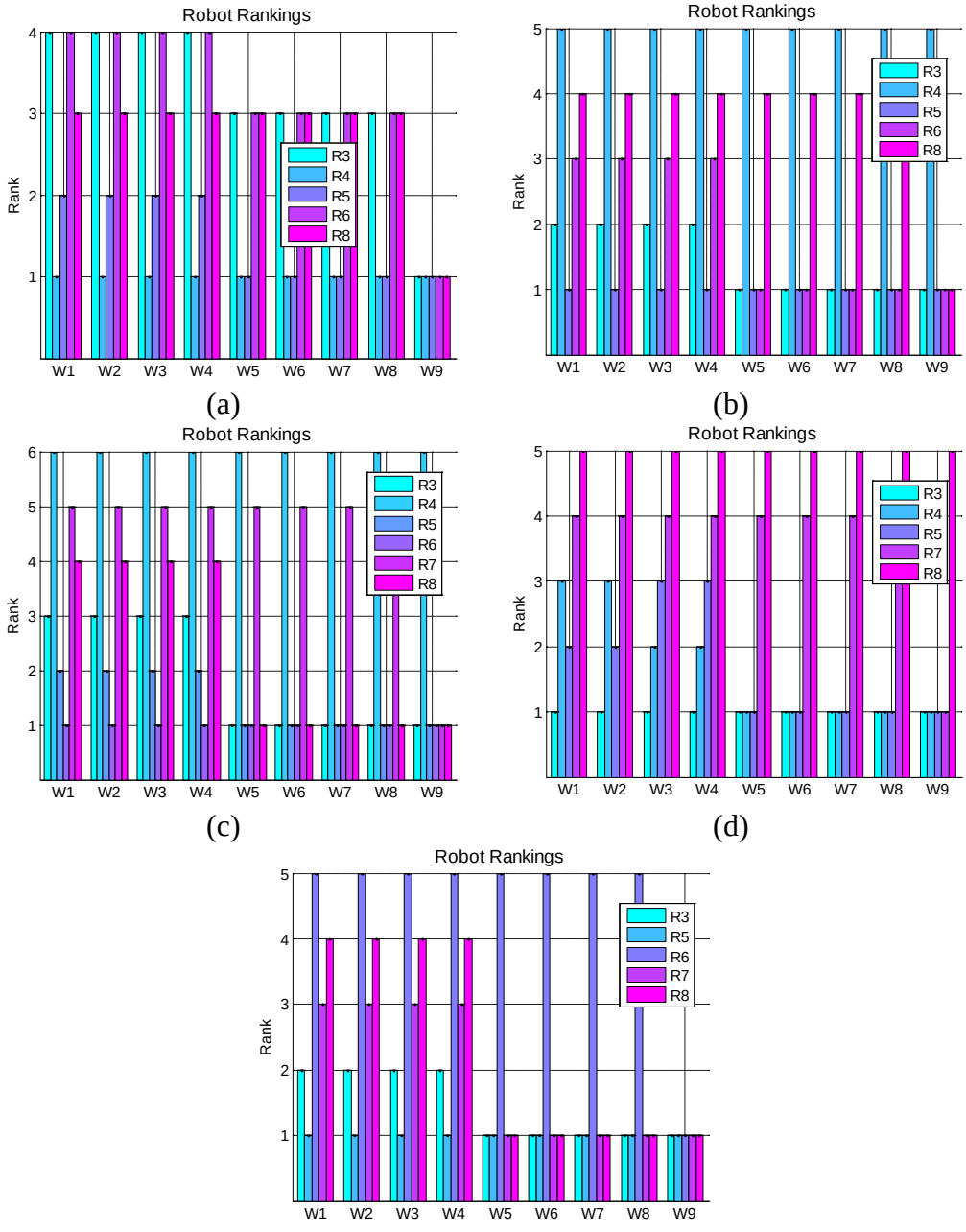


Figure 7.8: Robot rankings for worker task WT1 (planner).

Weight sets W5–W8 produce identical robot rankings within each robot set for all tested robot sets. Additionally, multiple robots have identical ranks for an individual weight set when these weight sets are employed. Similar to weight sets W5–W8, weight set W9 produces identical ranks for multiple robots in all robot sets.

Hence, Figure 7.8 illustrates that applying weightings to sensing and actuation in addition to processing and communication allows each robot to have a unique rank. A planner task is generally more processing and communication intensive than sensing and actuation. This means that the weightings of processing and communication should be greater than sensing and actuation. Weight set W3 has been selected since it has low weightings for sensing and actuation. It should be noted that weight sets W2 or W4 could be used as well. Applying small weightings to sensing and actuation also permits the task allocation process to give a higher ranking (closer to 1) to robots with better sensing and actuation resources. This is beneficial since a planner may also need to be mobile during execution of the multi-robot exploration task.

Worker task WT1 task score data (Figure 7.9) is utilised by the mapping and exploration task (chapter 6) for allocating planning requests made by explorers to the planners. Figure 7.9 illustrates that the task score data for robots with good processing resources approaches unity as the weight sets are altered from W1–W7. One example is R5 and R6 in robot set (b). Another example is R3, R5, and R8 in robot set (c). On the other hand, task score data for robots with weaker processing resources approaches zero as the weights are varied from W1–W7. An example of this is R4 and R8 in robot set (b). Another example is R7 and R8 in robot set (d). Task score data are high in weight sets W8 and W9 due to the high weighting of the good communication resources present on all robots.

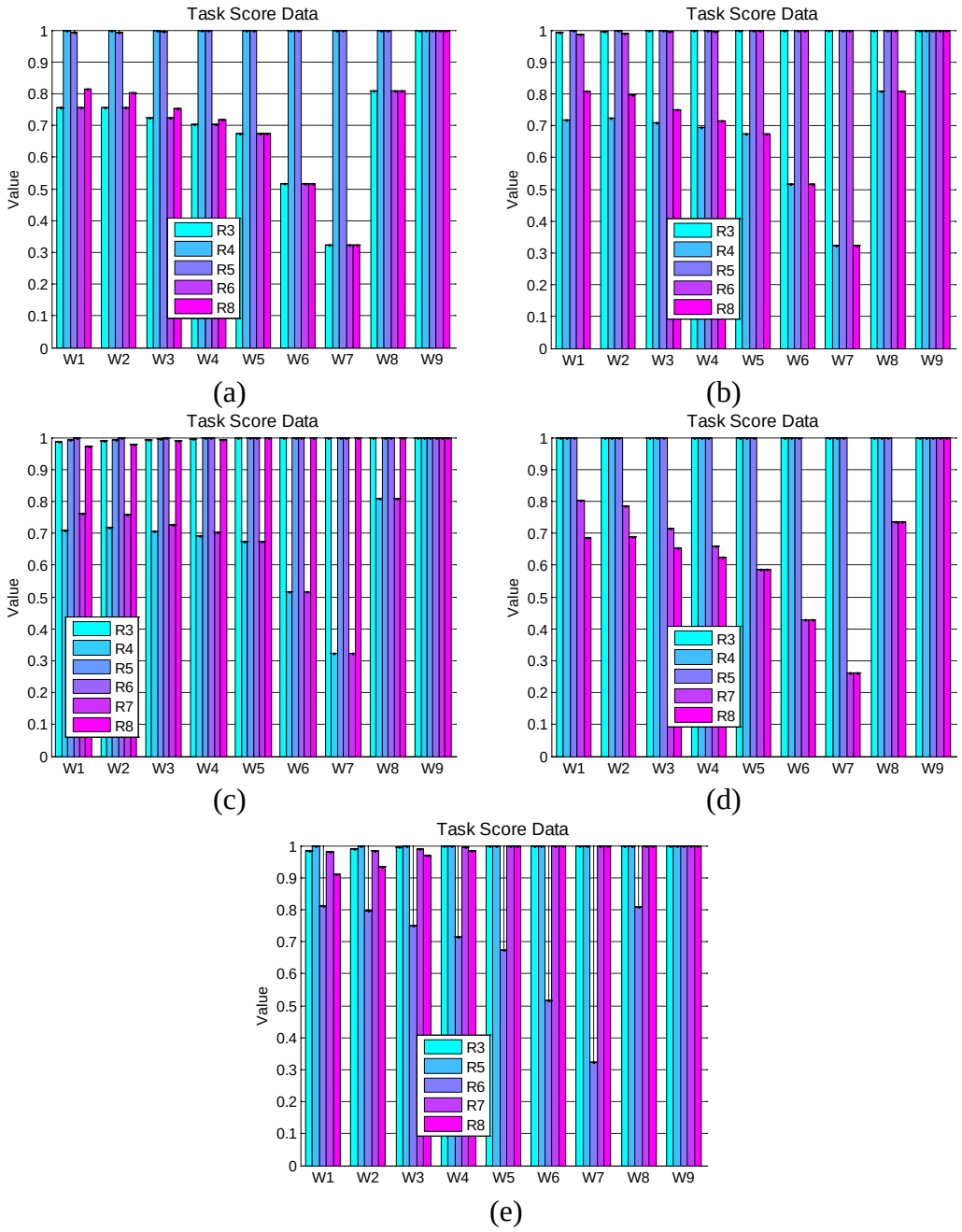


Figure 7.9: Task score data for worker task WT1 (planner).

Figure 7.10 details the robots assigned to the planner worker task for the tested weight and robot sets. In all five robot sets, the same robot is selected within each respective set for weight sets W1–W4. Similarly, the same robot within each respective robot set is chosen when weight sets W5–W8 are applied. Weight set W9 only gives weighting to the identical communication resources present on all robots. Thus, the robot with the lowest identification number (R3) is selected for all robot sets when weight set W9 is applied. In three robot sets (Figure 7.10 (b),(c),(e)), applying small weightings to sensing and actuation (W1–W4) selects a robot that is different from the selection

when no weighting is given to these two resource types (W5–W9). However, the selected robots have identical processing resources for all three robot sets.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3									x
R4	x	x	x	x	x	x	x	x	
R5									
R6									
R7									
R8									

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3					x	x	x	x	x
R4									
R5	x	x	x	x					
R6									
R7									
R8									

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3					x	x	x	x	x
R4									
R5									
R6	x	x	x	x					
R7									
R8									

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5									
R6									
R7									
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3					x	x	x	x	x
R4									
R5	x	x	x	x					
R6									
R7									
R8									

(e)

Figure 7.10: Robots assigned to worker task WT1 (planner).

7.4.2 Worker Task WT2 (Explorer)

Results of VOTS summation weight variation (Table 7.2) for worker task WT2 (explorer) are illustrated in Figure 7.11 – Figure 7.15. Weightings are given to all four resource types in weight sets W1–W4. Weight sets W5–W9 do not give any weighting to processing and communication resources. Actuation is the only resource given weighting in weight set W7. Weight set W9 only gives weighting to sensing.

Figure 7.11 details the variation in Value data as the weights are changed according to Table 7.2. In all robot sets, the low processing and communication requirements for the explorer task produces higher Value data in weight sets W1–W4 when compared to weight set W5. As the weight sets are altered from W1 to W4, the weightings for processing and communication are reduced thus lowering the Value data.

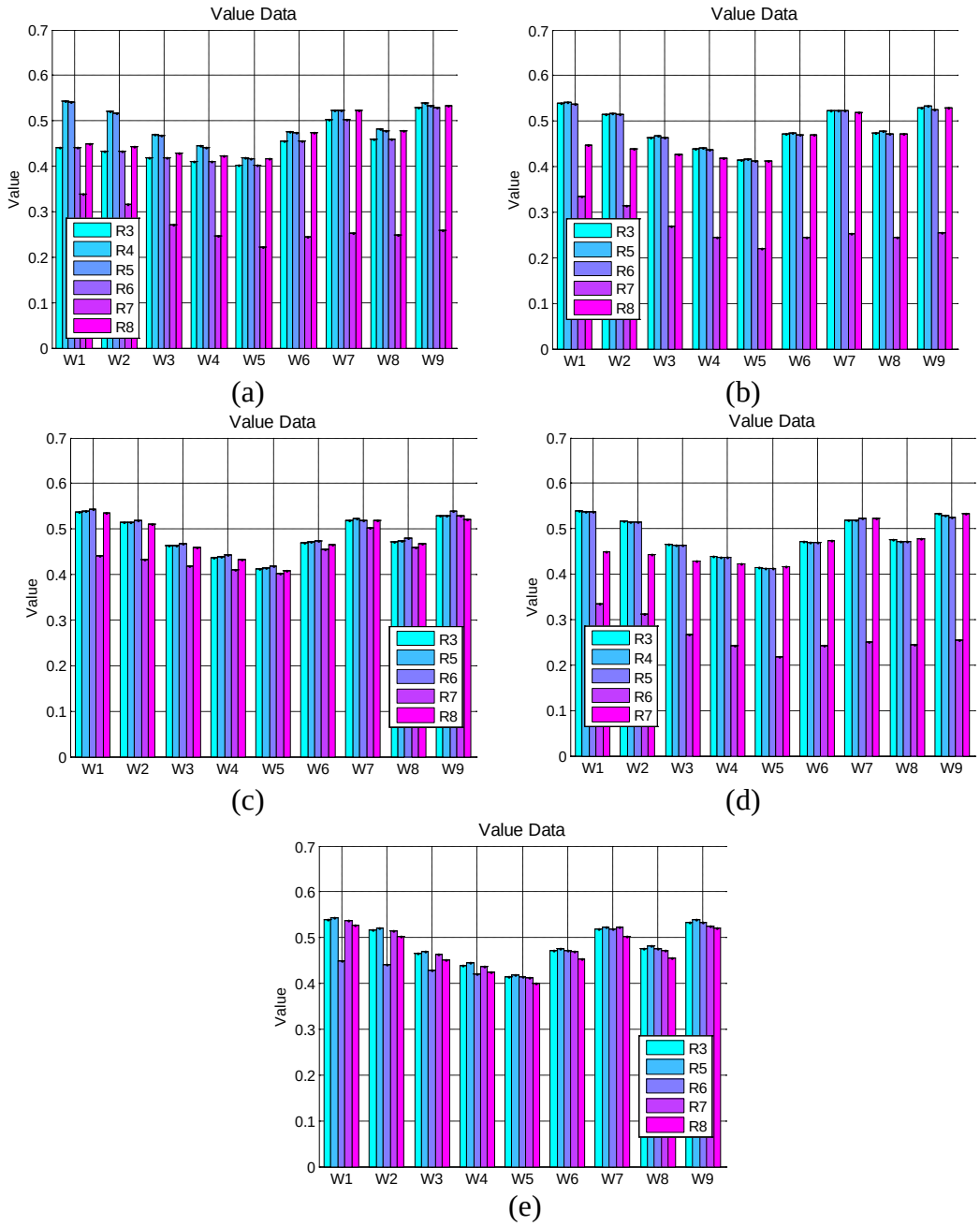


Figure 7.11: Value data for worker task WT2 (explorer).

Equal weighting is given to sensing and actuation in weight set W5. Weight sets W7 and W9 give weighting to only sensing and actuation, respectively. Comparing the

Value data for these three weight sets shows that W5 has the lowest Value data. This trend is present in all robot sets. Employing FISs to calculate Value data produces these reduced values. However, the value data within a particular weight set is critical for ranking robots. When compared to 100% actuation (W7) and 100% sensing (W9), the Value data for all robots is reduced proportionally in W5.

Similar to weight set W5, weight sets W6 and W8 also exhibit a slight reduction in Value data when compared with 100% actuation (W7) and 100% sensing (W9). This reduction in Value is also due to the FISs employed for Value data calculation. At W6, the FISs give 75% weighting to actuation and 25% weighting to sensing but reduce the Value data for all robots in similar proportion. A similar trend is noticed at W8 where 25% weighting is given to actuation and 75% weighting to sensing.

In Figure 7.11 (a),(b),(d), one robot (R7, R7, and R6, respectively) has a much lower value compared to the other robots within the respective robot set. This is due to that particular robot not having sufficient processing resources to execute planner task WT1.

Figure 7.12 illustrates the robot rankings obtained from the Value data of Figure 7.11 for the explorer task. Similar to the planner task WT1, a robot with a rank of unity has the highest rank. All five robot sets produce identical rankings within their respective sets for weight sets W1–W4. One robot set (Figure 7.12 (a)) produces an identical rank (4^{th}) for two robots (R3 and R6) for all weightings. Table 7.4 shows that these two robots have identical resource capabilities. In all other robot sets, each robot has a unique rank when weight sets W1–W4 are applied.

Applying weight sets W5–W9 ranks the candidate robots based on sensing and actuation resources only. It is possible for robots to have identical rankings for some weight sets due to identical sensing and actuation resources. This is valid for all robots sets (Figure 7.12 (a)–(e)) at weight sets W7 and W9 where weighting is only given to actuation and sensing resources, respectively. Figure 7.12 (a)–(e) show that robot rankings can be altered when weight sets W5–W9 are employed. At most, the ranking of a robot changes by one position if both sensing and actuation are weighted (weight sets W5, W6, and W8). A maximum rank change of three positions is shown in Figure 7.12 (e) at weight set W7 for R7. Similarly, R7 in Figure 7.12 (c) exhibits a

rank change of three positions at weight set W9. In these two situations, the change is due to a robot being strong in one resource type but weaker in all other resource types.

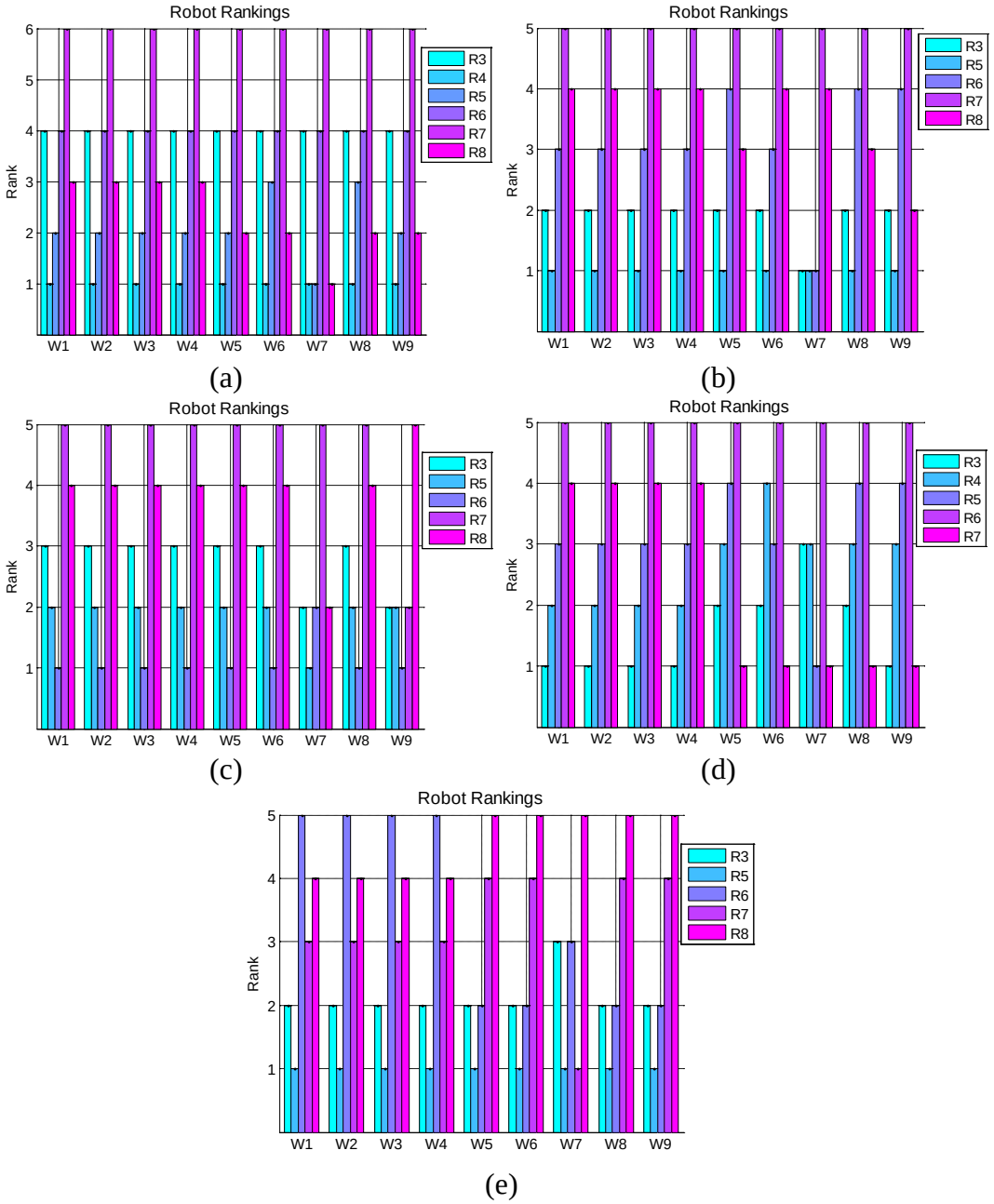


Figure 7.12: Robot rankings for worker task WT2 (explorer).

Thus, it is possible to achieve a unique rank for non-identical robots by applying weightings to planning and communication in addition to sensing and actuation. The explorer task is generally more sensing and actuation intensive than processing and communication. Hence, the weightings of sensing and actuation are expected to be greater that sensing and actuation. Weight set W4 has been selected as it provides low weightings for processing and communication in addition to sensing and actuation.

Weight sets W2 and W3 could also be employed since they produce the same rankings as W4. A better ranking (closer to unity) is achieved for robots with better processing and communication resources when small weightings are provided for these two resources. This can be beneficial since the selected robot(s) may be able to execute the planner task when a planner robot fails during task execution.

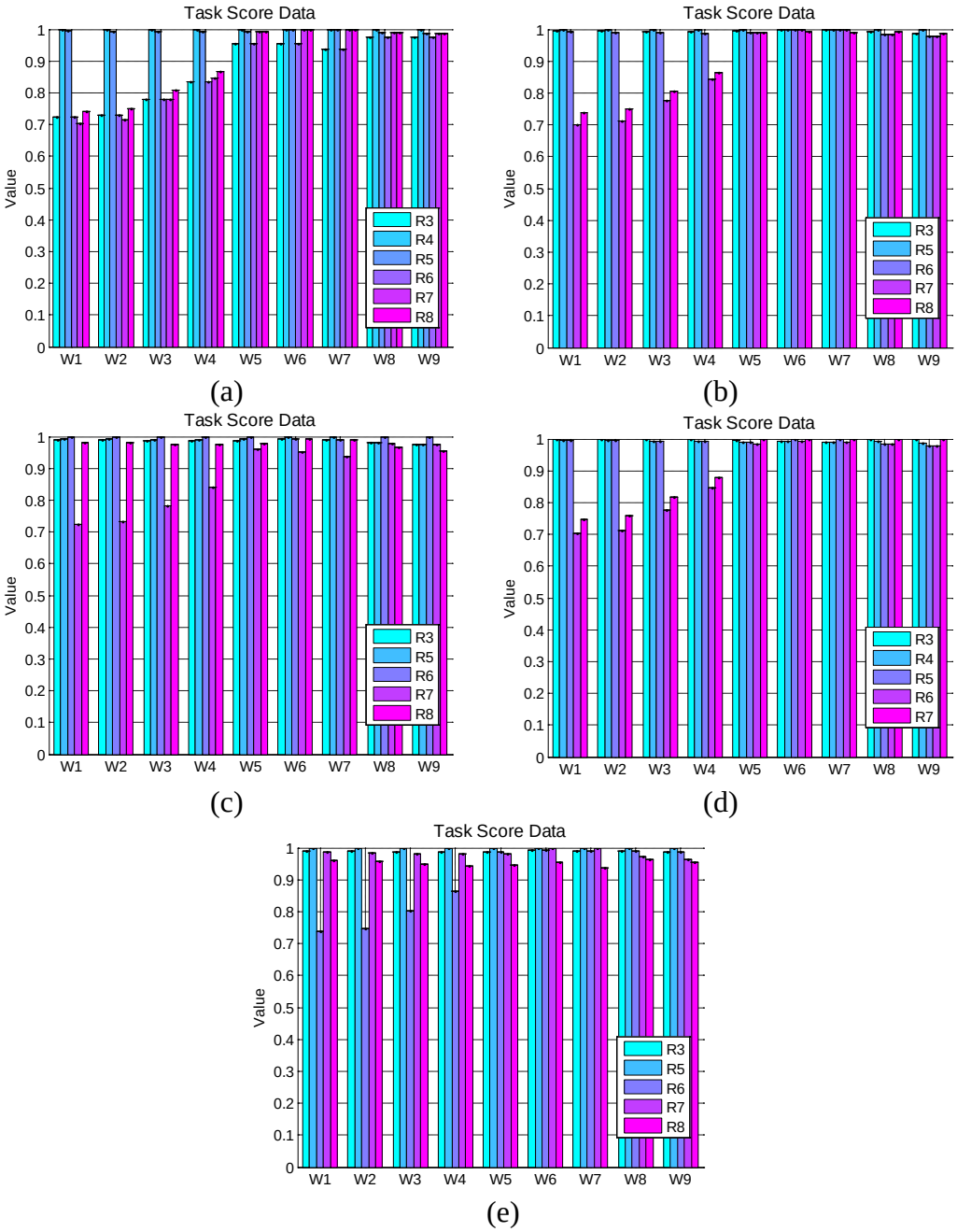


Figure 7.13: Task score data for worker task WT2 (explorer).

Figure 7.13 illustrates the initial task score data for worker task WT2. This data is utilised to bias the allocation of exploration areas to explorer robots in the mapping and exploration task (chapter 6). Section 5.5.1 details the computation of task score

data. Task score data is close to unity (> 0.9) for all robots within each robot set when weighting is given to only sensing and actuation resources (weight sets W5–W9). Giving weighting to planning and communication in addition to sensing and actuation (weight sets W1–W4), produces lower task score data for some robots in each robot set. The robots with lower task scores have weaker processing capabilities than the robots with higher task scores in these weight sets (W1–W4). For example, at W1 in set (a) R3 has a processing capability of 0.38 (from Table 7.4) and a task score of 0.72. But, R4 with a processing capability of 0.86 (also from Table 7.4) produces a task score of 1 at W1 in set (a).Weight set W4 has the least task score reduction amongst weight sets W1–W4 for all five robot sets.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4	x	x	x	x	x	x	x	x	x
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7									
R8	x	x	x	x	x	x	x	x	x

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7	x	x	x	x	x	x	x	x	x
R8	x	x	x	x	x	x	x	x	x

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7	x	x	x	x	x	x	x	x	x
R8	x	x	x	x	x	x	x	x	x

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4	x	x	x	x	x	x	x	x	x
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7	x	x	x	x	x	x	x	x	x
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7	x	x	x	x	x	x	x	x	x
R8	x	x	x	x	x	x	x	x	x

(e)

Figure 7.14: Robots assigned to worker task WT2 (explorer) when five explorers are required.

The multi-robot mapping and exploration task (chapter 6) and feedback coordination mechanism (section 5.6) have been tested on teams comprising five, three, and one explorer(s). Hence, the selection of these quantities of explorer robots is examined. For a team of five explorers, Figure 7.14 reveals that the same five explorers are selected for all weight sets within each robot set.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3									
R4	x	x	x	x	x	x	x	x	x
R5	x	x	x	x	x	x	x	x	x
R6									
R7									
R8	x	x	x	x	x	x	x	x	x

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x		x	x		
R7									
R8					x			x	x

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6	x	x	x	x	x	x	x	x	x
R7									
R8									

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4	x	x	x	x	x			x	x
R5	x	x	x	x		x	x		
R6									
R7					x	x	x	x	x
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x	x	x	x	x	x
R4									
R5	x	x	x	x	x	x	x	x	x
R6					x	x		x	x
R7	x	x	x	x			x		
R8									

(e)

Figure 7.15: Robots assigned to worker task WT2 (explorer) when three explorers are required.

Figure 7.15 details the robots selected when three explorers are required. Weight set W4 selects at least two robots that are also chosen in all other weight sets for each of the five robot sets. Figure 7.15 (a) and Figure 7.15 (c) select the same three robots for

all weight sets. Six out of eight weight sets (including W4) choose the same robots in Figure 7.15 (b). Figure 7.15 (d) and Figure 7.15 (e) select identical robots for four out of eight and five out of eight weight sets (including W4), respectively.

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3									
R4	x	x	x	x	x	x	x	x	x
R5									
R6									
R7									
R8									

(a)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3							x		
R4									
R5	x	x	x	x	x	x		x	x
R6									
R7									
R8									

(b)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3									
R4									
R5							x		
R6	x	x	x	x	x	x		x	x
R7									
R8									

(c)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3	x	x	x	x					x
R4									
R5							x		
R6									
R7					x	x		x	
R8									

(d)

Robot ID	Weight Set								
	W1	W2	W3	W4	W5	W6	W7	W8	W9
R1									
R2									
R3									
R4									
R5	x	x	x	x	x	x	x	x	x
R6									
R7									
R8									

(e)

Figure 7.16: Robots assigned to worker task WT2 (explorer) when one explorer is required.

If a single explorer is to be chosen, the robot with a rank of one in each weight set – robot set combination is the best candidate. Figure 7.16 illustrates this result. The same robot would be selected for all weight sets in robot sets (a) and (e). Robot sets (b) and (c) select the same robot for all weight sets except W7. Three robots (R3, R5, and R6) in set (b) (Figure 7.12 (b)) have an equal highest ranking at W7 due to identical actuation resource scores. Thus the robot with the lowest ID (R3) is selected.

In robot set (c), R5 is selected at weight set W7 since it has the best actuation capabilities.

Robot set (d) (Figure 7.16 (d)) produces mixed results. The better processing capabilities of R3 give it the highest ranking in weight sets W1–W4. When weighting is given to sensing and actuation using weight sets W5–W8, R7 is the best robot since it has the best sensing and actuation resources. Weight set W9 selects R3 as the best robot since it has identical sensing resources to R7 and has a lower identification number.

7.5 Alternative Techniques

Several methods for multi-robot task allocation have been reviewed in section 2.7. A key difference between these methods and the work presented in this chapter is the ability of the developed task allocation strategy to select robots using reduced human user input. Unlike the methods of section 2.7, tasks are specified with graded inputs (such as ‘low’, ‘medium’, or ‘high’) of processing, communication, sensing, and actuation physical resource requirements.

Other methods that utilise robot capabilities and task requirements for multi-robot task allocation are ASyMTRe [21], Vig and Adams [13], and RACHNA [115]. These methods use more complex heuristic greedy strategies (anytime algorithms [144], Shehory and Kraus’ algorithm [114], and market-based reverse auctions [115], respectively) to select robots for tasks. This gives them the ability to select a better team than the developed system for a given task specification, although still not optimal. Tasks need to be specified in a more complex manner in these task allocation methods. Additionally, there is no guarantee that any selected team will remain the best during task execution.

Based on the taxonomy of multi-robot task allocation by Gerkey and Mataric [94], the task allocation method developed in this thesis belongs to the multi-task robots, single-robot tasks (MT-SR) classification. This type of task allocation is uncommon as it assumes that robots can concurrently execute multiple tasks. None of the multi-robot task allocation methods reviewed in section 2.7 are of this type. Most robot

tasks require sensing and actuation components and it is often difficult to utilise these resources simultaneously for multiple tasks. However, as demonstrated in this thesis, computational tasks (such as planning) can be coupled with sensing or actuation tasks (such as exploration). The other methods that use robot capabilities and task requirements for multi-robot task allocation [13, 21, 115] belong to the single-task robots, multi-robot tasks (ST-MR) classification. However, MT-SR and ST-MR problems can be solved using similar algorithms.

If there is only one type of task to be allocated, the developed task allocation method becomes an equivalent of the iterated single-task robots, single-robot tasks (ST-SR) classification. The assignment of tasks to robots then becomes similar to BLE [102], ALLIANCE [12], and M+ [98] algorithms. These instances of the canonical greedy algorithm are known to be 2-competitive for the optimal assignment problem [145]. This means that in the worst case, these algorithms produce a solution whose benefit is half of the optimal benefit. However, anecdotal evidence suggests that the greedy algorithm works extremely well on typical multi-robot task allocation problems [94].

Market-based multi-robot task allocation methods [9, 97, 98, 100, 101, 105] are well suited for situations where there are more tasks than robots. They are suitable for applications where the costs and revenues of tasks can be quantified. An auctioning mechanism assigns tasks based on the bids placed by robots. This process is similar to a canonical greedy algorithm. As mentioned in section 2.7, it can be difficult to determine the revenue and cost functions for robot physical capabilities (such as processing, communication, sensing, and actuation) in market-based multi-robot task allocation. The only method that uses a variant of market-based methods for task allocation using physical robot capabilities [115] recognises this difficulty.

The specification of tasks with graded inputs for non-expert users has a limitation. If the task requirements are incorrectly specified, the initial selection of robots is likely to be inadequate regardless of the search algorithms utilised for the selection process. Due to this limitation, a simple greedy algorithm is utilised instead of more complex meta heuristic algorithms such as genetic algorithms, Tabu search, branch and bound, or pattern search [138]. To address this limitation, a feedback system (evaluated in

chapter 8) is utilised to monitor robot performance for detecting and correcting failures.

7.6 Summary

This chapter has presented results on a task allocation strategy that can be employed by a hierarchical heterogeneous multi-robot system comprising limited capability mobile robots. A multi-robot mapping and exploration task (chapter 6) has been employed as a model task for task allocation experiments on five sets of eight mobile robots. Tasks are specified with reduced (simplified) human user inputs, unlike the more complex specifications of the task allocation methods reviewed in section 2.7. This reduction requires additional effort from an expert user in designing FISs (section 5.3 and section 5.5) to match tasks and robots. The primary task devolution process is able to successfully identify and select suitable robots for manager tasks.

Worker robots are also successfully identified and selected during the secondary task devolution process. Small weightings for sensing and actuation in the planner worker task (W1–W4 in Table 7.1) are able to provide unique ranks for multiple candidate robots. This can also be beneficial since a planner robot may need to be mobile during execution of the mapping and exploration task. Using weightings W1–W4 in Table 7.1 also selects the same robot for the planner task, indicating robustness. This means that the weights can be intuitively selected within this range of values.

Applying small weightings for planning and communication in the explorer task (W1–W4 in Table 7.2) also permits unique ranks to be obtained for multiple candidate robots. Small weightings for planning and communication in the explorer task do not affect robot rankings adversely. When five explorers are required, the same robots in each robot set are selected for all tested weight sets. Weight sets W1–W4 select the same robots for each robot set when three explorers are required. At least two of the three selected robots are identical to those chosen when no weighting is given to planning and communication (W5–W9 in Table 7.2). Similarly, W1–W4 select the same robot for each robot set when one explorer is required. In four of the five tested robot sets, the robot chosen by weight sets W1–W4 is also selected by at least four of the five other weights (W5–W9 in Table 7.2). The small weightings for planning and

communication can also favour the selection of an explorer robot that may be able to execute the planner task when a planner robot fails during task execution. Using weightings W1–W4 in Table 7.2 also selects the same robot(s) for the explorer task, indicating robustness. This means that the weights can be intuitively selected within this range of values.

The robots selected for the global task are the best based on the FISs employed for robot capability data simplification (section 5.3) and the task devolution algorithms (section 5.5). However, this initial selection of robots may not necessarily be optimal or remain the best during task execution. Additionally, tasks can be inaccurately specified by non-expert users resulting in incorrect robot selection. Hence, a feedback coordination mechanism (section 5.6) is employed to monitor the robots after initial task allocation. Chapter 8 presents results of employing the feedback coordination mechanism to monitor robots during task execution.

8 Feedback Coordination Experiments

8.1 Overview

This chapter presents experiments on the feedback coordination technique presented in chapter 5. The multi-robot map-building and exploration task described in chapter 6 is employed as a model global task for the experiments. Section 8.2 presents the feedback coordination experiment configurations. It details the global environments, team configurations, and feedback configurations employed for the experiments presented in this chapter. Section 8.3 details the results of mapping and exploration employing initial task allocations (section 5.5) to robots without any feedback. Results obtained with task score feedback (but without task reallocation) are discussed in section 8.4. Section 8.5 presents the results of employing full feedback (task reallocation) to rectify poor performance, partial failures and complete failures. A summary of the results discussed in this chapter is presented in section 7.6.

8.2 Feedback Coordination Experiment Configurations

Based on a grid map resolution of 0.3 m with 16 square-sized local maps for exploration, a global exploration area size of 38.4 m × 38.4 m has been employed for the feedback experiments. The local map size (8 KB or 8192 Bytes) has been selected based on the execution speed of the MATLAB simulator and the sensing and processing capabilities of worker robots. Selecting larger local map and global map sizes results in lengthy simulation times using MATLAB.

Ten global worlds with randomly positioned obstacles at three obstacle densities (5%, 10%, and 15%) have been generated. An additional ten global worlds with boggy terrain that covers approximately 5% of the global area has been generated for each of these obstacle densities. Figure 8.1 illustrates five worker robots (one planner and five explorers) exploring local environments in a 10% obstacle density world with 5% boggy terrain.

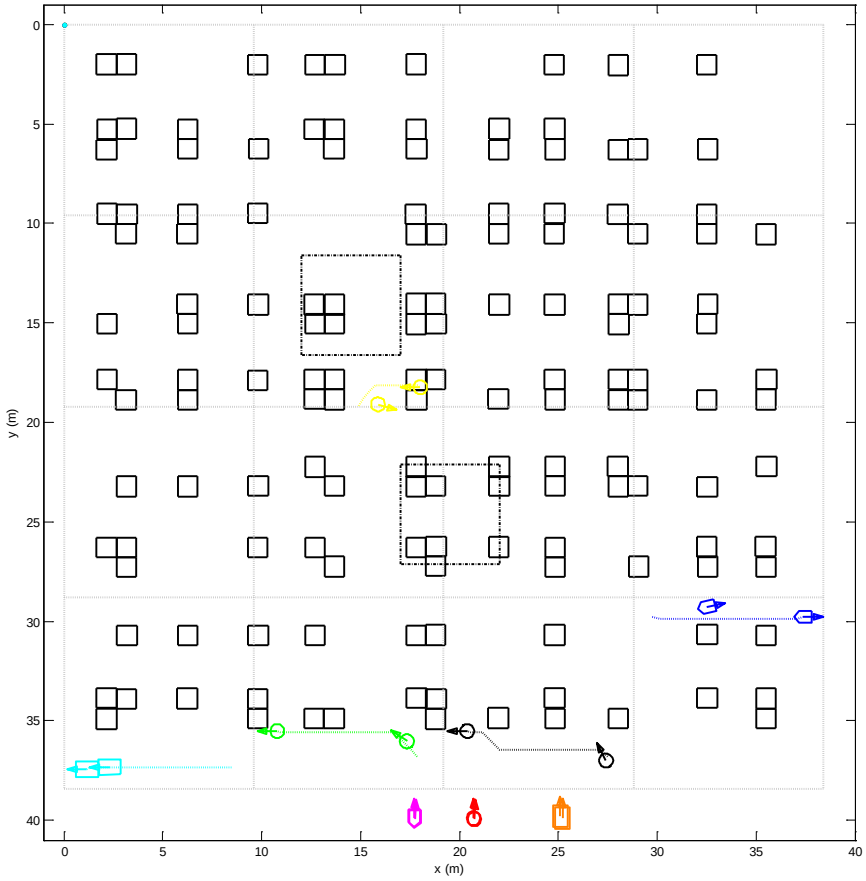


Figure 8.1: Five worker robots exploring a 10% obstacle density world with 5% boggy terrain.

In addition to the various obstacle and bog densities, three robot team configurations have been evaluated. These are single planner – single explorer ([1 1]), single planner – three explorers ([1 3]), and single planner – five explorers ([1 5]). The task and robot descriptions presented in Table 5.20 – Table 5.22 of chapter 5 are employed for experiments in this chapter. In the worker task descriptions, the outputs of the robot quantity criteria FIS (section 5.4) are manually set to achieve the three robot team configurations.

Table 5.29 in chapter 5 lists the feedback weights employed for the worker tasks. Achievement success bias weights *WS*, task execution success threshold TES_T, and overall task execution success threshold $OTES_T$ are tuned for the explorer task (WT2). These parameters are not tuned for the planner task (WT1) since a single planner is employed in all experiments. Task score feedback has no effect on a single planner. If the task requirements are met, a planner will function unless a complete failure occurs.

Table 8.1: Summary of feedback experiment configurations.

Test Level	Sub-Level	Feedback Type	Planners and Explorers [max_planner max_explorer]	Obstacle Density %	Bog Density %	Poor Performance PP (Unforced Failures)	Partial Failure PF (Forced Failures)	Complete Failure CF (Forced Failures)	PP OTES Threshold	PF TES Threshold	Monitor Time Tm (sec)	Achievement Bias Weights
1	1.1	No Feedback	[1 1] [1 3] [1 5]	5 10 15	0 5	Yes	No	No	-	-	-	-
	1.2	No Feedback	[1 3] [1 5]	5 10 15	0 5	Yes	Yes explorer failure only	No	-	-	-	-
	1.3	No Feedback	[1 3] [1 5]	5 10 15	0 5	Yes	No	Yes explorer & planner failure	-	-	-	-
2	2	Task Score Feedback only	[1 3] [1 5]	5 10 15	0 5	Yes	No	No	-	-	60 180 300	W1 = [0.0,0.5,0.5] W2 = [0.1,0.1,0.4,0.4] W3 = [0.2,0.2,0.3,0.3] W4 = [0.25,0.25,0.25,0.25]
3	3.1	Full Feedback Task Reallocation Poor Performance	[1 1] [1 3] [1 5]	5 10 15	0 5	Yes explorer only	No	No	P1 = 0.75 P2 = 0.65 P3 = 0.55 P4 = 0.45	0.15	60 180 300	Use best value from level 2
	3.2	Full Feedback Task Reallocation Partial Failure	[1 3] [1 5]	5 10 15	0 5	Yes	Yes explorer failure only	No	Use best value from level 3.1	F1 = 0.05 F2 = 0.15 F3 = 0.2 F4 = 0.25	60 180 300	Use best value from level 2
	3.3	Full Feedback Task Reallocation Complete Failure	[1 3] [1 5]	5 10 15	0 5	Yes	No	Yes explorer & planner failure	Use best value from level 3.1	Use best value from level 3.2	60 180 300	Use best value from level 2
	3.4	Full Feedback Task Reallocation Overall	[1 1] [1 3] [1 5]	5 10 15	0 5	Yes	No	No	Use best value from level 3.1	Use best value from level 3.2	60 180 300	Use best value from level 2

Table 8.1 summarises the feedback experiment configurations employed in this chapter. There are three test levels. Level 1 executes the mapping and exploration task without any feedback. Levels 2 and 3 evaluate performance with feedback. Feedback is tested using three monitor time interval values: T_m = 60 sec, T_m = 180 sec, and T_m = 300 sec. The monitor time values have been selected based on communication bandwidth constraints.

Task score feedback is assessed in level 2. The achievement bias weights (section 5.6.1) are employed to combine the achievement data of the four resource types (processing, communication, sensing, and actuation) into a task execution success (*TES*) value. Task score values are computed from *TES* data (section 5.6.1). An explorer's primary function is to traverse a local map and collect sensor data for the global map. Thus, greater weighting should be given to sensing and actuation achievement data. Weight sets W1–W3 in Table 8.1 give greater weighting to sensing and actuation when compared to processing and communication. At weight set W4 equal weighting is given to all four resource types.

Full feedback to correct poor performance (*PP*), partial failures (*PF*), and complete failures (*CF*) is tested in level 3. Poor performance is considered an unforced failure in the experiments as it arises due to interaction with the global environment. Four $OTES_T$ values (P1–P4 in Table 8.1) are heuristically tested to detect poor performance. Higher $OTES_T$ values (closer to one) will result in faster detection of poor performance. Poor performance detection and correction is fully explained in section 5.6.1.

Partial and complete failures have been programmed to occur as the global task executes. Hence, they are considered to be forced failures. These failures are present on single robots. Partial failures permanently impair a robot's sensing or actuation performance but the device remains operational. In a complete failure, all contact with the robot is permanently lost due to processing or communication failure. These two types of failures are fully explained in section 5.6.1. Four TES_T values (F1–F4 in Table 8.1) are heuristically tested to detect partial failures. Lower TES_T values (closer to zero) result in slower detection of partial failures. However, if TES_T is high, false detections of partial failures are possible.

Testing has been carried out in the following steps:

1. Execute all level 1 tests (no feedback experiments).
2. Run level 2 tests with monitor time interval T_m = 60 sec. Test the achievement bias weights W1–W4 detailed in the last column of Table 8.1. (T_m = 60 sec is the fastest update rate of the three tested monitor time values. Hence, it should be the most reactive during task execution.)
3. Evaluate the results obtained from step 2 and select the best set of achievement bias weights.
4. Execute level 2 tests with T_m = 180 sec and T_m = 300 sec. Use the best achievement bias weight set determined from step 3.
5. Run level 3.1 tests with T_m = 60 sec. Test poor performance (unforced failures) with the *PP OTES* threshold ($OTES_T$) values P1–P4 listed in Table 8.1. Set the *PF TES* threshold (TES_T) to a constant value (0.15) for these experiments. The *PF* TES_T value is to be tuned in level 3.2. (*OTES* data are obtained by integrating *TES* values over T_m so it is appropriate to use the smallest T_m value.)
6. Evaluate the results obtained from step 5 to determine the best *PP OTES* threshold value.
7. Execute level 3.1 tests with T_m = 180 sec and T_m = 300 sec. Use the best *PP OTES* threshold value determined in step 6.
8. Run level 3.2 tests with T_m = 60 sec. Test partial failures with the *PF TES* threshold (TES_T) values F1–F4 listed in Table 8.1. Set the *PP OTES* threshold to the best value obtained in step 6. (T_m = 60 sec is the fastest update rate and produces the most accurate *TES* value.)
9. Evaluate the results obtained from step 8 and select the best *PF TES* threshold value.
10. Execute level 3.2 tests with T_m = 180 sec and T_m = 300 sec. Use the best *PF TES* threshold value obtained in step 9.
11. Run level 3.3 complete failure tests with the best *PP OTES* and *PF TES* threshold values.
12. Execute level 3.4 overall feedback tests. Test poor performance (unforced failures) with the best *PP TES* and *PF TES* threshold values.

Note that the above twelve steps are not part of an auto-tuning process. Instead, they just list the tests and experiments conducted to evaluate the response weight and threshold variation. A similar procedure can be employed to evaluate the feedback system for other global task types.

A maximum exploration time (*MET*) of 15000 sec (250 min) is permitted for the single planner – single explorer configuration. For the single planner – three explorer and single planner – five explorer configurations an *MET* of 5000 sec (83.3 min) is allowed. These times have been determined from initial experiments to permit completion of exploration in most situations when feedback is employed. Exploration performance is assessed based on exploration time (*ET*) and percent area explored (*PAE*). An exploration rate value (*ER*) is calculated from *ET* and *PAE* (8.1).

$$ER = \frac{PAE}{ET} \tag{8.1}$$

Each experiment is repeated ten times due to simulation time length constraints. Since ten environments with evenly spaced obstacles are automatically generated for each environment type, a total of one hundred experiments are conducted in each type of environment. In the results figures, each bar represents the average value and the corresponding error bar illustrates standard deviation. A paired sample t-test with two-sided p-values is used to compare the feedback and non-feedback experiment data. Comparisons are statistically significant if p-values are less than or equal to 0.05 (5% statistical significance level). Exploration performance parameter ratios are computed to determine superiority or inferiority by dividing the feedback experiment data by the non-feedback experiment data.

8.3 Experiments without Feedback

Figure 8.2 illustrates the exploration time, area explored, and exploration rate score for exploration without any feedback. Generally, exploration time reduces when more explorers are employed (Figure 8.2 (a)). The percentage area explored improves as more explorers are utilised (Figure 8.2 (b)). A maximum mean area of approximately 80% is explored for the [1 1] team configuration, which is less than the minimum mean area explored when three ([1 3]) or five ([1 5]) explorers are utilised.

Exploration time is very close to the *MET* values and percentage area explored is generally lower at 15% obstacle density (as expected). This indicates that the robots have not been able to complete exploration of the global environment. The exception to this trend is when five explorers are employed in a 15% obstacle density environment without any bogs. In this case, the additional explorers in the five explorer team are sufficient to complete exploration before the *MET*.

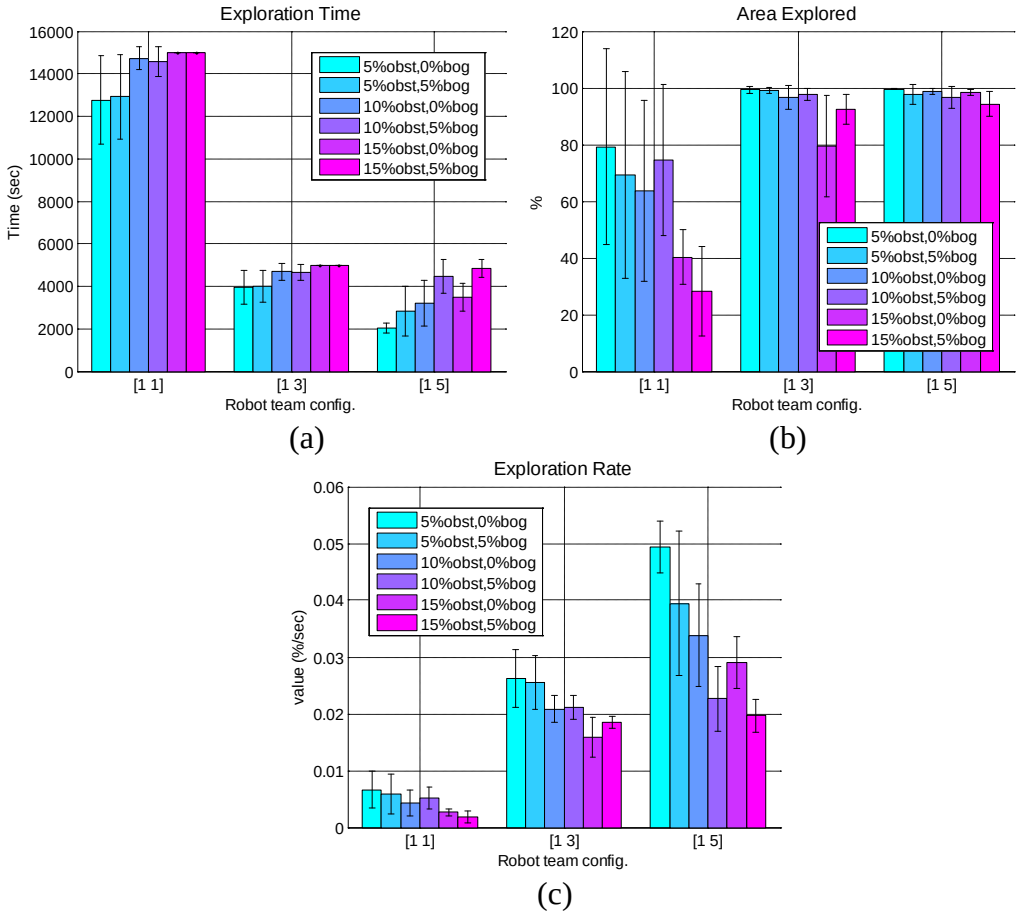


Figure 8.2: Results of exploration without feedback for the various robot team – environment combinations.

The trends of exploration time and area explored are combined in the overall exploration rate data (Figure 8.2 (c)) (8.1). As expected, the highest overall scores are obtained when five explorers are employed while the lowest values are obtained when only one explorer is utilised. Figure 8.3 combines the data for individual environments in Figure 8.2 into single values for each robot team configuration. This is achieved by collating all data samples and computing overall mean and standard deviation values. An easier comparison can made between the robot team

configurations by doing this. It also permits easier comparison with the various feedback experiments discussed in section 8.4 and section 8.5.

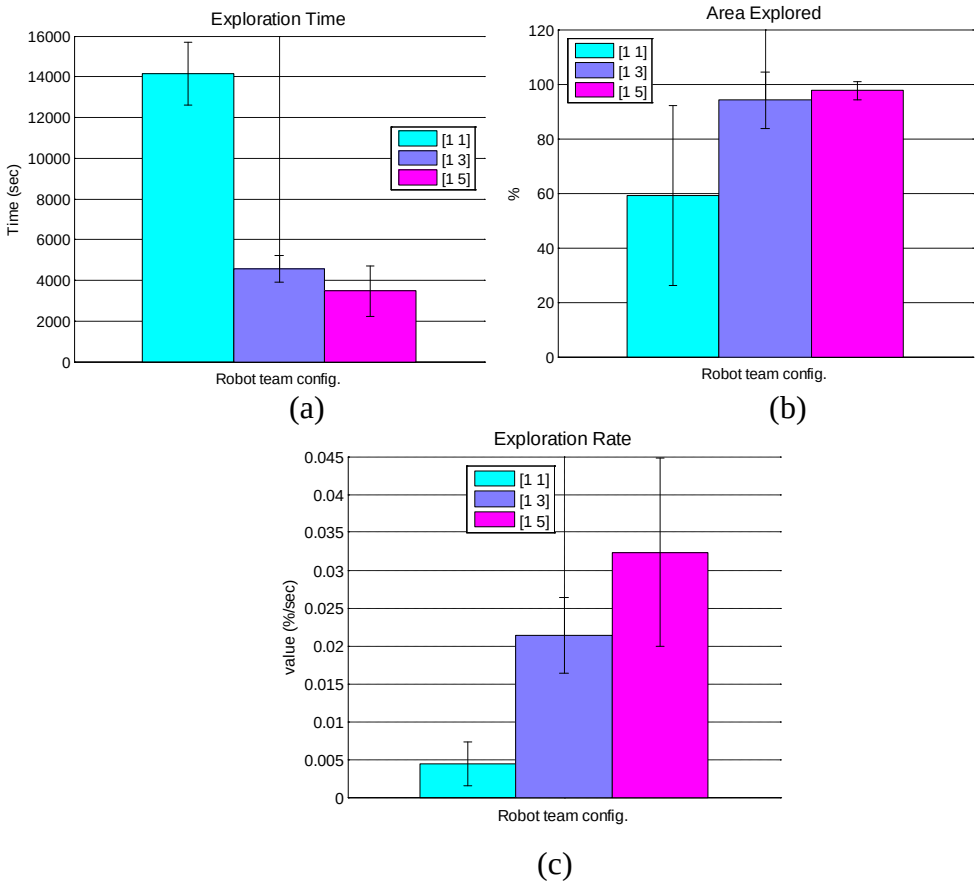


Figure 8.3: Reduction of non-feedback exploration results into single values.

Figure 8.4 compares the performance of the [1 3] and [1 5] robot teams with the [1 1] robot team over all tested environments. Comparisons are made by dividing the [1 3] and [1 5] robot team data by the [1 1] robot team data. Exploration time is reduced to 32.1% and 24.5% when three and five explorers are employed, respectively (Figure 8.4 (a)). Figure 8.4 (b) reveals that 58.9% and 64.8% greater area is explored when the [1 3] and [1 5] robot teams are utilised respectively. Exploration rates are 4.80 and 7.28 times greater for the three and five explorer team configurations, respectively (Figure 8.4 (c)). A 5% statistical significance test produced zero p-values for all comparisons made with the [1 1] robot team.

Comparing the [1 3] and [1 5] robot teams also produces statistically significant results at the 5% level. Results such as this will not generally be plotted in this chapter since these are a side issue from the chapter's primary objective, which is a comparison of the non-feedback and feedback experiments. Exploration time is

reduced to 76.3% when five explorers are employed instead of three. A small increase in exploration area (3.6%) is achieved when five explorers are utilised. Hence, the exploration rate is improved by 52% when five explorers are employed instead of three. The p-values for exploration time and exploration rate are zero. A p-value of 0.0001 has been obtained for the exploration area comparison.

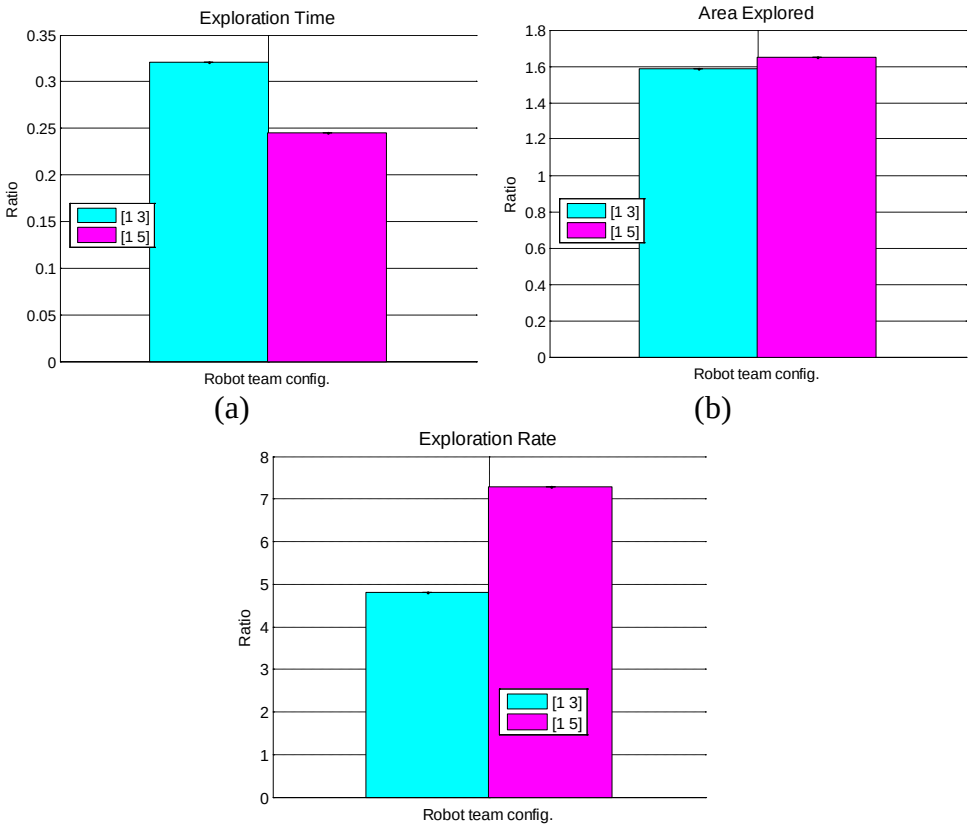
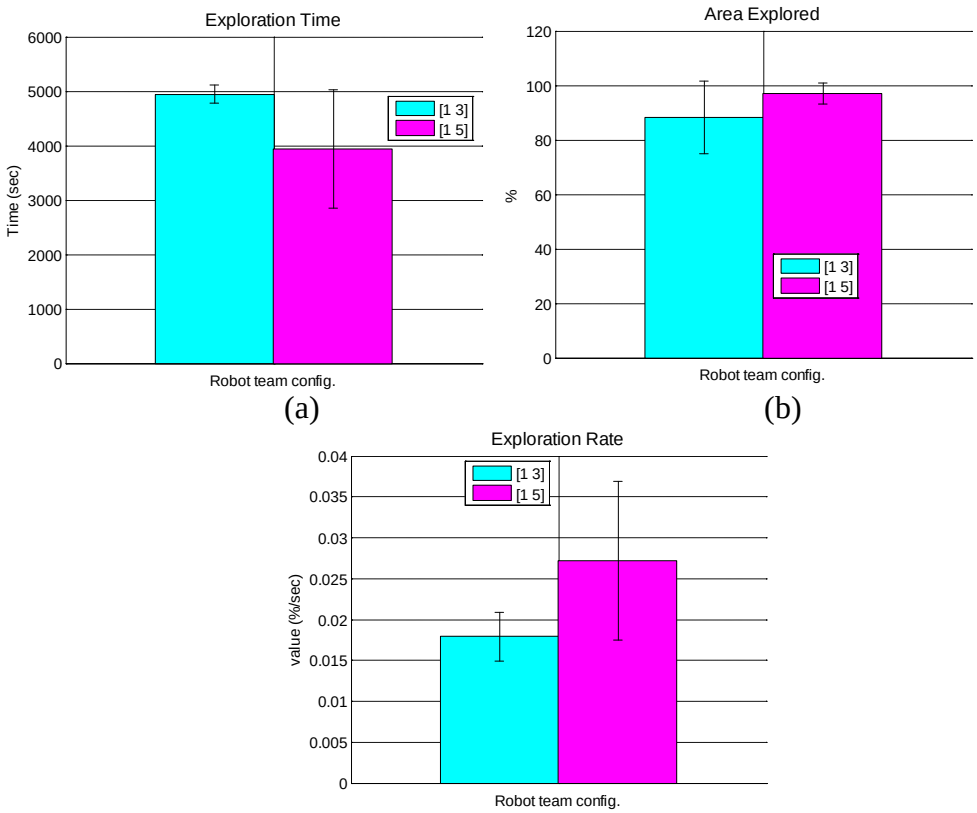


Figure 8.4: Comparison of [1 3] and [1 5] robot team configurations with the [1 1] robot team configuration.

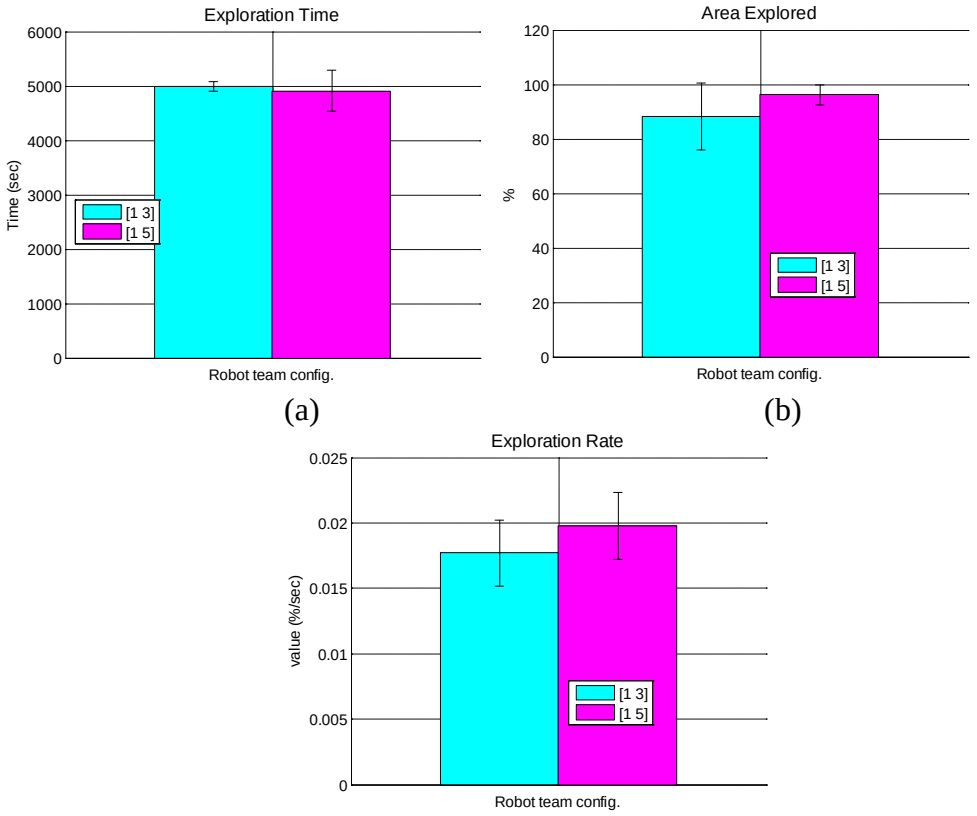
Figure 8.5 and Figure 8.6 illustrate the results obtained when partial failures and complete failures are present in the non-feedback experiments, respectively. The [1 1] robot team has been omitted from the tests since it will be impossible to complete the global task without feedback when a partial or complete failure occurs on a robot. An example of a partial failure is the situation where a single explorer collides with an obstacle or otherwise stops moving. Complete failures result in the single planner failing. In the [1 3] and [1 5] robot teams, partial and complete failures are present on single robots. The results presented in Figure 8.5 and Figure 8.6 are statistically compared with the full feedback experiments presented in section 8.5.



(a) (b)

(c)

Figure 8.5: Non-feedback exploration results with partial failures.



(a) (b)

(c)

Figure 8.6: Non-feedback exploration results with complete failures.

8.4 Experiments with Task Score Feedback

Figure 8.7 illustrates the results obtained when task score feedback is employed for the [1 3] and [1 5] robot team configurations with a monitor time T_m = 60 sec. Table 8.1 details the numerical values of achievement weight sets W1–W4. There is little variation in the exploration time, area explored, and overall exploration rate when the achievement weight sets are varied from W1 to W4 (Figure 8.7 (a)–(c)). Inspecting the exploration rate data reveals that weight sets W2 and W3 have the highest values. This suggests that they are *potentially* the best weight sets to employ. Task score feedback is not assessed for the [1 1] robot team configuration since it will not alter task execution.

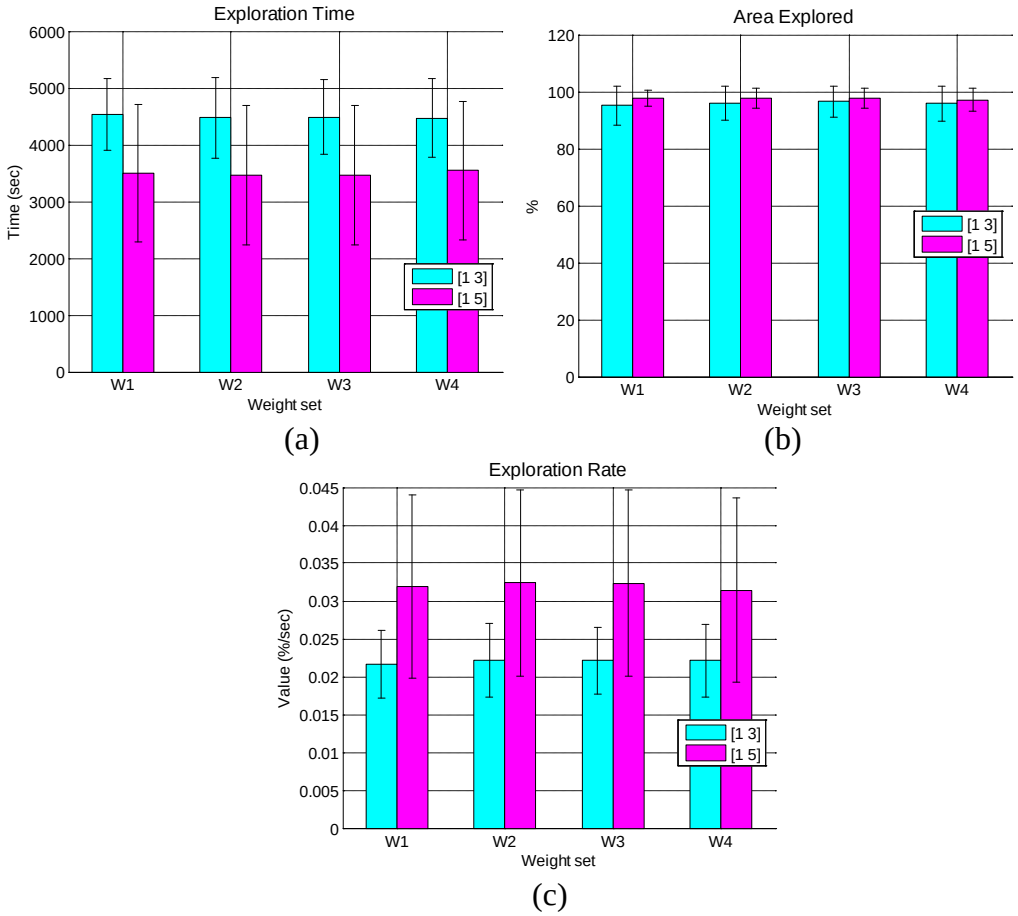
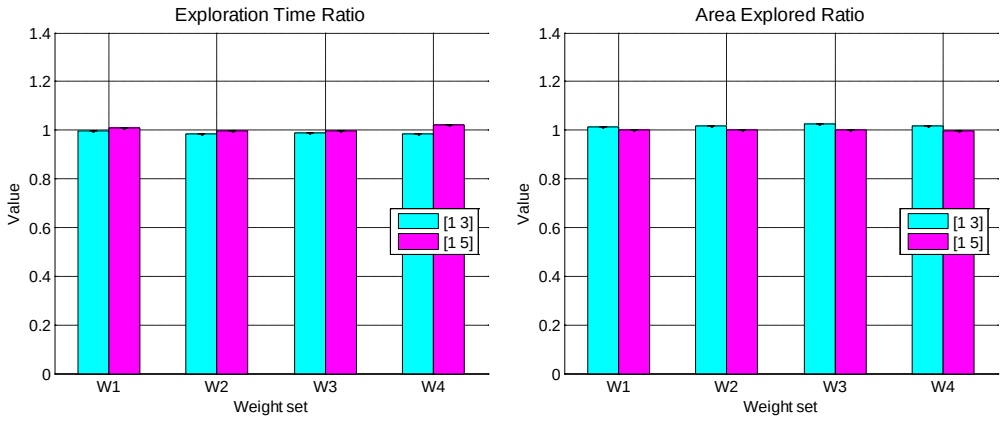


Figure 8.7: Results of exploration with task score feedback.

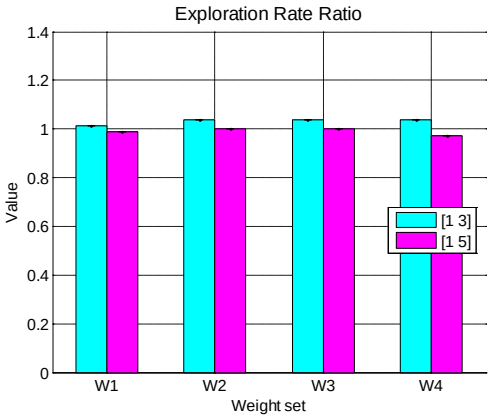
To verify whether W2 or W3 is the best weight set to further consider, Figure 8.7 is statistically compared with the non-feedback experiments of Figure 8.3. Figure 8.8 details the results of the comparison. A comparison with a p-value less than or equal to 0.05 is statistically significant. Achievement weight sets W2 and W3 have mean

exploration time ratio values less than unity for both robot team configurations (Figure 8.8 (a)). However, all exploration time ratio comparisons are not statistically significant (Figure 8.8 (d)).



(a)

(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)					
	Robot Config.	Weight			
		W1	W2	W3	W4
Exploration Time (a)	[1 3]	0.39	0.17	0.22	0.15
	[1 5]	0.42	0.47	0.49	0.30
Area Explored (b)	[1 3]	0.14	0.03	< 0.01	0.04
	[1 5]	0.37	0.33	0.39	0.15
Exploration Rate (c)	[1 3]	0.30	0.07	0.07	0.07
	[1 5]	0.39	0.48	0.50	0.26

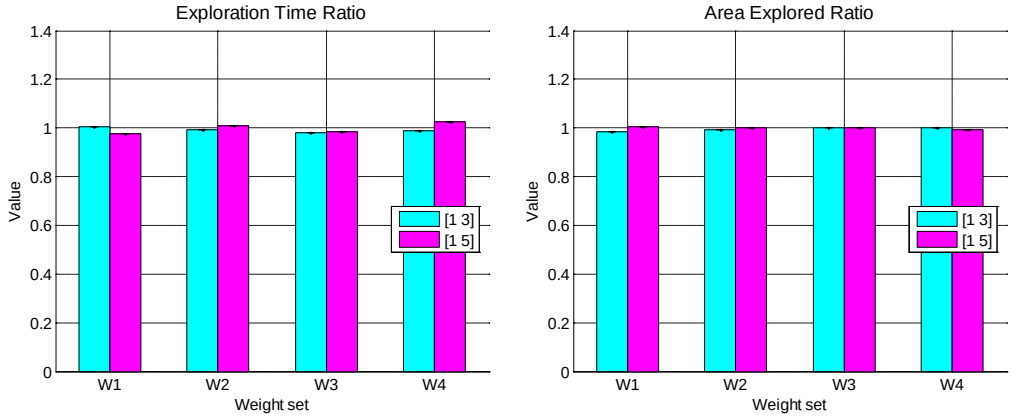
(d)

Figure 8.8: Comparison of task score feedback and non-feedback results at T_m = 60 sec.

When task score feedback is used, the area explored is not less than the area explored without feedback (Figure 8.8 (b)). However, the comparison is only statistically significant for weight sets W2–W4 in the [1 3] robot team configuration (Figure

8.8 (d)). Weight set W3 has the lowest p-value and increases the area explored by the greatest percentage (2.6%). The exploration rate ratios for weight sets W2 and W3 are not statistically significant (Figure 8.8 (c),(d)). Based on the comparisons, weight set W3 has been selected as the best weight set for task score feedback since it achieves a statistically significant improvement in area explored.

The 2.6% increase in area explored at W3 is small. Task score feedback does not appear to be sensitive to weight variation when at least 25% weighting is given to both sensing and actuation resources. This indicates robustness. Task score feedback is able to send good explorer robots to distant unexplored local environments without negatively affecting exploration performance.



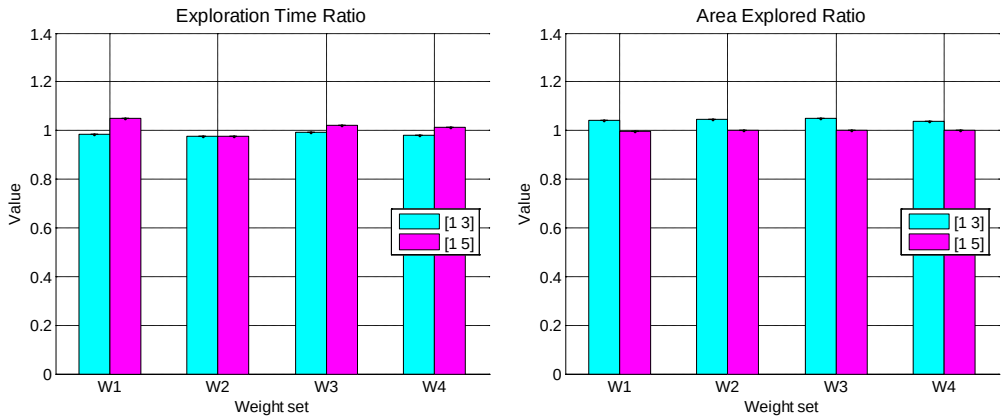
(a) (b)

p-values (two decimal places, < 0.05 shaded blue)								
Robot Config.	Exploration Time (a)				Area Explored (b)			
	W1	W2	W3	W4	W1	W2	W3	W4
[1 3]	0.41	0.37	0.18	0.30	0.04	0.20	0.45	0.45
[1 5]	0.32	0.41	0.37	0.29	0.24	0.38	0.40	0.13

(c)

Figure 8.9: Comparison of task score feedback and non-feedback results for non-boggy environments.

To evaluate if task score feedback offers any statistically significant improvement to the [1 5] robot team, separate comparisons have been made for boggy environments and non-boggy environments. Figure 8.9 illustrates the comparison for non-boggy environments. There is no statistically significant variation in exploration time (Figure 8.9 (a),(c)). Except when weight set W1 is employed for the [1 3] robot team configuration, there is also no statistically significant difference in area explored (Figure 8.9 (b),(c)).



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)								
Robot Config.	Exploration Time (a)				Area Explored (b)			
	W1	W2	W3	W4	W1	W2	W3	W4
[1 3]	0.28	0.16	0.41	0.20	0.02	0.01	< 0.01	0.03
[1 5]	0.19	0.34	0.36	0.40	0.19	0.17	0.38	0.43

(c)

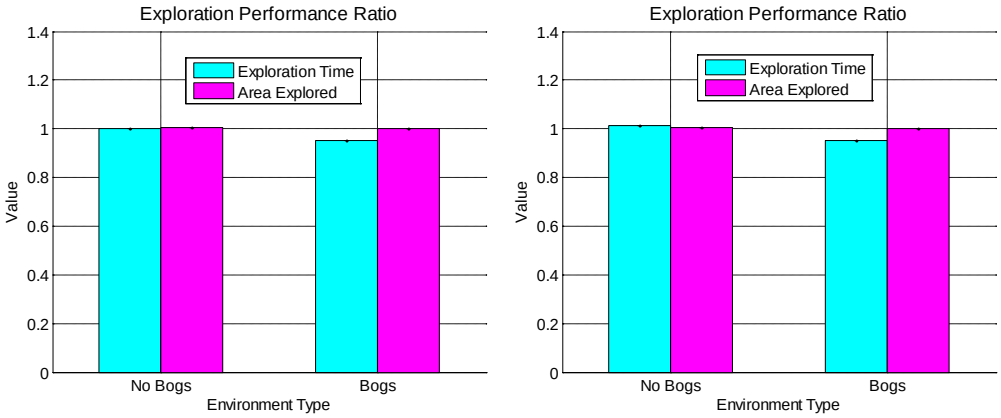
Figure 8.10: Comparison of task score feedback and non-feedback results for environments with boggy terrain.

Figure 8.10 shows the comparison for global environments with boggy terrain. Task score feedback has statistically significant benefits in environments with boggy terrain (Figure 8.10 (b),(c)). It can increase the area explored by up to 5.2% for the [1 3] team in the tested global environments (W3 in Figure 8.10 (b)). However, there is no statistically significant reduction in exploration time for the [1 3] team with task score feedback (Figure 8.10 (a),(c)). No statistically significant improvement is recorded in the [1 5] team configuration.

As mentioned previously, the small variation in performance over different weight sets indicates that task score feedback is robust to weight variation. In non-boggy terrain environments, sending good explorer robots to distant unexplored local environments does not negatively affect exploration performance. By utilising task score feedback, a small robot team (such as the [1 3] team) can improve area explored in environments comprising sections of boggy terrain.

It is possible that the global world is relatively small in comparison to the number of robots deployed for exploration in the [1 5] team. Hence, the [1 5] team has been

tested in two larger environments comprising 25 and 36 local maps, respectively. Similar to the 16 local map environment results above, no statistically significant improvement is recorded in the 25 and 36 local map environments. Figure 8.11 illustrates the comparison of task score feedback and non-feedback experiments when weight set W3 is employed.



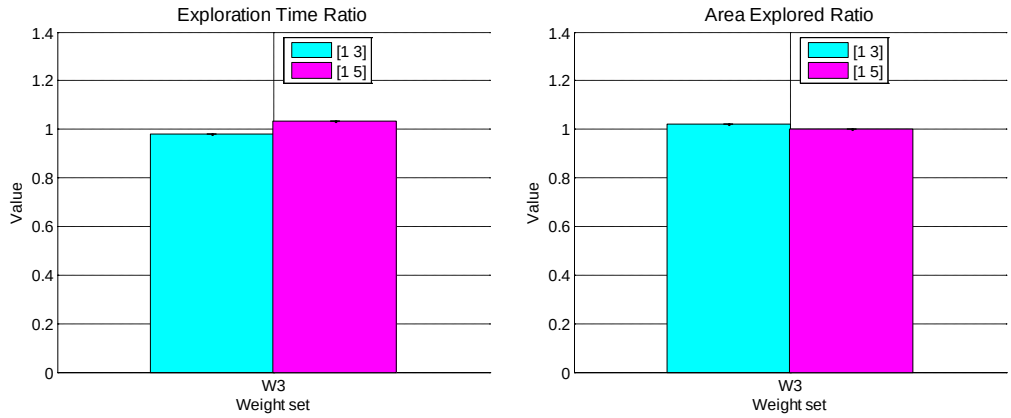
(a) (b)

p-values (two decimal places, < 0.05 shaded blue)				
Exploration Performance	25 Local Maps (a)		36 Local Maps (b)	
	No Bogs	Bogs	No Bogs	Bogs
Exploration Time	0.48	0.25	0.11	0.19
Area Explored	0.22	0.22	0.31	0.14

(c)

Figure 8.11: Comparison of task score feedback and non-feedback results for the [1 5] team in a 25 local map world (a) and a 36 local map world (b).

Task score feedback aids the dispersion of robots in the global environment (section 6.3.1). It sends good explorers to local environments that may be distant from their currently allocated local environment. On the other hand, weak explorers generally navigate to the closest adjacent local map, similar to the non-feedback experiments. In the [1 5] team configuration, the distance traversed by good explorers to reach new unexplored local maps negates the reservation of unexplored space around poor explorers as indicated in Figure 8.11.



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)		
Robot Config.	**Exploration Time (a)**	**Area Explored (b)**
	W3	**W3**
[1 3]	0.12	0.02
[1 5]	0.20	0.46

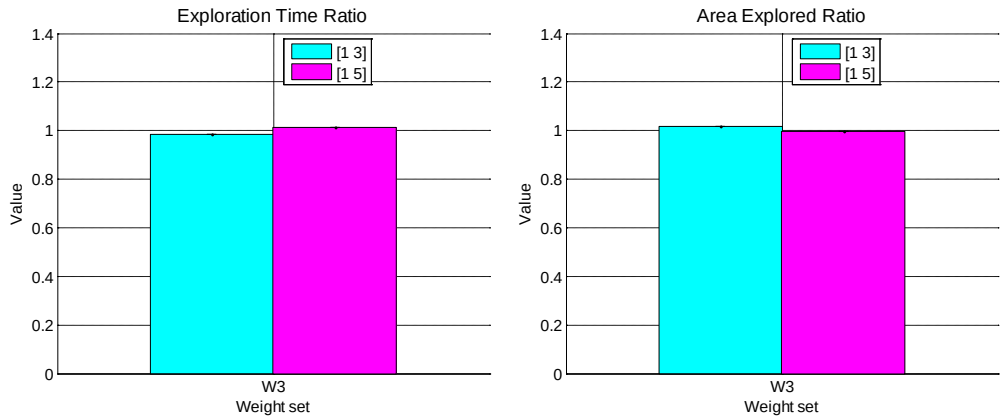
(c)

Figure 8.12: Comparison of task score feedback and non-feedback results at T_m = 180 sec.

The performance of the best weight set, W3, has been evaluated for two additional monitor time values (T_m = 180 sec and T_m = 300 sec). Figure 8.12 shows a comparison of exploration with task score feedback at T_m = 180 sec and exploration without any feedback. The result is similar to W3 in Figure 8.8 (T_m = 60 sec). A statistically significant increase in area explored (2.1%) is achieved for the [1 3] robot team configuration (Figure 8.12 (b),(c)).

Figure 8.13 compares exploration with task score feedback at T_m = 300 sec and exploration without any feedback. Exploration with task score feedback using T_m = 300 sec (Figure 8.13) also offers similar improvements to exploration using T_m = 60 sec (Figure 8.8) when compared to exploration without any feedback. The [1 3] robot team configuration achieves an increase of 1.9% in area explored that is statistically significant (Figure 8.13 (b),(c)).

This is another indication that task score feedback is robust and able to perform at varying monitor time intervals.



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)		
Robot Config.	Exploration Time (a)	Area Explored (b)
	W3	W3
[1 3]	0.20	0.04
[1 5]	0.39	0.34

(c)

Figure 8.13: Comparison of task score feedback and non-feedback results at T_m = 300 sec.

8.5 Experiments with Full Feedback

8.5.1 Poor Performance Experiments

Figure 8.14 shows the results obtained when poor performance feedback is employed for the [1 1], [1 3], and [1 5] robot team configurations using a monitor time T_m = 60 sec. Numerical values for threshold values P1–P4 are given in Table 8.1. At threshold value P4, exploration time is greater for all three robot team configurations than at threshold P1 (Figure 8.14 (a)). Exploration time increases for the [1 1] robot team configuration as the threshold value is changed from P1–P4. This result is expected since poor performance is detected more rapidly when the threshold value is higher. The trend is particularly evident for the [1 1] team since the single explorer system lacks robustness to failures when compared the multiple explorer configurations ([1 3] and [1 5] teams).

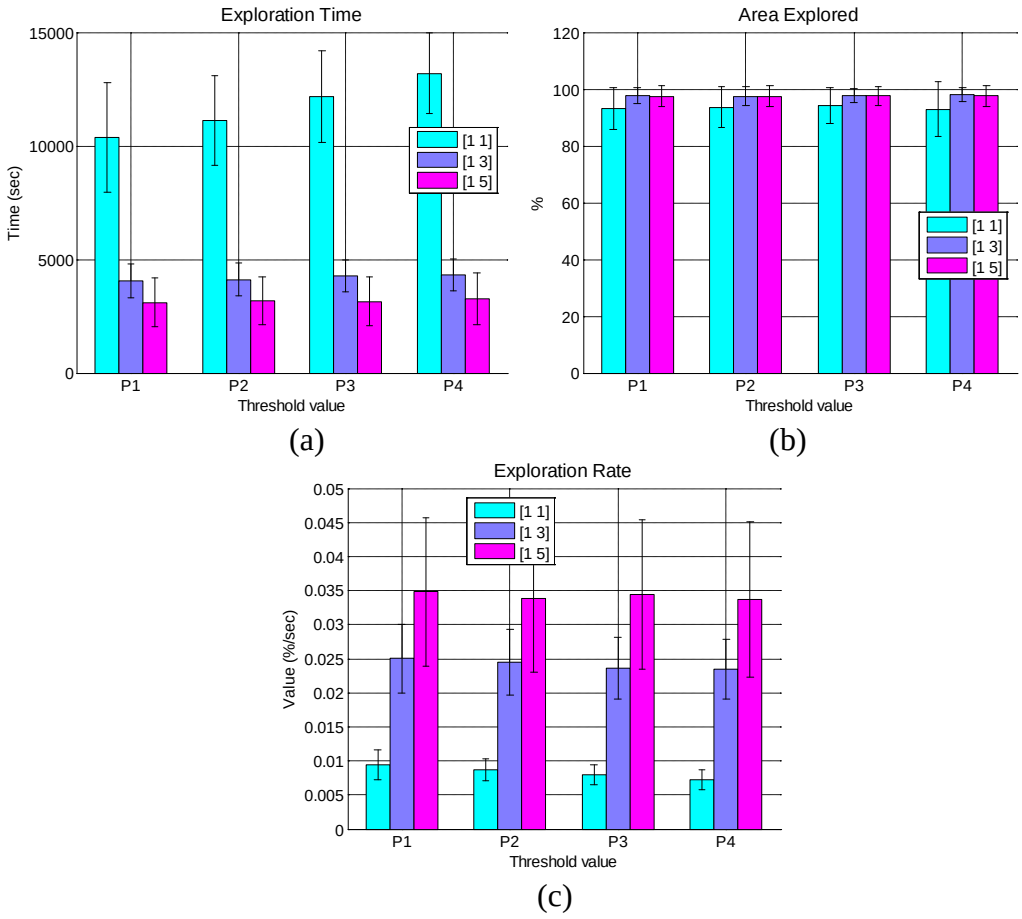
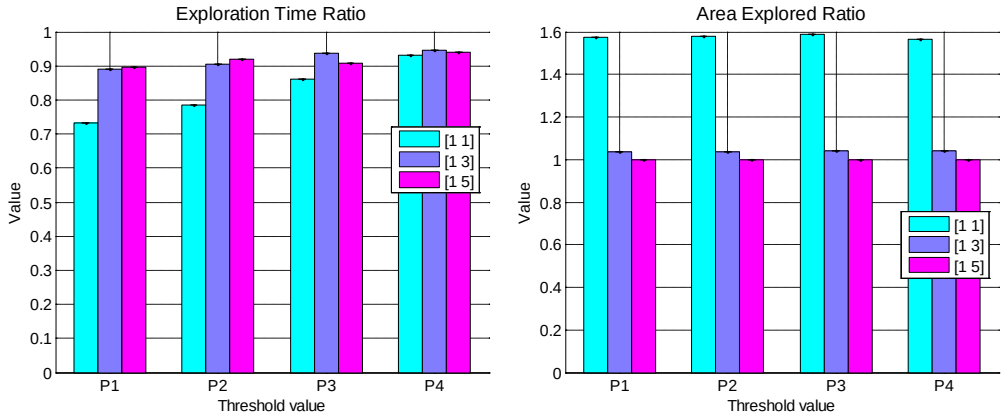


Figure 8.14: Results of exploration with poor performance feedback.

There is little variation in percentage area explored for threshold values P1–P4 (Figure 8.14 (b)). The exploration rate obtained using threshold value P1 is greatest for all robot team configurations (Figure 8.14 (c)). This indicates that P1 likely to be the best threshold value to employ.

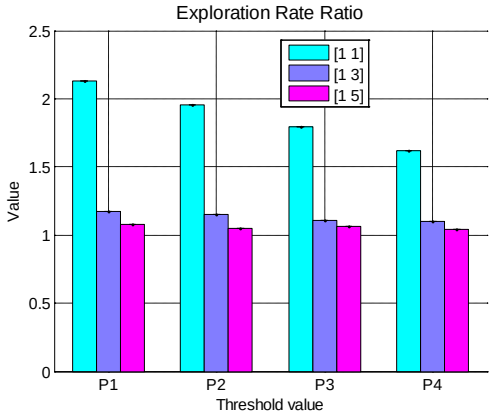
A statistical significance comparison has been made between the poor performance feedback and non-feedback exploration results in Figure 8.15. Maximum reduction in exploration time is achieved at threshold value P1 (Figure 8.15 (a)). Exploration time is reduced to 73.3%, 89%, and 89.5% for the [1 1], [1 3], and [1 5] robot teams, respectively at P1 (Figure 8.15 (a)). All reductions are statistically significant with p-values less than 0.05 except for the [1 5] robot team at P4 (Figure 8.15 (d)). Maximum improvement in area explored (59%) is obtained at threshold value P3 for the [1 1] robot team (Figure 8.15 (b)). For the [1 3] team, a statistically significant improvement of approximately 4% is attained for all threshold values (Figure 8.15 (b),(d)). The results of the area explored by the [1 5] robot team is similar to the

non-feedback experiments (Figure 8.15 (b),(d)) because five explorers are able to complete exploration within the specified *MET* (5000 sec) in both cases.



(a)

(b)



(c)

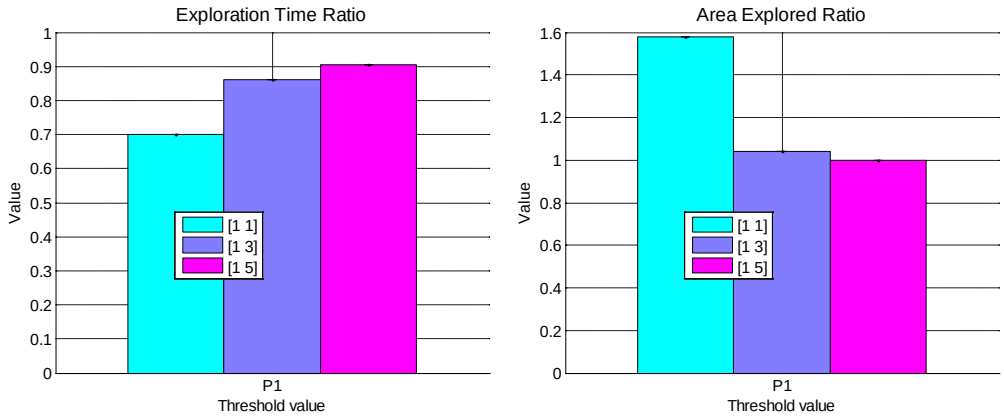
p-values (two decimal places, < 0.05 shaded blue)					
	Robot Config.	Threshold Value			
		P1	P2	P3	P4
Exploration Time (a)	[1 1]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	0.02	< 0.01	0.06
Area Explored (b)	[1 1]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	0.40	0.38	0.47	0.47
Exploration Rate (c)	[1 1]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	0.03	0.13	0.06	0.16

(d)

Figure 8.15: Comparison of poor performance feedback and non-feedback results at T_m = 60 sec.

Threshold value P1 has been selected as the best since it has the highest exploration rate improvement for all robot team configurations when compared to the non-

feedback experiments (Figure 8.15 (c)). This comparison is statistically significant (Figure 8.15 (d)). It reduces exploration time to 73.3%, 89% and 89.5% for the [1 1], [1 3], and [1 5] robot teams, respectively. Area explored is improved by 57.4%, and 3.75% for the [1 1] and [1 3] robot teams, respectively. Hence exploration rate is improved by 113%, 17.2% and 7.8% for the [1 1], [1 3], and [1 5] robot teams, respectively.



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)		
Robot Config.	Exploration Time (a)	Area Explored (b)
	P1	P1
[1 1]	< 0.01	< 0.01
[1 3]	< 0.01	< 0.01
[1 5]	< 0.01	0.43

(c)

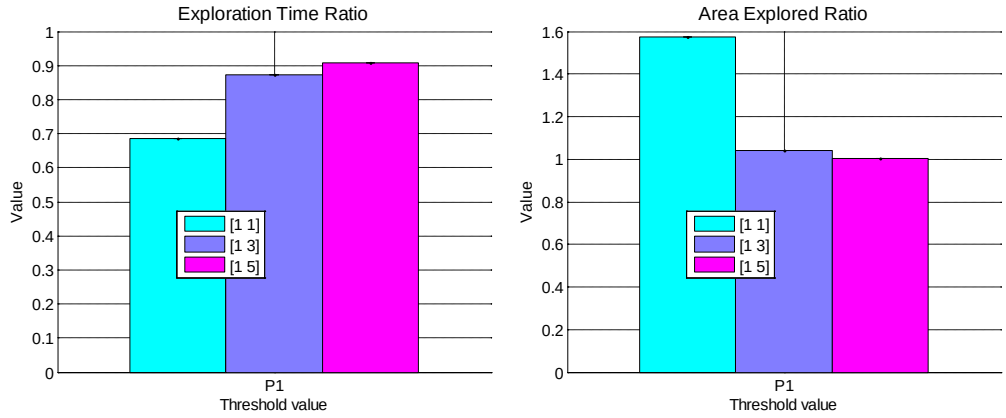
Figure 8.16: Comparison of poor performance feedback and non-feedback results at T_m = 180 sec.

Similar to the task score feedback experiments, the performance of the best threshold value (P1) has been evaluated for T_m = 180 sec and T_m = 300 sec. Figure 8.16 illustrates the comparison of exploration with poor performance feedback at T_m = 180 sec and exploration without any feedback. A similar result to P1 in Figure 8.15 has been obtained. Exploration time is reduced to 70.1%, 86.1%, and 90.4% for the [1 1], [1 3], and [1 5] robot teams, respectively (Figure 8.16 (a),(c)). Area explored is improved by 57.8% and 4.1% for the [1 1] and [1 3] robot teams, respectively (Figure 8.16 (b),(c)).

Figure 8.17 compares the poor performance feedback and non-feedback results when the monitor time is set to 300 sec. Setting the monitor time to 300 sec (Figure 8.17)

also achieves similar improvements to exploration with a monitor time of 60 sec (Figure 8.15). The [1 1], [1 3], and [1 5] robot teams have their exploration times reduced to 68.6%, 87.3%, and 90.8%, respectively (Figure 8.17 (a),(c)). Improvements of 57.5% and 4.3% in area explored have been obtained for the [1 1] and [1 3] robot teams, respectively (Figure 8.17 (b),(c)).

In a team of physical robots, poor performance can be successfully detected and corrected thereby improving task execution. The poor performance detection threshold value will usually be a non-zero number. A threshold value closer to unity enables faster detection of poor performance but risks false detection. Poor performance failure detection and correction is robust to threshold variation and monitor time interval variation.



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)		
Robot Config.	Exploration Time (a)	Area Explored (b)
	P1	P1
[1 1]	< 0.01	< 0.01
[1 3]	< 0.01	< 0.01
[1 5]	< 0.01	0.23

(c)

Figure 8.17: Comparison of poor performance feedback and non-feedback results at T_m = 300 sec.

8.5.2 Partial Failure Experiments

Partial failure tests have been carried out on the [1 3] and [1 5] robot teams only. As detailed in section 5.6.2, a partial failure is corrected by reallocating the faulty robot's

task to a new robot. It is obvious that detecting and correcting a partial failure in the [1 1] team will have better performance than without employing any feedback. Figure 8.18 shows the results obtained when partial failure feedback is employed for the [1 3], and [1 5] robot team configurations using a monitor time T_m = 60 sec. Table 8.1 lists the numerical values of thresholds F1–F4.

In the [1 3] and [1 5] robot teams, there is little variation in exploration time for all tested threshold values (Figure 8.18 (a)). Thresholds F1–F4 also show little variation in area explored for the [1 3] and [1 5] robot teams (Figure 8.18 (b)). The small variation in exploration rates for both team configurations (Figure 8.18 (c)) indicates that partial failure feedback is not sensitive to threshold variation. This means that the rate at which partial failures are detected and corrected is similar for the tested threshold values.

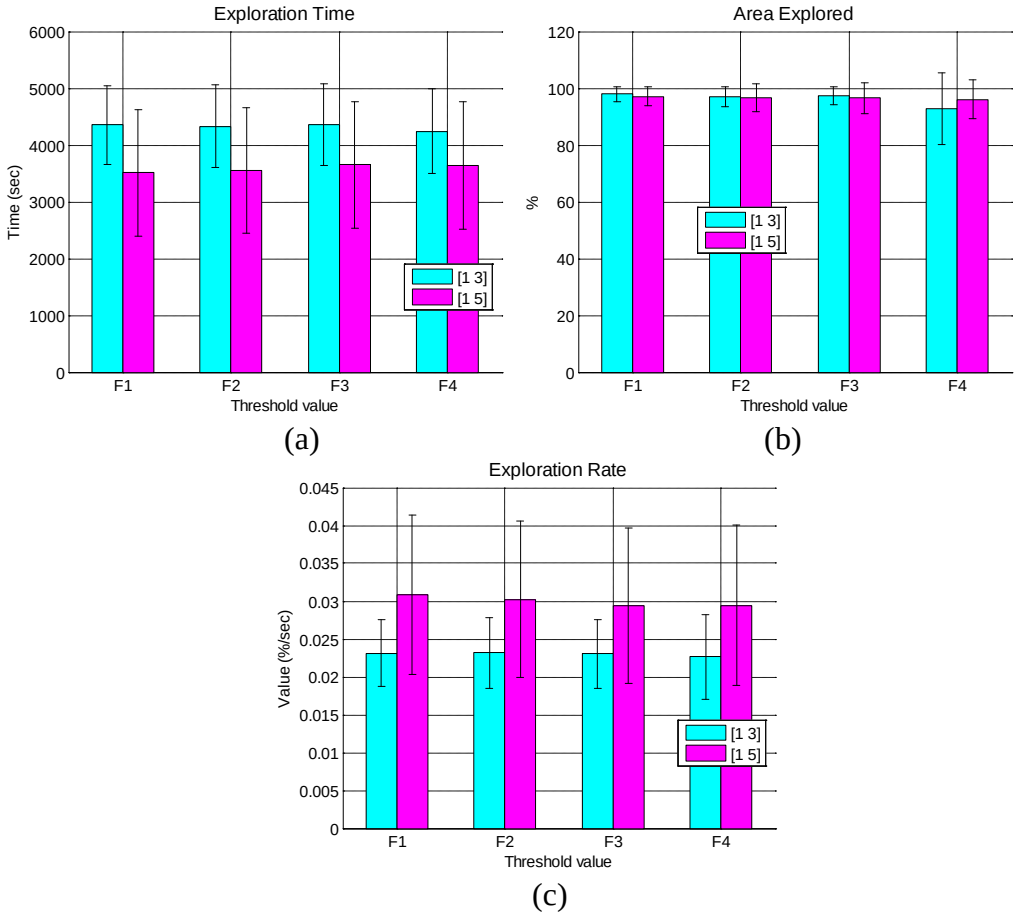
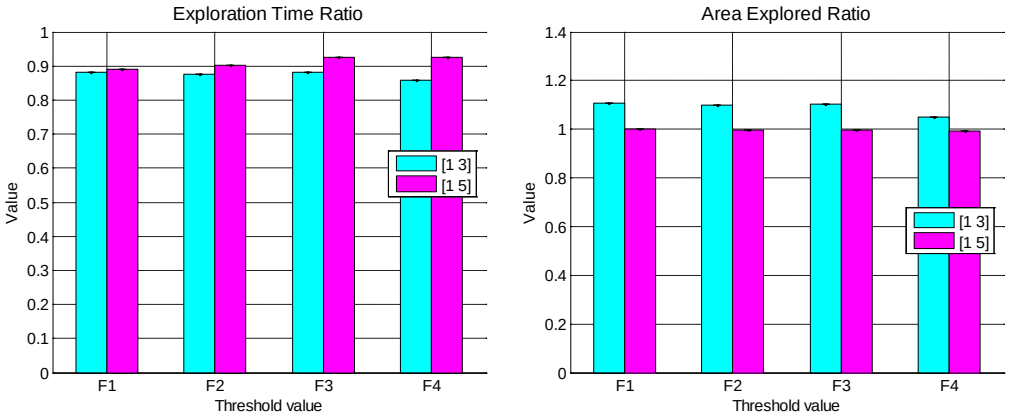


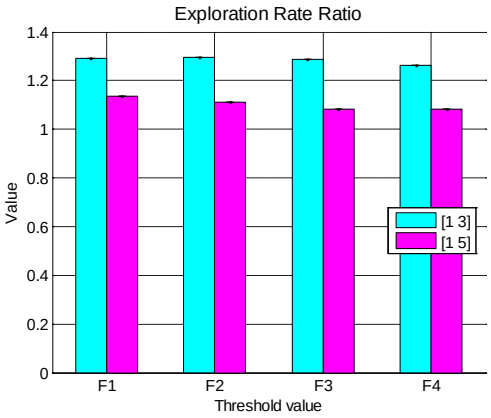
Figure 8.18: Results of exploration with partial failure feedback.

Figure 8.19 illustrates the statistical significance comparison made between the partial failure feedback and non-feedback exploration results. A maximum reduction in exploration time is achieved at threshold value F4 for the [1 3] robot team (Figure 8.19 (a)). Exploration time is reduced to 85.8% at this threshold value. Threshold value F1 produces the greatest reduction in exploration time (89.1%) for the [1 5] robot team (Figure 8.19 (a)). The reductions in exploration time are statistically significant with p-values less than 0.05 (Figure 8.19 (d)).



(a)

(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)					
	Robot Config.	Threshold Value			
		F1	F2	F3	F4
Exploration Time (a)	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	< 0.01	0.01	0.01
Area Explored (b)	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	0.30	0.26	0.21	0.09
Exploration Rate (c)	[1 3]	< 0.01	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	< 0.01	0.03	0.03

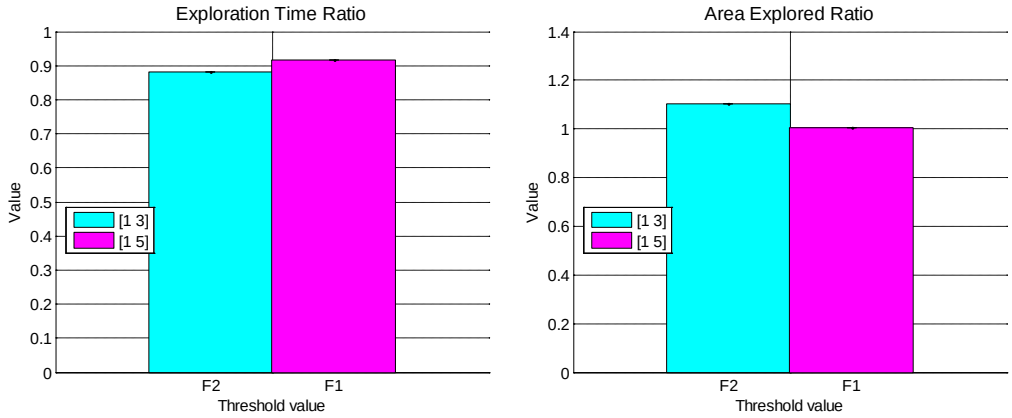
(d)

Figure 8.19: Comparison of partial failure feedback and non-feedback results at T_m = 60 sec.

Area explored improvement is greatest at threshold value F1 (Figure 8.19 (b)). A statistically significant improvement of 10.8% is achieved for the [1 3] robot team at F1 (Figure 8.19 (b),(d)). The area explored by the [1 5] robot team is similar to the non-feedback experiments (Figure 8.19 (b),(d)) because five explorers are able to complete exploring the global world within the *MET* (5000 sec) in both cases.

Threshold value F2 has been selected the best for the [1 3] team since it has the highest exploration rate improvement of 29.4% when compared to the non-feedback experiments (Figure 8.19 (c)). This comparison is statistically significant (Figure 8.19 (d)). It reduces exploration time to 88.1% and area explored is improved by 10.8%. It should be noted that F1 and F3 also produce similar exploration rate improvements to F2 indicating robustness.

For the [1 5] robot team, threshold F1 is the best since it has the highest statistically significant exploration rate improvement (13.5%) when compared to the non-feedback experiments (Figure 8.19 (c),(d)). Exploration time is reduced to 89.1% when this threshold value is employed.



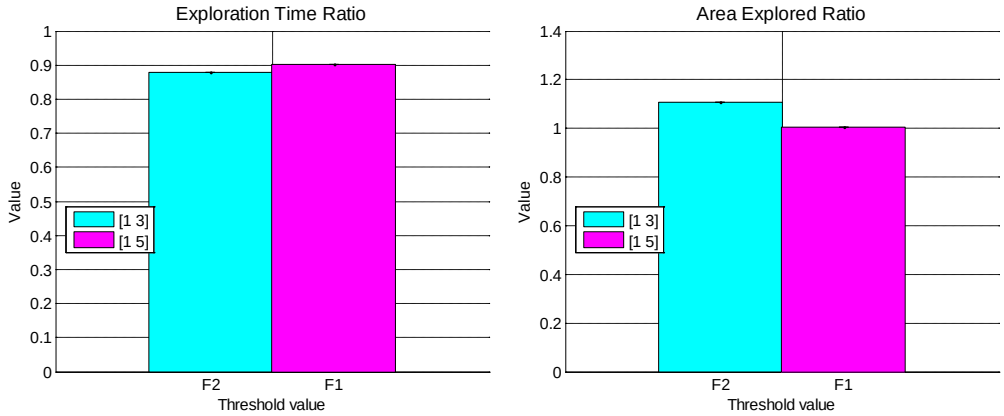
(a) (b)

p-values (two decimal places, < 0.05 shaded blue)				
Robot Config.	Exploration Time (a)		Area Explored (b)	
	F1	F2	F1	F2
[1 3]	–	< 0.01	–	< 0.01
[1 5]	< 0.01	–	0.07	–

(c)

Figure 8.20: Comparison of partial failure feedback and non-feedback results at T_m = 180 sec.

The performance of the best weight sets have also been evaluated for monitor time values of 180 sec and 300 sec. Figure 8.20 shows a comparison of exploration with partial failure feedback at T_m = 180 sec and exploration without any feedback. The results for the [1 3] and [1 5] robot teams are similar to thresholds F2 and F1 in Figure 8.19 (T_m = 60 sec), respectively. Exploration time is reduced to 88.1% for the [1 3] robot team (Figure 8.20 (a)). In the [1 5] robot team, exploration time is reduced to 91.8% (Figure 8.20 (a)). These exploration time reductions are statistically significant (Figure 8.20 (c)). A statistically significant increase in area explored (10.4%) is achieved for the [1 3] robot team (Figure 8.20 (b),(c)).



(a) (b)

p-values (two decimal places, < 0.05 shaded blue)				
Robot Config.	Exploration Time (a)		Area Explored (b)	
	F1	F2	F1	F2
[1 3]	–	< 0.01	–	< 0.01
[1 5]	< 0.01	–	0.10	–

(c)

Figure 8.21: Comparison of partial failure feedback and non-feedback results at T_m = 300 sec.

Figure 8.21 compares exploration with partial failure feedback at T_m = 300 sec and exploration without any feedback. Partial failure feedback using T_m = 300 sec (Figure 8.21) also offers similar improvements to using T_m = 60 sec (Figure 8.19). In the [1 3] robot team, exploration time is reduced to 88% (Figure 8.21 (a)). Exploration time is reduced to 90.1% in the [1 5] robot team. Both exploration time reductions are statistically significant (Figure 8.21 (c)). The [1 3] robot team also achieves a 10.7% increase in area explored that is statistically significant (Figure 8.21 (b),(c)).

For a team of physical robots, partial failures can be successfully detected and corrected thus improving task execution. The partial failure detection threshold value will usually be a small number (close to zero) to reduce false detections. For the tested values, partial failure detection and correction is robust to threshold variation and monitor time interval variation.

8.5.3 Complete Failure Experiments

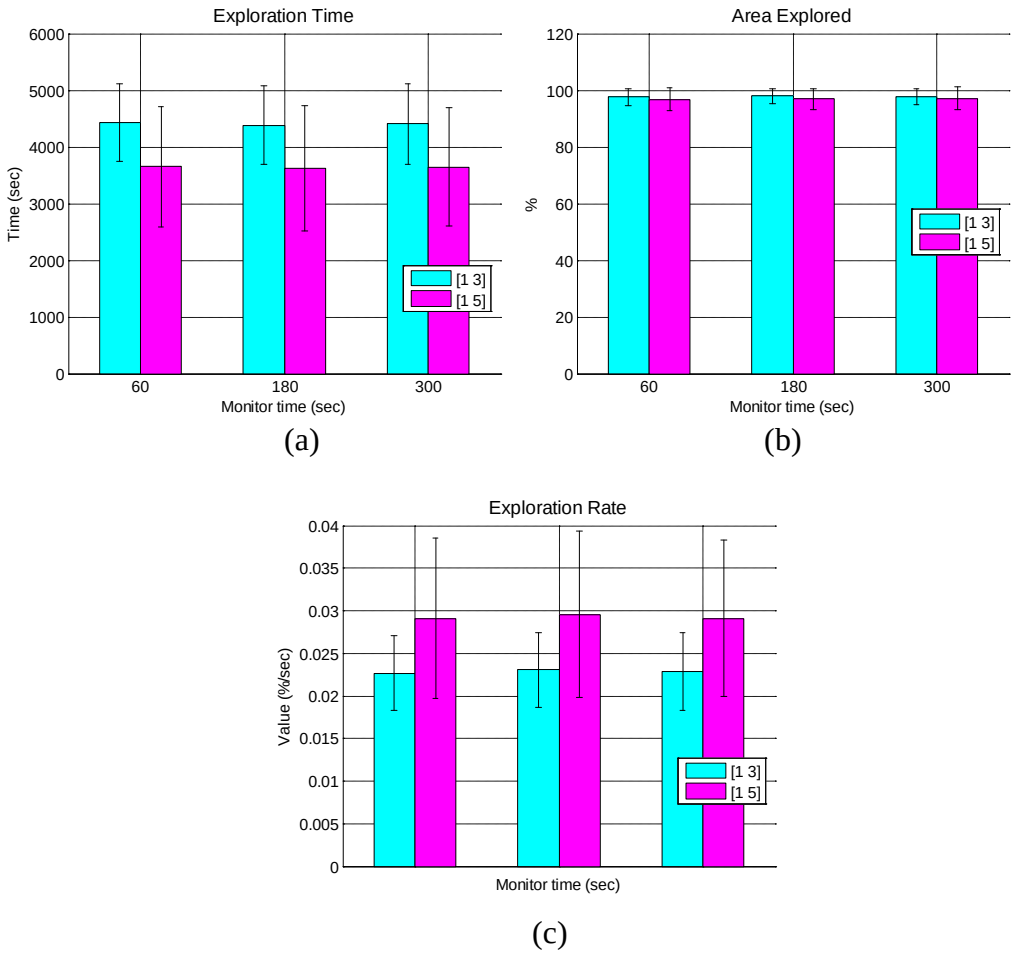
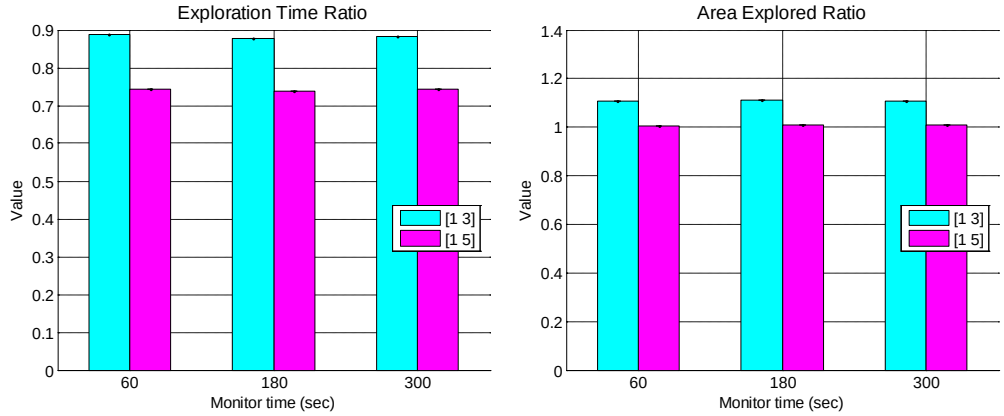


Figure 8.22: Results of exploration with complete failure feedback.

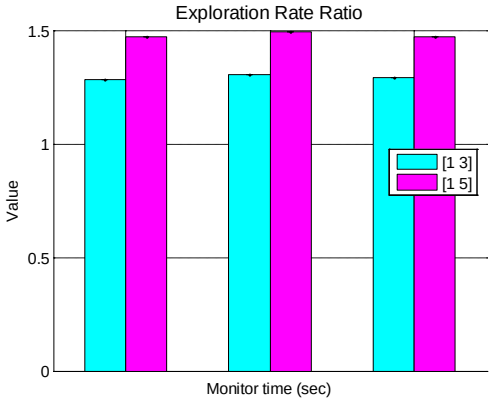
Similar to the partial failure experiments, complete failure experiments have also been carried out on the [1 3] and [1 5] robot teams only. It is clear that exploration will fail in the [1 1] team when complete failures happen. Figure 8.22 illustrates the exploration results obtained when complete failure feedback is employed. Exploration time, area explored, and overall scores are similar for all three monitor time values

(Figure 8.22 (a)–(c)). Thus complete failure feedback can perform at multiple monitor time values. At least 97% of the global environment is explored for all tested robot team – monitor time combinations when complete failure feedback is utilised (Figure 8.22(b)).



(a)

(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)				
	Robot Config.	Monitor Time (sec)		
		60	180	300
Exploration Time (a)	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	< 0.01	< 0.01
Area Explored (b)	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	0.08	0.04	0.02
Exploration Rate (c)	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	< 0.01	< 0.01

(d)

Figure 8.23: Comparison of complete failure feedback and non-feedback results.

Complete failure feedback results and non-feedback results are compared with statistical significance tests in Figure 8.23. Exploration time is reduced for all robot

team – monitor time combinations (Figure 8.23 (a)). These reductions are all statistically significant with p-values less than 0.01 (Figure 8.23 (d)). In the [1 3] team, exploration time is reduced to 89.9%, 87.8%, and 88.3% for monitor time values of 60 sec, 180 sec, and 300 sec, respectively. Monitor times 60 sec, 180 sec, and 300 sec reduce exploration time to 74.5%, 73.9%, and 74.3%, respectively, for the [1 5] robot team. These performance improvements over the various monitor times are similar within measurement errors.

Area explored is increased for all robot team – monitor time combinations (Figure 8.23 (b)). These increments are all statistically significant except for the [1 5] robot team at 60 sec monitor time (Figure 8.23 (d)). An 11% increase in area explored is observed for the [1 3] robot team. In the [1 5] robot team, a 1% increase in area explored is achieved for monitor time values of 180 sec and 300 sec.

The reduction in exploration time and increase in area explored for all tested combinations produces a statistically significant increase in the exploration rate (Figure 8.23 (c),(d)). In Figure 8.23 (d), the exploration rate p-values are all less than 0.01. For the [1 3] team, exploration rate is improved by 28.1%, 30.3%, and 29.3% for monitor time values of 60 sec, 180 sec, and 300 sec, respectively. Monitor times 60 sec, 180 sec, and 300 sec improve exploration rate by 47.2%, 49.5%, and 47.2%, respectively, for the [1 5] robot team. Again, these improvements are similar within measurement errors.

Hence, complete failure detection is robust when monitor time is varied from 60 sec to 300 sec.

8.5.4 Combined Feedback Experiments

This section evaluates the combined feedback system with all three types of failures (poor performance, partial failure, and complete failure) and task score feedback (section 8.4) enabled using their best threshold values and weights. Experiments and results presented in this section are designed to determine if the various components of the feedback system can function when combined in a complete system. The small variation in performance over different threshold values and weights for the

individual failure tests and task score feedback experiments means that selecting different values from the best ones will not affect combined system performance significantly.

Figure 8.24 illustrates the results obtained when combined feedback is employed for exploration. Exploration time (Figure 8.24 (a)), area explored (Figure 8.24 (b)), and exploration rate (Figure 8.24 (c)) are similar for all three monitor time values. Hence, the combined feedback system is capable of performing at multiple monitor time values. At least 94% of the global environment is explored for all tested robot team – monitor time combinations (Figure 8.24 (b)).

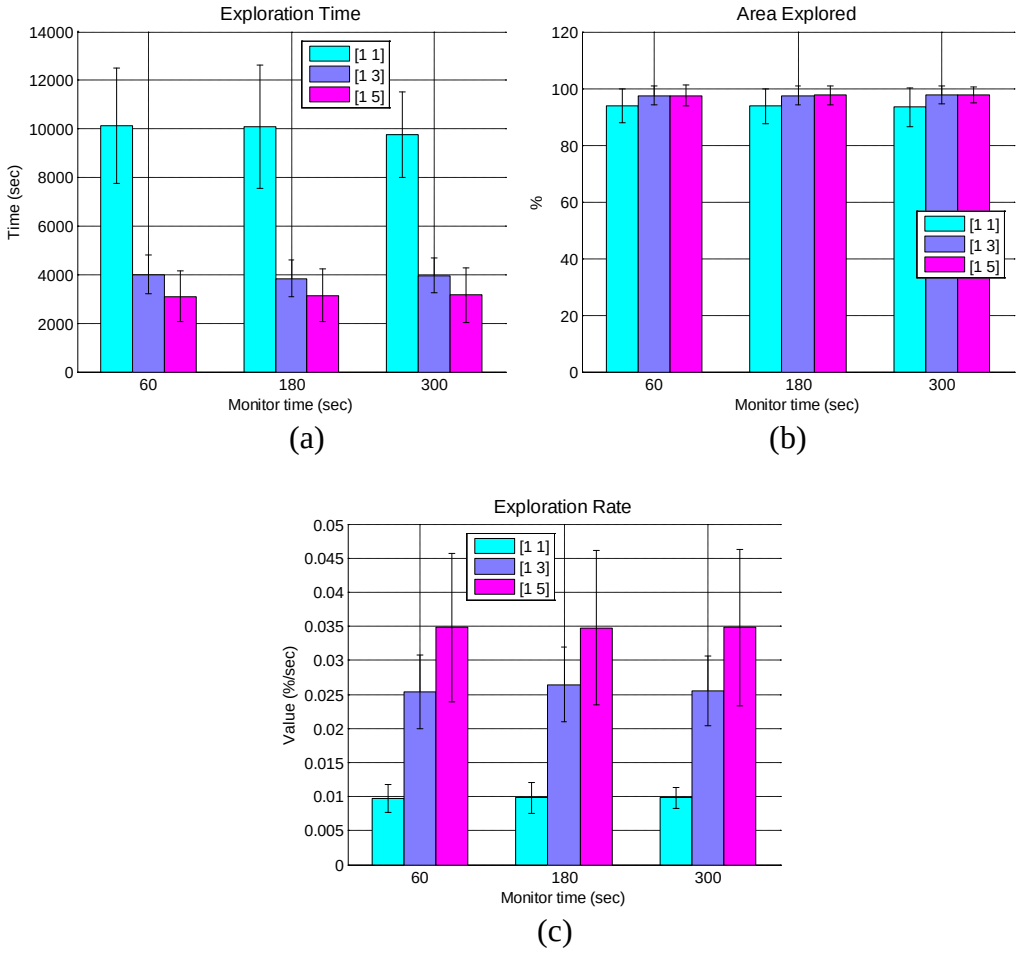
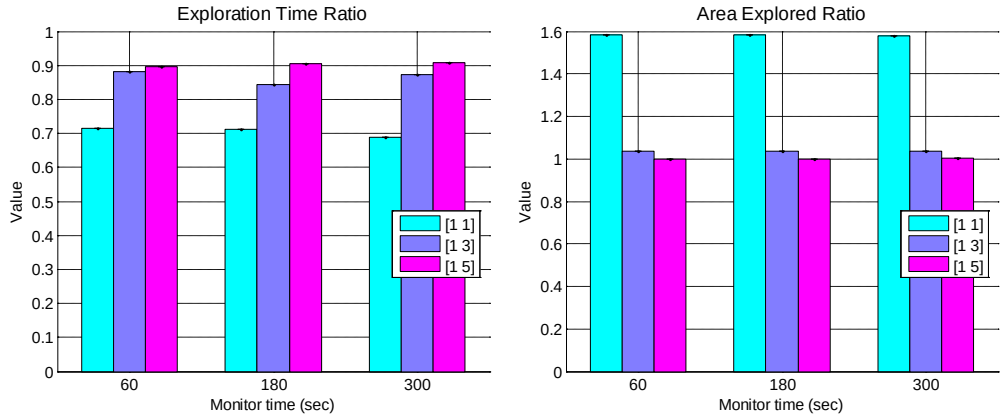


Figure 8.24: Results of exploration with combined feedback.

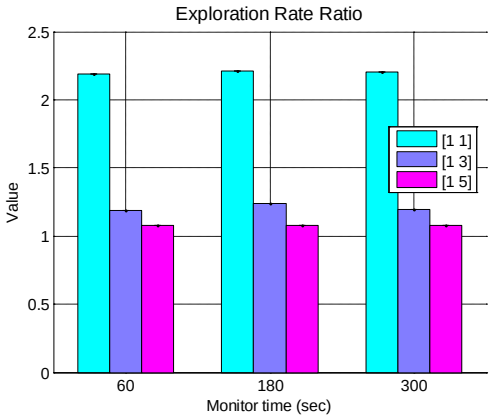
A statistical significance comparison between the combined feedback and non-feedback exploration results has been made in Figure 8.25. Exploration time (Figure 8.25 (a)), area explored (Figure 8.25 (b)), and exploration rate (Figure 8.24 (c)),

produce similar results for each robot team as the monitor time is varied from 60 sec to 300 sec.



(a)

(b)



(c)

p-values (two decimal places, < 0.05 shaded blue)				
	Robot Config.	Monitor Time (sec)		
		60	180	360
Exploration Time (a)	[1 1]	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	< 0.01	< 0.01	< 0.01
Area Explored (b)	[1 1]	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	0.39	0.42	0.23
Exploration Rate (c)	[1 1]	< 0.01	< 0.01	< 0.01
	[1 3]	< 0.01	< 0.01	< 0.01
	[1 5]	0.03	0.04	0.03

(d)

Figure 8.25: Comparison of combined feedback and non-feedback results.

Robot team [1 1] achieves exploration rate improvements of 119%, 120%, and 120% for T_m = 60 sec, T_m = 180 sec, and T_m = 300 sec (Figure 8.25 (c)). The improvement is due to the exploration time being reduced to a statistically significant 70% of the non-

feedback exploration time (Figure 8.25 (a),(d)). A statistically significant increase of 58% in area explored is achieved for the [1 1] robot team (Figure 8.25 (b),(d)).

At T_m = 180 sec, the greatest exploration rate improvement of 22.8% is achieved for the [1 3] robot team (Figure 8.25 (c)). Exploration rate improvements of 19.7% and 20.7% are achieved for T_m = 60 sec and T_m = 300 sec, respectively. These values are similar within measurement error. The area explored improvement is 3.6% for all monitor time values (Figure 8.25 (b)). Exploration time is reduced to 87%, 84.8%, and 86.4% for T_m = 60 sec, T_m = 180 sec, and T_m = 300 sec, respectively. Similar to the exploration rate comparison, these values are similar within measurement error.

For [1 5] team, there is not much variation in exploration time or area explored when monitor time is changed (Figure 8.25 (a),(b)). This suggests that there may be many explorers relative to global environment size to see the effects of monitor time change. However, exploration time is reduced to a statistically significant 89.5%, 90.4%, and 90.8% for T_m = 60 sec, T_m = 180 sec, T_m = 300 sec, respectively (Figure 8.25 (a),(d)). This produces an exploration rate improvement of approximately 7.7% for the [1 5] robot team when complete feedback is employed.

These results verify that the various components of the feedback system exhibit synergy when integrated. The combined system is also robust when monitor time is varied from 60 sec to 300 sec.

8.6 Alternative Techniques

8.6.1 Mapping and Exploration

Methods that develop efficient exploration strategies to disperse robots and take advantage of heterogeneous robots have been reviewed in section 2.9.

Singh and Fujimura [1] take advantage of heterogeneity by selecting appropriately sized robots to investigate “tunnels” leading to unexplored regions. The BERODE architecture [2] uses behavioural roles (maintainer, recoverer, and pusher) to address limited communication in multi-robot exploration. In this thesis, heterogeneity in the processing capabilities of robots is exploited for hierarchical multi-robot exploration.

A computationally powerful robot (manager) coordinates the lower-tiered limited capability robots (planners and explorers). Different levels of abstraction are maintained. Manager robots have a global view of the task. On the other hand, explorer robots are only aware of the section of the environment that they are required to explore. Planners are made aware of what plans they need to make and for which robots.

Simmons [3] also divides the exploration task but not based on heterogeneity due to computational ability. A central mapper unit and central executive (similar to a manager robot) coordinate exploration. However, the approach does not use separate planner robots as this is handled by the central executive. This reduces its ability to handle decentralised groups of explorer robots. On the other hand, it is less dependent on inter-robot communication. Similar to Simmons [3], Tovar et al. [4] also employ a centralised planner to coordinate robot movement for exploration.

The robots employed for exploration in this thesis utilise infrared rangefinders. Other exploration strategies are based on superior sensors such as ultrasonics [1, 2, 5-7], laser scanners [3, 4, 8, 9], and vision [4]. This limitation in sensing necessitates the division of the global environment into smaller territories for individual robots unlike the approaches reported in section 2.9. It also makes comparisons of exploration time in similar sized environments inappropriate. Naturally, a team of robots with superior sensors is able to explore an environment more efficiently.

Dispersing robots within an environment for exploration generally balances an expected revenue (or utility) and the cost of travel (a function of distance). The approach taken in this thesis is similar to some of the methods reviewed in section 2.9 ([3-8]).

Yamauchi [5] disperses robots based on proximity to frontier regions (cost of travel) only. The robots maintain separate global maps and make independent decisions about where to explore. Hence, there is no explicit coordination which can result in time wastage during exploration.

Simmons [3] computes a utility value for representative frontier cells based on the difference between an estimated information gain value (revenue) and a cost value. The cost value is computed from the optimal path between the robot's current location and the frontier cell. Estimated information gain is an estimate of the unexplored space around the frontier cell. A discounting factor is applied to the estimated information gain value to keep robots apart during exploration. Burgard et al. [8] use an approach similar to Simmons [3]. In this approach, estimated information gain is replaced by a utility value and the difference between utility and cost is computed. A utility reduction function reduces the utility of target points within the sensing range of a robot to aid dispersion. Poernomo and Ying [6] use a cost function similar to [3, 5, 8] but reduce the cost value based on the distances between robots to aid dispersion.

Zlot et al. [7] developed a market-based exploration strategy that uses distance information to compute costs and information gained by visiting a goal point to compute revenue. A profit (similar to utility) is computed from the difference between revenue and cost. Robots bid on tasks (goal points to visit) in auctions and the highest bidders (highest profiting robots) above a reserve price are awarded tasks. This approach is more complex than [3, 8] as it requires revenue functions, cost functions, and reserve prices to be appropriately defined to achieve coordination. Tovar et al. [4] use a complex product utility function that employs fourteen variables to balance cost of travel and expected information gain (revenue).

Similar to [3, 6, 8], the method presented in this thesis disperses robots by trading off utility and cost values. Utility values are unit interval values computed based on the size of the local environment. Cost values are determined from a weighted combination of the distance to a local environment and the distance to other robots. Unlike [3-8], time-based (or performance-based) cost information is included in the form of a novel of task score feedback value. Task score feedback allows weaker explorer robots to reserve local environments near them.

A drawback of the method presented in this thesis is the dependence on accurate localisation of the limited robots. This need arises since the robots are required to traverse the environment and acquire sensor data simultaneously. Due to the limited processing capabilities of the robots, navigation aids such as beacons or GPS are

preferred over SLAM. The exploration method also depends on initially specified environment boundaries.

Unlike other map-building and exploration techniques (section 2.9), a feedback coordination system is used to improve the area explored and reduce the overall exploration time. It makes the team dynamic during exploration (to account for failures) unlike the other approaches that employ fixed robot teams for exploration.

8.6.2 Fault Tolerance (Feedback Coordination)

Fault tolerance in multi-robot systems has been reviewed in section 2.8. The L-ALLIANCE architecture [10] is designed for behaviour-based systems. Task completion time is used as the performance metric and motivational behaviours effect task reallocation. Hence, the system can detect failures but not classify them. The performance of a robot in executing a specific task over a few recent trials is used as an estimate of expected performance.

Kannan and Parker's [11] task execution success and failure metrics influence fault tolerance and overall system performance. Overall performance is computed as the difference between successfully executed tasks (success metric) and unsuccessful task execution (failure metric). Hence, performance can only be determined after the completion of tasks. Kannan and Parker [12] further develop adaptive causal model based performance metrics for fault-tolerant systems. Causal models are complex and cumbersome to implement. They also need to be tailored for different tasks and environments. In the SFX-EH architecture [13], robots share knowledge of the state of their sensors and task execution to detect sensing failures. Failures are dealt with by taking corrective actions such as reconfiguring perceptual schemas or recalibrating sensors.

The feedback coordination system developed in thesis is designed to be used with the novel reduced human user input task allocation system. Unlike the methods of section 2.8, the feedback system monitors the four broad categories of robot hardware resources (processing, communication, sensing, and actuation) explicitly. Robots are monitored during task execution and can be replaced before task completion if

performance is unsatisfactory. Failure detection accounts for suboptimal greedy allocations, inaccurate task specifications, and hardware failures (such as sensing, actuation, or processing). Unlike [13], the feedback system replaces robots instead of attempting to correct the failed sensors of a robot. A simple linear weighted combination of the four resource categories is used to determine the performance a robot. However, similar to [10], a reasonable estimate of the ideal (best-case scenario) performance of the robots is still required before task execution. Unlike [11], a ratio of the current performance to ideal performance produces a task execution success metric that can be determined during task execution.

8.7 Summary

This chapter has presented results on a novel feedback coordination mechanism that can be employed by a hierarchical heterogeneous multi-robot system comprising limited capability mobile robots. A customised multi-level abstracted multi-robot mapping and exploration task (chapter 6) has been employed as a model task for experiments. Exploration results obtained with the various types of feedback have been compared to the results obtained without any feedback.

Task score feedback does not affect exploration time. However, it can positively impact area explored in a global world that comprises boggy terrain. In the [1 3] robot team, task score feedback improved the area explored by 5.2% for global worlds with boggy terrain. Task score feedback is capable of performing at multiple monitor time values (60 sec, 180 sec, and 300 sec).

The feedback system is able to successfully identify and correct three types of failures. Poor performance feedback offers improvements for all three robot teams. In the [1 1] robot team, poor performance feedback can reduce exploration time to as low as 69% and increase area explored by up to 59%. For the [1 3] robot team, it can reduce exploration time to as low as 84.4% and increase area explored by up to 4.1%. Exploration time can be reduced to as low as 89.5% in the [1 5] robot team. The impact of exploration time reduction is less as the number of explorers is increased

due to redundancy. Poor performance feedback is able to function successfully at monitor time values of 60 sec, 180 sec, and 300 sec.

Partial failure feedback has been tested on the [1 3] and [1 5] robot team configurations. In the [1 3] robot team, area explored can be increased by up to 10.8% and exploration time can be reduced to as low as 85.8%. For the [1 5] robot team, exploration time can be reduced to as low as 89.1% when partial failure feedback is employed. Similar to poor performance feedback, partial failure feedback operates successfully at monitor time values of 60 sec, 180 sec, and 300 sec.

Complete failure feedback has also been tested on the [1 3] and [1 5] robot teams. Area explored can be increased by up to 11% and exploration time can be reduced to as low as 87.8% for the [1 3] robot team. In the [1 5] robot team, exploration time can be reduced to 73.9% and a 1% increase in area explored can be achieved. Complete failure feedback is also able to perform at various monitor time values.

When combined into a complete system, task score feedback, poor performance feedback, partial failure feedback, and complete failure feedback are able to function successfully for all robot team – monitor time interval combinations.

Task score feedback is robust to achievement bias weight variation within the tested limits. Similarly, the detection and correction of the three forms of failures is robust to threshold value variation within the tested limits. Hence, achievement bias weights and threshold values can be intuitively selected without having a negative impact on performance.

All individual feedback system components are also robust to monitor time interval variation within the tested limits. Robustness to monitor time interval variation is also achieved in the combined system comprising all feedback system components. Compared to a monitor time interval of 60 sec, a monitor time interval of 300 sec reduces the volume of messages transmitted and received in the heterogeneous multi-robot system without negatively affecting performance.

A single set of ideal achievement data is employed for all the experiments presented in this chapter. Hence, an expert user is expected to “calibrate” a robot’s performance only when the task type is altered.

A similar procedure can be utilised to evaluate the feedback system for different applications.

9 Conclusions

9.1 Overview

This thesis has presented an artificial intelligence system that exploits the benefits of hierarchical heterogeneous multi-robot systems and potentially allows non-expert human users to utilise it. In a system that consists of computationally powerful robots at the upper level and limited capability robots at the lower levels, resources (such as processing) can be shared and tasks can be abstracted.

A hierarchical hybrid navigation system that does not rely on periodic path planning to enable limited capability heterogeneous robots to explore and traverse environments is summarised in section 9.2. Section 9.3 summarises a novel two-tiered global path planner that permits limited memory robots to utilise the memory of computationally powerful robots. The navigation system and global path planner are components of a customised multi-level abstracted multi-robot map building and exploration task that is used to demonstrate a novel reduced human user input task allocation and feedback coordination technique (section 9.4). Limited capability mobile robots are able to efficiently execute a group task using the developed task allocation and feedback coordination strategy.

Section 9.5 lists future extensions to the research presented in this thesis. Publications that have arisen from this thesis and their corresponding contributions are highlighted in section 9.6. A summary of achievements is provided in section 9.7.

9.2 Basic Robot Navigation System

A hierarchical hybrid navigation system has been developed for basic individual robot navigation that permits both reactive and deliberative control. This offers flexibility for navigation in known and unknown environments. The underlying navigation system is based upon a rudimentary implementation of Lee-Johnson’s navigation system [51]. Several bugs and shortfalls in Lee-Johnson’s initial navigation system

have been identified and corrected. The rudimentary system has been successfully further extended to facilitate the control of heterogeneous robots.

Deliberative control is developed using a modified version of the A* path planning algorithm and a rectangular occupancy grid map (section 3.2.2). The modified A* algorithm permits continuous cost values instead of binary data. Inspired by frontier-based exploration, the rectangular occupancy grid map is updated in real-time and path replanning takes place when required.

The reactive control strategy combines a modified dynamic window approach (section 3.5) and a direction sensor similar to the vector field histogram technique (section 3.4). Employing only a dynamic window approach for reactive control yields poor navigation. Combining the developed modified dynamic window with a direction sensor reduces navigation time to 20%–50% when compared to solely relying on the dynamic window. This combines the benefits of directional methods and velocity space techniques to produce a reactive system that is not dependent on periodic path planning.

In unknown environments, reactive navigation is able to achieve similar performance to combined reactive – deliberative navigation. This removes the dependence on periodic path planning for hybrid navigation in unexplored obstructed environments. While the navigation system employs many empirically tuned parameters to achieve reduced dependence on periodic planning, a fixed set of parameters has been functional for the tested robot-environment combinations in simulations. The hybrid navigation system has also been successfully implemented on a physical tricycle robot and tested in an indoor corridor environment.

9.3 Memory Constrained Path Planning

A novel method for global path planning that enables limited memory robots to utilise the memory of computationally powerful robots has been developed and evaluated. The technique is suitable for use in hierarchical heterogeneous multi-robot systems to allow memory constrained robots to navigate beyond localised regions of a global

environment. This is useful for tasks such as exploration where robots may be required to map large environments.

Specifically, the developed global path planning strategy is well suited for use in a hierarchical heterogeneous multi-robot system such as Victoria University of Wellington's urban search and rescue system (chapter 1). As outlined in section 1.1, manager robots supervise worker robots during task execution. However, due to limited processing ability, manager robots may not be able to perform global path planning and maintain full communication with all worker robots. Instead of relying on a single centralised path planning robot, global path planning is decentralised and assigned to some of the worker robots.

Smaller sized local maps are created from a large global map that cannot be completely stored in a memory constrained robot. The local map size is dependent on the memory constrained robot's memory capacity. After dividing the large environment into small sections, a two-tiered A* algorithm was developed that sequentially searches the local maps for a global path. The path planning algorithm executes entirely on the memory constrained robot. During path planning, the memory constrained robot retrieves local map data from other higher memory capacity robots.

Superior or comparable execution times to non-memory constrained path planning are achieved by the memory constrained technique when the local map size is much smaller than the global map size. A limited memory robot is able to further improve memory constrained planning by dividing its available memory space to store multiple smaller sized local maps.

Memory constrained path planning search space is reduced at high ($\geq$ 20%) obstacle densities. This can affect obstacle clearance. However, path length is not adversely affected at these higher obstacle densities. In general, many real environments have low overall obstacle densities and smaller regions of high density obstacles may be confined to a few local maps. This reduces obstacle clearance issues in memory constrained planning. Employing a hybrid navigation system with a good reactive controller also mitigates this issue. Moreover, if obstacles block some local map

boundaries, the back tracking algorithm can find a path through other local maps provided that exit points exist in them.

9.4 Task Allocation and Feedback Coordination Mechanism

A novel reduced human user input task allocation and feedback coordination mechanism for limited capability mobile robots has been developed and evaluated. As previously mentioned, an urban search and rescue system under development at Victoria University of Wellington is a potential application of the developed task allocation and feedback technique. Three levels of control exist in the mechanism. At the highest level of control, a remote base station computer specifies a group task and the robots available for selection. The second and third levels of control comprise manager robots and worker robots, respectively. Task allocation (devolution) processes identify the managers and workers from the list of available robots. Once the managers are identified, the remote base station is no longer needed since executive control is transferred to them. Worker robots execute tasks based on instructions from the manager robots.

To reduce human user input, tasks and robots are specified using four major categories of robot capabilities (i.e. resources): processing, communication, sensing and actuation. The task allocation algorithms employ numerical vector of merit (VOM) data that specifies robot capabilities. The four major resource categories are also encoded in vector of task requirements (VOTR) data to specify tasks that require allocation. By representing robots and tasks using these four major resources, the task allocation process is also made generic. A vector of task suitability (VOTS) is computed from VOTR and VOM data to identify eligible robots for task execution. A primary task devolution process identifies the manager robots while a secondary task allocation process is employed to select worker robots.

Often, a human user may not be able to accurately specify the type of mobile robot required for a task. For example, it is often unreasonable to specify the exact quantity and type of sensors required for an exploration task. In such situations it is often better to input a grading for a sensor type (or the sensing resource/capability) and let the task

allocation process choose the best robot for the task. Hence, the reduced human user input task allocation system also permits tasks to be specified with such graded information. This is unlike many other task allocation strategies that rely on expert knowledge for task specification. Fuzzy Inference Systems (FISs) have been employed to simplify the VOM data for comparison with the VOTR data. However, it should be noted that additional effort is required from the system designer to reduce complexity for non-expert users.

After initial task allocation, the worker robots may fail to perform adequately. This can be due suboptimal initial task allocations or unexpected failures (such as inaccurate task specifications or hardware failure). In the developed system, a feedback mechanism monitors the efficiency of worker robots during task execution. If the performance of a worker robot is unsatisfactory, a task reallocation algorithm adjusts the task-robot combinations of the team.

Three forms of unsatisfactory performance can be detected by the feedback system. Poor performance is detected when the overall success of a robot during its entire period of operation falls below a non-zero threshold. Partial failure detection considers the recent success of a robot. If a robot's recent success falls below a threshold value that is usually close to zero, it has partially failed. A completely failed robot is detected when performance data are not received from the robot and it fails to send pulse signal activity messages.

A customised multi-level abstracted multi-robot map-building and exploration task has been implemented as a model group task to demonstrate the effectiveness of the developed task allocation and feedback coordination system. It is important to note that the developed system is generic and not limited purely to exploration.

The map-building and exploration technique takes advantage of the benefits of hierarchical heterogeneous multi-robot systems. It is well suited for a three-tiered multi-robot system comprising worker robots with limited sensing and processing capabilities. Worker robots can be assigned planner and explorer tasks. A computationally powerful manager robot coordinates the planner and explorer tasks to enable mapping of a large environment containing scattered obstacles. Thus, the manager robot maintains a global view of the task. Planner tasks enable robots to

navigate to new localised regions of the large environment. Robots that are assigned planner tasks only need to be made aware of what plans they need to make and for which robots. The explorer task permits navigation and exploration within a localised region of the global environment. Hence, robots that are assigned explorer tasks are only made aware of the section of the environment that they are required to explore.

Task devolution experiments show that primary task devolution is able to successfully identify and select suitable robots for manager tasks. The selection of manager robots is robust for the tested VOTS summation weight sets. Secondary task devolution is able to successfully identify and select worker robots. Applying small weightings to sensing and actuation in the planner worker task gives unique rankings to robots without affecting robustness of task allocation. Similarly, giving small weightings to planning and communication in the explorer task also enables unique ranks for robots without adversely affecting robustness.

Experiments on the feedback system show that the three forms of failures (poor performance, partial failure and complete failure) are successfully detected and corrected. This results in improved task execution performance. In smaller teams, task score feedback it able to provide a small improvement in the area explored for environments comprising regions of boggy terrain without affecting the time taken to complete exploration. Task score feedback, poor performance feedback and partial failure feedback are robust to weight and threshold variation within the tested limits. Hence, it is possible to select weights and thresholds intuitively without negatively affecting performance. The feedback system is also robust to monitor time interval variation within the tested limits, allowing dependence on communication to be varied. Additionally, a single set of ideal achievement data is employed in the experiments. Thus, it is expected that an expert user will need to “calibrate” a robots performance only when the task type is changed.

9.5 Future Work

The vast majority of the experiments presented in this thesis are simulations because multiple robots are tested in a variety of environments. Navigation system experiments on physical robots have been limited to a single tricycle mobile robot.

More experiments on a variety of physical heterogeneous robots in a range of real environments would be useful to evaluate the navigation system's performance in real world situations. Extensive simulation experiments have evaluated the novel two-tiered global path planning technique for limited memory mobile robots. The performance of the two-tiered path planner in the real world can be investigated with physical robots in tasks such as multi-robot exploration. Similarly, the task allocation and feedback coordination technique can be applied to a multi-robot mapping and exploration task utilising physical robots to evaluate its effectiveness in the real world.

To investigate the degree to which the reduced human user input task allocation and feedback coordination technique is generic, it would be useful to implement and evaluate its performance in alternative multi-robot applications such as cooperative object transportation and security system applications. Eventually, it would be beneficial to have the task allocation and feedback coordination strategy integrated in the three-tiered hierarchical multi-robot system for urban search and rescue being developed at Victoria University of Wellington. This would test the effectiveness of the reduced human user input system in the real world.

In all the navigation and exploration experiments presented in this thesis, the robots were assumed to know their global position at all times. Long term experiments in the real world without external devices such as GPS or beacons will cause localisation uncertainties as (for example) odometry or inertial navigation system (INS) errors will accumulate. Such errors will also cause distortions in the maps created during exploration. Thus, the implementation of a robust simultaneous localisation and mapping (SLAM) algorithm would be beneficial to reduce these uncertainties and errors.

The presented navigation system employs empirically tuned fixed parameter values and simple weighted linear objective functions. An extension to the existing implementation would be to design objective functions using fuzzy inference systems. The performance of a non-linear fuzzy system utilising various rule combinations can be investigated. Furthermore, neuro-adaptive techniques can be employed to train the weightings of rules in the objective function fuzzy inference systems for optimal navigation system performance.

In the task allocation component, fuzzy inference systems have been employed to simplify human user input and compute weighted sums in the task allocation process. It would be useful to extend the use of fuzzy systems to the feedback system. For instance, if a human user is required to input threshold, weight and monitor time values, these can be specified as graded inputs (say for example, ‘low’, ‘medium’ or ‘high’). Following this, a fuzzy inference system can be developed to detect robot failures. Furthermore, the feedback fuzzy system can utilise neuro-adaptive techniques to self-tune thresholds and weights if user input is incorrect. Presently, the developed feedback system is able to function appropriately without incorporating hysteresis loops for avoiding false detections due to noise during task reallocation. It would be useful to investigate the incorporation of hysteresis loops into the task reallocation process to improve feedback system performance.

The current implementation of the feedback system requires an initial test sequence in a simple real environment or some expert knowledge to determine an informed starting estimate of expected achievement data. If this expected achievement data is incorrectly determined, it will affect failure detection. It would be useful to investigate the degree to which the feedback system is tolerant to incorrect achievement data.

9.6 Publications

Five conference proceedings, one book chapter and four international journal articles (two currently under review) have emerged from the research presented in this thesis. They are listed in the subsections below.

Book chapter B1 presents an overview of the concept of utilising a decentralised hierarchical multi-robot system. It also presents a review of cooperative mobile robot control architectures (section 2.6).

Publication C1 presents initial work on the reactive control system for limited sensing robots that employs a hybrid of directional approaches (polar histogram) and velocity space techniques (dynamic window approach). Initial work on the novel two-tiered global path planning strategy for limited memory robots is presented in conference proceeding C2. The application of the hybrid deliberative-reactive navigation system

in the customised multi-level abstracted multi-robot mapping and exploration task utilising heterogeneous mobile robots is presented in conference proceeding C4. Journal paper J2 presents and evaluates the latest version the hybrid reactive-deliberative navigation system based on the work presented in chapter 3. Based on chapter 4, the latest version of the two-tiered global path planning strategy is presented and evaluated in journal paper J3.

Conference proceeding C3 presents initial work on the reduced human user input task allocation and feedback coordination system for limited capability robots. The feedback system component is further developed to detect and correct the three forms of robot failures in publication C5. Journal paper J1 presents and evaluates the feedback system detailed in chapter 5. A reduced human user input task allocation system for non-expert users that employs fuzzy inference systems (detailed in chapter 5 and chapter 7) is presented and evaluated in journal paper J4.

9.6.1 Refereed Conference Proceedings

C1. P. Chand and D. A. Carnegie, "Reactive control of a tricycle mobile robot," in *Proceedings of the Twelfth Electronics New Zealand Conference*, 2005, pp. 129-134.

C2. P. Chand and D. A. Carnegie, "Memory-time tradeoffs in a path planning approach utilising limited memory robots," in *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2007, pp. 243-248.

C3. P. Chand and D. A. Carnegie, "Task Allocation and Coordination for Limited Capability Mobile Robots," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2007.

C4. C. P. Lee-Johnson, P. Chand, and D. A. Carnegie, "Applications of a Adaptive Hierarchical Mobile Robot Navigation System," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2007.

C5. P. Chand and D. A. Carnegie, "Feedback coordination of limited capability mobile robots," in *Proceedings of the International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 531-536.

9.6.2 Book Chapters

B1. D. A. Carnegie, A. D. Payne, and P. Chand, "The design of a pair of identical mobile robots to investigate cooperative behaviours," in *Cutting Edge Robotics – Section V Multi-Robot Systems*. Austria: International Journal of Advanced Robotic Systems, 2005, pp. 377-396.

9.6.3 International Journal Articles

J1. P. Chand and D. A. Carnegie. "Feedback coordination of limited capability mobile robots," *International Journal of Intelligent Systems Technologies and Applications*, vol. 8, no. 1/2/3/4, pp. 144-157, 2010.

J2. P. Chand and D. A. Carnegie, “Development of a Navigation System for Heterogeneous Mobile Robots.” *International Journal of Intelligent Systems Technologies and Applications*, vol. 10, no. 3, pp. 250-278, 2011.

J3. P. Chand and D. A. Carnegie, “A Two-Tiered Global Path Planning Strategy for Limited Memory Mobile Robots.” *Robotics and Autonomous Systems*. (under review)

J4. P. Chand and D. A. Carnegie, “A Multi-Robot Task Allocation Technique Using Fuzzy Inference Systems," *Robotica*. (under review)

9.7 Summary of Achievements (Contributions)

This thesis has made a number of contributions to mobile robotics research. A hierarchical hybrid deliberative-reactive navigation system for heterogeneous mobile robots with limited sensing and processing capabilities has been developed. It is capable of offering a high degree of flexibility for navigation in known and unknown environments. While the system relies on a number of empirically tuned parameters, a fixed set of parameters has been functional in the various robot and environment configurations tested. A reactive controller comprising a modified dynamic window method (velocity space technique) and a polar histogram (directional method) outperforms reactive control that relies only on the dynamic window method. The developed reactive controller is able to offer similar performance to hybrid deliberative-reactive navigation when utilised in unknown environments. This removes the need for periodic path planning when employing hybrid navigation in unexplored obstructed environments. Subsequently, this can be beneficial in a multi-

robot system where paths are planned on other mobile robots due to processing constraints.

A novel global path planning technique that utilises the memory of a computationally powerful robot but executes entirely on a limited memory mobile robot has been developed. This allows global path planning to be decentralised in a hierarchical heterogeneous multi-robot system instead of relying on a single computationally powerful robot. The two-tiered A* based path planning strategy is able to achieve superior or comparable execution times to non-memory constrained path planning when small sized local maps (such as 64 KB or 128 KB) are employed in large global environments (such as 38.15 MB). In large global environments, the distance traversed by a robot can be greater than the communication range. This would require additional communication nodes or relocation of the computationally powerful robot when replanning is needed. Reduced search space of the memory constrained planning technique at higher obstacle densities (≥ 20%) can potentially affect obstacle clearance. However, when obstacles are sufficiently large to block some local map boundaries, the back tracking algorithm can find a path through alternative local maps as long as exit points exist in them. This technique for global path planning is unique when compared to other memory constrained path planning methods reported in the literature.

Furthermore, a novel reduced human user input task allocation and feedback coordination mechanism for the efficient execution of a global task by limited capability mobile robots has been successfully developed. It is well suited for hierarchical heterogeneous multi-robot systems. The task allocation process has the advantage of employing fuzzy inference systems to permit simplified human user inputs (such as “low”, “medium”, or “high”) for physical robot capability requirements at the task specification stage. This is unlike other task allocation methods reported in the literature that require more detailed expert knowledge for specifying tasks. However, additional effort is required by an expert user to design the reduced human user input system. A simple greedy technique allows tasks to be allocated quickly.

The success of task allocation is monitored with a feedback system that detects and corrects abnormalities during task execution thus improving the performance of task execution. This mitigates the potentially sub-optimal initial greedy allocations and handles unexpected robot failures (such as hardware failure or failures due to inaccurate task specification). The developed feedback system is robust to weight and threshold variation within the tested limits. This allows weights and thresholds to be intuitively selected without negatively affecting performance. Additionally, dependence on communication can be varied as the feedback system is robust to monitor time interval variation within the tested limits. An expert user is expected to "calibrate" a robot's performance only when the task type is altered. Hence, the developed task allocation and feedback coordination strategy has the ability to specify tasks in a generic format such that non-expert human users can adapt and utilise multi-robot systems.

References

[1] H. S. Shim, H. S. Kim, M. J. Jung, I. H. Choi, J. H. Kim, and J. O. Kim, "Designing distributed control architecture for cooperative multi-agent system and its real-time application to soccer robots," *Robotics and Autonomous Systems*, vol. 21, no. 2, pp. 149-165, 1997.

[2] J. L. de la Rosa, A. Oller, J. Vehi, and J. Puyol, "Soccer team based on agent-oriented programming," *Robotics and Autonomous Systems*, vol. 21, no. 2, pp. 167-176, 1997.

[3] E. Pagello, A. D'Angelo, and E. Menegatti, "Cooperation issues and distributed sensing for multirobot systems," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1370-1383, 2006.

[4] M. J. Mataric, M. Nilsson, and K. T. Simsarian, "Cooperative multi-robot box-pushing," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995, pp. 556-561.

[5] N. Miyata, J. Ota, T. Arai, and H. Asama, "Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 769-780, 2002.

[6] B. Donald, L. Gariepy, and D. Rus, "Distributed manipulation of multiple objects using ropes," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000, pp. 450-457.

[7] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A. J. Ganino, M. Garrett, S. S. Joshi, and P. S. Schenker, "CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 33, no. 5, pp. 550-559, 2003.

[8] B. Yamauchi, "Decentralized coordination for multirobot exploration," *Robotics and Autonomous Systems*, vol. 29, no. 2-3, pp. 111-118, 1999.

[9] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 3016-3023.

[10] A. W. Stroupe and T. Balch, "Value-based action selection for observation with robot teams using probabilistic techniques," *Robotics and Autonomous Systems*, vol. 50, no. 2-3, pp. 85-97, 2005.

[11] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376-386, 2005.

[12] L. E. Parker, "ALLIANCE: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics*, vol. 14, no. 2, pp. 220-240, 1998.

[13] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 637-649, 2006.

[14] T. Balch, "Reward and diversity in multirobot foraging," in *Proceedings of the IJCAI-99 Workshop on Agents Learning About, From and With Other Agents*, 1999.

[15] D. A. Carnegie, A. D. Payne, and P. Chand, "The design of a pair of identical mobile robots to investigate cooperative behaviours," in *Cutting Edge Robotics – Section V Multi-Robot Systems*. Austria: International Journal of Advanced Robotic Systems, 2005, pp. 377-396.

[16] D. A. Williamson and D. A. Carnegie, "Embedded platform for search and rescue applications," in *Proceedings of the International Conference on Autonomous Robots and Agents*, 2006, pp. 373-378.

[17] C. L. Cawley, *The Enhancement of a Multi-Terrain Mechatron for Autonomous Outdoor Applications*. MSc Thesis, University of Waikato, 2006.

[18] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of

hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3-34, 1995.

[19] B. P. Gerkey and M. J. Mataric, " Multi-robot task allocation: analyzing the complexity and optimality of key architectures," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 3862-3868.

[20] A. Farinelli, L. Iocchi, and D. Nardi, "Multirobot systems: a classification focused on coordination," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no. 5, pp. 2015-2028, 2004.

[21] L. E. Parker and F. Tang, "Building multirobot coalitions through automated task solution synthesis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1289-1305, 2006.

[22] L. E. Parker, "L-ALLIANCE: task-oriented multi-robot learning in behavior-based systems," *Advanced Robotics*, vol. 11, no. 4, pp. 305-322, 1997.

[23] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14-23, 1986.

[24] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 92-112, 1989.

[25] R. C. Arkin, *Behavior-Based Robotics*. Cambridge Massachusetts: The MIT Press, 1998.

[26] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 304-312, 1992.

[27] M. J. Mataric and F. Michaud, "Behavior-based systems," in *Springer Handbook of Robotics*, vol. part E, B. Siciliano and O. Khatib, Eds. Berlin: Springer, 2008, pp. 891-909.

[28] D. Jung and A. Zelinsky, "An architecture for distributed cooperative planning in a behaviour-based multi-robot system," *Robotics and Autonomous Systems*, vol. 26, no. 2-3, pp. 149-174, 1999.

[29] J. Rosenblatt, "DAMN: a distributed architecture for mobile navigation," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 339-360, 1997.

[30] J. S. Albus, "Outline for a theory of intelligence," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 3, pp. 473-509, 1991.

[31] A. Kosaka and A. C. Kak, "Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties," in *Proceedings of the Proceedings of the 1992 lEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992, pp. 2177-2186.

[32] R. C. Arkin and T. Balch, "AuRA: principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175-189, 1997.

[33] D. M. Lyons and A. J. Hendriks, "Planning as incremental adaptation of a reactive system," *Robotics and Autonomous Systems*, vol. 14, no. 4, pp. 255-288, 1995.

[34] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," in *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 1992, pp. 809-815.

[35] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, "The saphira architecture: a design for autonomy," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 215-235, 1997.

[36] P. Chand and D. A. Carnegie, "Reactive control of a tricycle mobile robot," in *Proceedings of the Twelfth Electronics New Zealand Conference*, 2005, pp. 129-134.

[37] C. P. Lee-Johnson, P. Chand, and D. A. Carnegie, "Applications of a Adaptive Hierarchical Mobile Robot Navigation System," in *Australasian Conference on Robotics and Automation*, vol. 1. Brisbane, Australia: ARAA, 2007.

[38] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23-33, 1997.

[39] P. Ogren and N. E. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 188-195, 2005.

[40] J. A. Primbs, V. Nevistic, and J. C. Doyle, "Nonlinear optimal control: A control Lyapunov function and receding horizon perspective," *Asian Journal of Control*, vol. 1, no. 1, pp. 14-24, 1999.

[41] L. Guoyang, W. Genxia, and W. Wei, "ND-DWA: a reactive method for collision avoidance in troublesome scenarios," in *Proceedings of the The Sixth World Congress on Intelligent Control and Automation*, 2006, pp. 9307-9311.

[42] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 446-451.

[43] R. Murphy, *Introduction to AI Robotics*. Cambridge MA: MIT Press, 2000.

[44] K. Macek, I. Petrovic, and E. Ivanjko, "An approach to motion planning of indoor mobile robots," in *Proceedings of the IEEE International Conference on Industrial Technology*, 2003, pp. 969-973.

[45] M. Seder, K. Macek, and I. Petrovic, "An integrated approach to real-time mobile robot control in partially known indoor environments," in *Proceedings of the Annual Conference of IEEE Industrial Electronics Society*, 2005, pp. 1785-1790.

[46] A. Stentz, "Optimal and efficient path planning for partially-knownenvironments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1994, pp. 3310-3317.

[47] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999, pp. 341-346.

[48] K. O. Arras, J. Persson, N. Tomatis, and R. Siegwart, "Real-time obstacle avoidance for polygonal robots with a reduced dynamic window," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 3050-3055.

[49] C. Schlegel, "Fast local obstacle avoidance under kinematic and dynamic constraints," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998, pp. 594-599.

[50] J. Minguez, "The obstacle-restriction method for robot obstacle avoidance in difficult environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2284-2290.

[51] C. P. Lee-Johnson and D. A. Carnegie, "Towards a Computational Model of Affect for the Modulation of Mobile Robot Control Parameters," in *Proceedings of the International Conference on Autonomous Robots and Agents*, 2006, pp. 367-372.

[52] C. P. Lee-Johnson, *Emotion-based Parameter Modulation for a Mobile Robot Planning and Control System*. PhD Thesis, Victoria University of Wellington, 2008.

[53] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90-98, 1986.

[54] M. Khatib and R. Chatila, "An extended potential field approach for mobile robot sensor-based motion," in *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS-4)*, 1995, pp. 490-496.

[55] H. J. S. Feder and J. J. E. Slotin, "Real-time path planning using harmonic potentials in dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1997, pp. 874-881.

[56] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1993, pp. 802-807.

[57] J. Minguez and L. Montano, "Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45-59, 2004.

[58] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1998, pp. 1572-1577.

[59] R. Simmons, "The curvature velocity method for local obstacle avoidance," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996, pp. 3375-3382.

[60] M. Khatib, H. Jaouni, R. Chatila, and J. P. Laumond, "Dynamic path modification for car-like nonholonomic mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1997, pp. 2920-2925.

[61] J. Minguez and L. Montano, "Robot navigation in very complex, dense, and cluttered indoor/outdoor environments," in *Proceedings of the IFAC World Congress on Automatic Control*, 2002.

[62] J. Minguez and L. Montano, "The ego-kinodynamic space: collision avoidance for any shape mobile robots with kinematic and dynamic constraints," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 637-643.

[63] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000, pp. 2505-2511.

[64] N. Y. Ko and R. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998, pp. 1615-1621.

[65] P. Chand and D. A. Carnegie, "Memory-time tradeoffs in a path planning approach utilising limited memory robots," in *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2007, pp. 243-248.

[66] S. Thrun, "Robotic mapping: a survey," in *Exploring artificial intelligence in the new millennium*, G. Lakemeyer and B. Nebel, Eds. San Francisco, California: Morgan Kaufmann Publishers Inc., 2003, pp. 1-35.

[67] R. Siegwart and Nourbakhsh, *Introduction to Autonomous Mobile Robots*. MA: The MIT Press, 2004.

[68] N. J. Nilsson, "A mobile automaton: an application of artificial intelligence techniques," in *Proceedings of the First International Conference on Artificial Intelligence*, 1969, pp. 509-520.

[69] F. Aurenhammer, "Voronoi diagrams: a survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345-405, 1991.

[70] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading Massachusetts: Addison-Wesley, 1984.

[71] B. Bakker, Z. Zivkovic, and B. Krose, "Hierarchical dynamic programming for robot path planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2756-2761.

[72] S. Sekhavat, P. Svestka, J. P. Laumond, and M. H. Overmars, "Multilevel Path Planning for Nonholonomic Robots Using Semiholonomic Subsystems," *The International Journal of Robotics Research*, vol. 17, no. 8, pp. 840-857, 1998.

[73] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21-71, 1998.

[74] F. Wallner, M. Kaiser, H. Freidrich, and R. Dillman, "Integration of topological and geometrical planning in a learning mobile robot," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1994, pp. 1-8.

[75] C. W. Warren, "Fast path planning using modified A* method," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1993, pp. 662-667.

[76] Y. Zhao, C. V. Ravishankar, and S. L. BeMent, "Coping with Limited On-Board Memory and Communication Bandwidth in Mobile-Robot Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 1, pp. 58-72, 1994.

[77] S. Koenig, "Agent-centered search," *AI Magazine*, vol. 22, no. 4, pp. 109-131, 2001.

[78] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2383-2388.

[79] K. Doki, S. Hayakawa, T. Suzuki, S. Okuma, and T. Aoki, "Hierarchical memory structure for the real-time search for action acquisition of an autonomous mobile robot," in *Proceedings of the IEEE International Symposium on Intelligent Control*, 2002, pp. 815-820.

[80] A. A. Razavian and J. Sun, "Cognitive Based Adaptive Path Planning Algorithm for Autonomous Robotic Vehicles," in *Proceedings of the IEEE SoutheastCon*, 2005, pp. 153-160.

[81] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald, "Hierarchical A*: Searching Abstraction Hierarchies Efficiently," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996, pp. 530-535.

[82] M. DeLoura, *Game Programming Gems*. Boston MA: Charles River Media, 2000.

[83] M. Schneider, M. Guthe, and R. Klein, "Real-time Rendering of Complex Vector Data on 3D Terrain Models," in *Proceedings of the 11th International Conference on Virtual Systems and Multimedia*, 2005, pp. 573-582.

[84] D. Z. Chen, R. J. Szcerba, and J. J. Uhran, "A Framed-Quadtree Approach for Determining Euclidean Shortest Paths in a 2-D Environment," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 5, pp. 668-681, 1997.

[85] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt, "Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1998, pp. 650-655.

[86] R. Korf, "Depth-first iterative deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, no. 1, pp. 97-109, 1985.

[87] R. Korf, "Linear-space best-first search," *Artificial Intelligence*, vol. 62, no. 1, pp. 41-78, 1993.

[88] R. Korf, W. Zhang, I. Thayerand, and H. Hohwald, "Frontier Search," *Journal of the ACM*, vol. 52, no. 5, pp. 715-748, 2005.

[89] R. Zhou and E. Hansen, "Sparse-memory graph search," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003, pp. 1259-1266.

[90] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7-27, 1997.

[91] T. Fukuda and G. Iritani, "Construction mechanism of group behaviour with cooperation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995, pp. 535-542.

[92] P. Glorennec, "Coordination between autonomous robots," *International Journal of Approximate Reasoning*, vol. 17, no. 4, pp. 433-446, 1997.

[93] C. Sossai, P. Bison, G. Chemello, and G. Trainito, "Sensor fusion for localization using possibility theory," *Control Engineering Practice*, vol. 7, no. 6, pp. 773-782, 1999.

[94] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939-954, 2004.

[95] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, "First results in the coordination of heterogeneous robots for large-scale assembly," in *Experimental Robotics VII*, vol. 271, *Lecture Notes in Control and Information Sciences*. London, UK: Springer-Verlag, 2000, pp. 323-332.

[96] F. R. Noreils, "Toward a robot architecture integrating cooperation between mobile robots: application to indoor environment," *International Journal of Robotics Research*, vol. 12, no. 1, pp. 79-98, 1993.

[97] M. B. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2714-2720.

[98] S. C. Botelho and R. Alami, "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999, pp. 1234-1239.

[99] L. Chaimowicz, V. Kumar, and M. F. M. Campose, "A paradigm for dynamic coordination of multiple robots," *Autonomous Robots*, vol. 17, no. 1, pp. 7-21, 2004.

[100] M. B. Dias and A. Stentz, "A market approach to multi-robot coordination," Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU-RI -TR-01-26, August 2001.

[101] B. P. Gerkey and M. J. Mataric, "Sold! auction methods for multi-robot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758-768, 2002.

[102] B. B. Werger and M. J. Mataric, "Broadcast of local eligibility for multi-target observation," in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, 2000, pp. 347-356.

[103] I. D. Chase, M. Weissburg, and T. H. Dewitt, "The vacancy chain process: a new mechanism of resource distribution in animals with application to hermit crabs," *Animal Behavior*, vol. 36, pp. 1265-1274, 1988.

[104] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, "Multi-robot task allocation through vacancy chains," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 2293-2298.

[105] M. J. Mataric, G. S. Sukhatme, and E. H. Ostergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2-3, pp. 255-263, 2004.

[106] M. B. Dias and A. Stentz, "Traderbots: a market-based approach for resource, role, and task allocation in multirobot coordination," Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU-RI-TR-03-19, August 2003.

[107] R. Zlot and A. Stentz, "Complex task allocation for multiple robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005, pp. 1515-1522.

[108] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, no. 2, pp. 241-273, 1999.

[109] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed multirobot exploration and mapping," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1325-1339, 2006.

[110] N. R. Jennings, "Controlling cooperative problem solving in industrial multi-agent systems," *Artificial Intelligence*, vol. 75, no. 2, pp. 195-240, 1995.

[111] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 83-124, 1997.

[112] P. R. Cohen and H. Levesque, "Teamwork," *Nous*, vol. 25, no. 4, pp. 487-512, 1991.

[113] B. J. Grosz, "Collaborative systems," *AI Magazine*, vol. 17, no. 2, pp. 67-85, 1996.

[114] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 165-200, 1998.

[115] L. Vig and J. A. Adams, "Market-based multi-robot coalition formation," in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, 2006, pp. 227-236.

[116] B. Kannan and L. E. Parker, "Fault-tolerance based metrics for evaluating system performance in multi-robot teams," in *Performance Metrics for Intelligent Systems Workshop*. Gaithersburg, Maryland, 2006.

[117] B. Kannan and L. E. Parker, "Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot teams," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 951-958.

[118] M. T. Long, R. R. Murphy, and L. E. Parker, "Distributed multi-agent diagnosis and recovery from sensor failures," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 2506-2513.

[119] R. R. Murphy and D. Hershberger, "Handling sensing failures in autonomous mobile robots," *The International Journal of Robotics Research*, vol. 18, no. 4, pp. 382-400, 1999.

[120] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Proceedings of the National Conference on Artificial Intelligence*, 2000, pp. 852-858.

[121] B. Tovar, L. Muñoz-Gómez, R. Murrieta-Cid, M. Alencastre-Miranda, R. Monroy, and S. Hutchinson, "Planning exploration strategies for simultaneous localization and mapping," *Robotics and Autonomous Systems*, vol. 54, no. 4, pp. 314-331, 2006.

[122] K. Singh and K. Fujimura, "Map making by cooperating mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1993, pp. 254-259.

[123] J. V. D. Diosdado, *Behaviour Based Simulated Low-Cost Multi-Robot Exploration*. PhD Thesis, University of Edinburgh, 2006.

[124] S. Thrun, "A probabilistic on-line mapping algorithm for teams of mobile robots," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335-363, 2001.

[125] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun, "Map building with mobile robots in dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 1557-1563.

[126] J. W. Fenwick, P. M. Newman, and J. J. Leonard, "Cooperative concurrent mapping and localization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 1810-1817.

[127] A. K. Poernomo and H. S. Ying, "New cost function for multi-robot exploration," in *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision*, 2006, pp. 1-6.

[128] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the International Conference on Autonomous Agents*, 1998, pp. 47-53.

[129] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000, pp. 476-481.

[130] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Magazine*, vol. 9, no. 2, pp. 61-74, 1988.

[131] C. P. Lee-Johnson, *The Development of a Control System for an Autonomous Mobile Robot*. MSc Thesis, University of Waikato, 2004.

[132] P. Chand and D. A. Carnegie, "Feedback coordination of limited capability mobile robots," in *Proceedings of the International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 531-536.

[133] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston Massachusetts: Addison Wesley Longman, 2001.

[134] Hacker Friendly LLC. (2007). Wireless Networking in the Developing World: A practical guide to planning and building low-cost telecommunications infrastructure. [Online]. Available: http://wndw.net/pdf/wndw2-en/wndw2-ebook.pdf.

[135] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*. Stanford, California: Heuris Tech Press, 1981.

[136] L. Reznik, *Fuzzy Controllers*. Oxford Great Britain: Newnes (Butterworth-Heinemann), 1997.

[137] RoboProbe Technologies, Inc. (2009). Model 2 - Platform with Claw-Arm. [Online]. Available: http://www.roboprobe.com/catalog.aspx?productid=2.

[138] E.-G. Talbi, *Metaheuristics: From Design to Implementation* Hoboken New Jersey: Wiley, 2009.

[139] T. J. Richer and D. R. Corbett, "A dynamic territorial robotic system," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2004.

[140] M. Schneider-Fontan and M. J. Mataric, "Territorial multi-robot task division," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 815-822, 1998.

[141] H. Choset, "Coverage for robotics - a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113-126, 2001.

[142] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, 2nd ed. New York: Addison-Wesley, 2006.

[143] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed. New York: Addison-Wesley, 1996.

[144] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI Magazine*, vol. 17, no. 3, pp. 73-83, 1996.

[145] D. Avis, "A survey of heuristics for the weighted matching problem," *Networks*, vol. 13, no. 1, pp. 475-493, 1983.