Cooperative Navigation for Teams of Mobile Robots

by

Mike Peasgood

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Mike Peasgood

Abstract

Teams of mobile robots have numerous applications, such as space exploration, underground mining, warehousing, and building security. Multi-robot teams can provide a number of practical benefits in such applications, including simultaneous presence in multiple locations, improved system performance, and greater robustness and redundancy compared to individual robots. This thesis addresses three aspects of coordination and navigation for teams of mobile robots: *localization*, the estimation of the position of each robot in the environment; *motion planning*, the process of finding collision-free trajectories through the environment; and *task allocation*, the selection of appropriate goals to be assigned to each robot. Each of these topics are investigated in the context of many robots working in a common environment.

A particle-filter based system for cooperative global localization is presented. The system combines the sensor data from three robots, including measurements of the distances between robots, to cooperatively estimate the global position of each robot in the environment. The method is developed for a single triad of robots, then extended to larger groups of robots. The algorithm is demonstrated in a simulation of robots equipped with only simple range sensors, and is shown to successfully achieve global localization of robots that are unable to localize using only their own local sensor data.

Motion planning is investigated for large teams of robots operating in tunnel and corridor environments, where coordinated planning is often required to avoid collision or deadlock conditions. A complete and scalable motion planning algorithm is presented and evaluated in simulation with up to 150 robots. In contrast to popular decoupled approaches to motion planning (which cannot guarantee a solution), this algorithm uses a multi-phase approach to create and maintain obstacle-free paths through a graph representation of the environment. The resulting plan is a set of collision-free trajectories, guaranteeing that every robot will reach its goal.

The problem of task allocation is considered in the same type of tunnel

and corridor environments, where tasks are defined as locations in the environment that must be visited by one of the robots in the team. To find efficient solutions to the task allocation problem, an optimization approach is used to generate potential task assignments, and select the best solution. The multi-phase motion planner is applied within this system as an efficient method of evaluating potential task assignments for many robots in a large environment. The algorithm is evaluated in simulations with up to 20 robots in a map of large underground mine.

A real-world implementation of 3 physical robots was used to demonstrate the implementation of the multi-phase motion planning and task allocation systems. A centralized motion planning and task allocation system was developed, incorporating localization and time-dependent trajectory tracking on the robot processors, enabling cooperative navigation in a shared hallway environment.

Acknowledgements

The developments in this thesis build on foundations of mobile robotics research that have been established over the past several decades. The individuals whose names fill the bibliography of this work have created a fascinating field of research, entertainment, and great practical value, to which I make a small contribution. I am grateful for their insights passed through the literature, and for their contributions to this research through reviews of presentations and papers.

I thank my supervisors, Chris Clark and John McPhee, for their guidance, wisdom, and patience, as they advised and supported my research. I appreciate Chris' hours of brainstorming, reviewing, and revising, as well as feedback from the rest of our team in the LAIR lab. John has been a tremendous role model and advisor. He has led me into rewarding graduate research (more than once) and his guidance and assistance through the academic system have been invaluable.

Tim Barfoot was an excellent supervisor during my research at MDA Space Missions, and I appreciate his support and encouragement. I would also like to thank Frontline Robotics for the contribution of three robots and technical support to the project. This work was supported in part by the National Sciences and Engineering Research Council of Canada (NSERC) and Ontario Centres of Excellence (OCE).

I thank my parents for instilling in me the value of education and the principle that anything worth doing is worth doing right.

Finally, I am most grateful for the continual love and support of my wife Kelly, whose endless patience and kindness have carried me through the good times and the rough times of this project, and all my endeavors. Thank you.

Contents

List of Figures

Chapter 1

Introduction

The number of applications for mobile robots has grown significantly in recent years, and continues to increase. The recent Mars exploration missions of Spirit and Opportunity have demonstrated the practicality and value of autonomous mobility in space, allowing access to research areas inaccessible to humans. Closer to home, robots can be found delivering mail in office buildings, assisting the elderly and disabled in hospitals and retirement homes, guiding tours in museums, cleaning floors in residential homes, and assisting in military operations. According to the UNECE 2004 World Robotics Survey, "At the end of 2003, about 610,000 autonomous vacuum cleaners and lawn-mowing robots were in operation. In 2004-2007, more than 4 million new units are forecasted to be added." [83].

In many consumer, industrial, and research applications, teams of multiple robots can provide a number of practical benefits:

Distributed Presence: Multiple robots can make simultaneous measurements from multiple locations. This is a significant feature for some research tasks, such as environment monitoring of a large area.

Force Multiplication: In applications requiring the movement of heavy items, cooperative teams of multiple robots can move objects too heavy for a single robot.

Improved Performance: Multiple robots have the potential to achieve a set of tasks more quickly than an individual by dividing the task into smaller subtasks, each assigned to one member of the team. In addition, by coordinating their tasks, a *cooperative* team may be able to accomplish a set of tasks more efficiently than a set of robots operating independently.

Robustness and Redundancy: Well-structured teams of multiple robots can increase overall system reliability by allowing a mission to continue despite the failure of one member. This can be particularly valuable for high-risk tasks such as space exploration.

These potential benefits come with the additional challenges of effectively managing and controlling teams of multiple mobile robots, each robot of which typically has some degree of autonomy in controlling its own motion, perception, and communication. A significant research effort has recently developed to address the challenges in the field of multi-robot control and navigation; this thesis is a contribution to that effort.

The research presented in this thesis investigates three aspects of navigation for a team of mobile robots:

1. Localization: Where are each of the robots in the environment?
2. Task Allocation: Where should each of the robots go?
3. Motion planning: What is the best path for each robot to follow to reach its goal?

The developments in this thesis build on a wealth of prior research in the areas of mobile robot control systems, multi-robot system architecture, localization algorithms, motion planning algorithms, and task allocation systems. This introductory chapter reviews these topics, and gives an overview of recent literature in multi-robot systems and architectures. More detailed reviews of the current research into the problems of localization, motion planning and task allocation are presented in the respective chapters that follow.

1.1 Mobile Robot Control Architectures

Two different strategies have been commonly used to control mobile robots: a behaviour-based approach, based on the emergence of complex behaviours from a set of simple rules; and a planning-based approach, based on a model of the robot and its environment.

Behaviour-based robot control, also referred to as "reactive control", uses a set of rules to determine what action a robot will take under certain circumstances. By assembling a large set of these simple rules, some interesting (and potentially useful) robot behaviour can be developed, such as a robot vacuum cleaner moving around a house. Rodney Brooks formalized one behaviour-based approach in the Subsumption Architecture [5], which has proven to be a useful method for controlling groups of simple robots to perform various tasks. By incorporating behaviours that respond to the observation of other robots, some multi-robot behaviours have been developed, such as playing tag and searching for explosives [52]. Reactive algorithms are attractive due to their low computational and communication requirements, and apparent simplicity. However, developing the behaviour rules to accomplish a particular task can be challenging and time-consuming.

An alternative is a planning-based approach to robot control, in which the robot develops and executes a plan to navigate through the environment and accomplish specified tasks. This approach enables generic navigation systems to be used for a wide variety of applications. Planning-based control involves a number of elements:

- A *map*, or representation of the working environment of the robot.
- A *localization* system, which enables the robot to estimate its position within the *map*.
- A *planning* system, which can determine a plan, or route from the current position estimated by the *localization* system to a goal position in the map.

- A *motion execution* system to control the robot actuators and follow the computed plan.

Behaviour-based and planning-based approaches can be used together to create a *hybrid* control system. A hybrid controller can take advantage of the fast response of a reactive system for real-time control requirements (such as obstacle avoidance), while using the world model of a planning-based approach for navigation.

A hierarchical architecture is often used to decompose a robot control system and define interfaces between each of the components. Figure 1.1 shows a typical control hierarchy of a hybrid controller, where the lowest level of control is a reactive motion controller with obstacle-avoidance, and the upper levels implement planning-based navigation.

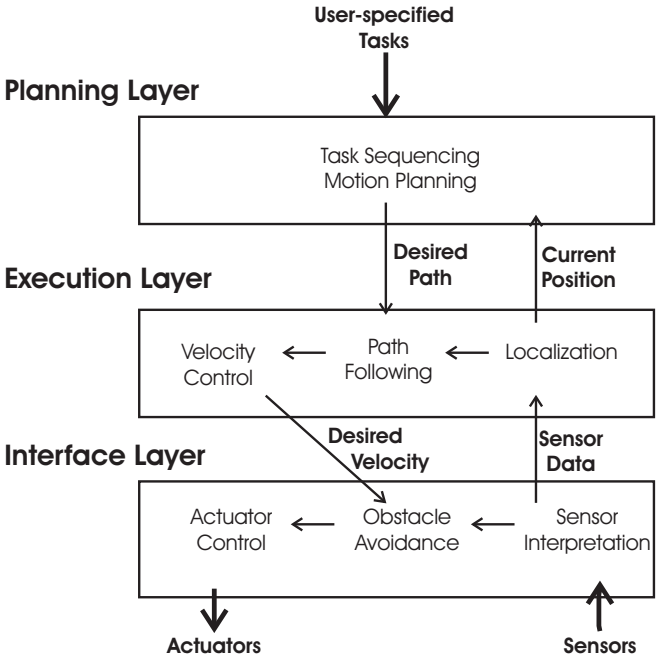


Figure 1.1: A three layer hierarchical control architecture

At the highest level of the architecture, a motion planner finds an obstacle-free path through the environment from the current position to the goal po-

sition (which is specified by an operator or task allocation system). This path is passed to the execution layer, which determines suitable velocities that will guide the robot along the specified path. The angular and translational velocities are updated in real-time, based on the current position and velocity of the robot as determined by the localization system, and passed to the interface layer.

At the lowest level is the interface between the robot and its environment, through sensors and actuators. This may be achieved using a behaviour-based rule system, defining actuator responses to sensor inputs, to drive in the desired direction while avoiding obstacles. Alternatively, fast feedback control loops can be used at this level to tightly control position and velocity according to the values requested by the execution level, while detecting and observing dynamic obstacles.

The contributions in this thesis are developed in the context of the planning-based approach to robot control, and address the upper two layers of the control hierarchy shown in Figure 1.1.

1.2 Map Representations

A map representation, or model of the environment, is required for planning-based control and navigation. The map allows a robot to

1. localize itself (determine its position in the environment) by comparing sensor readings to the data in the map, and

2. plan its route through the environment, by finding obstacle-free paths between its current position and a goal position.

Different representations have been proposed, and are suitable to different aspects of the navigation problem. Figure 1.2 shows a map based on the floor-plan of a simple building with five rooms connected by a hallway; two different representations are illustrated using this example. The first is the common representation of an *occupancy grid*, proposed by Moravec and Elfes [54] [15].

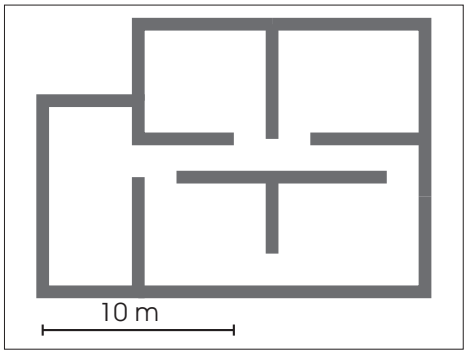
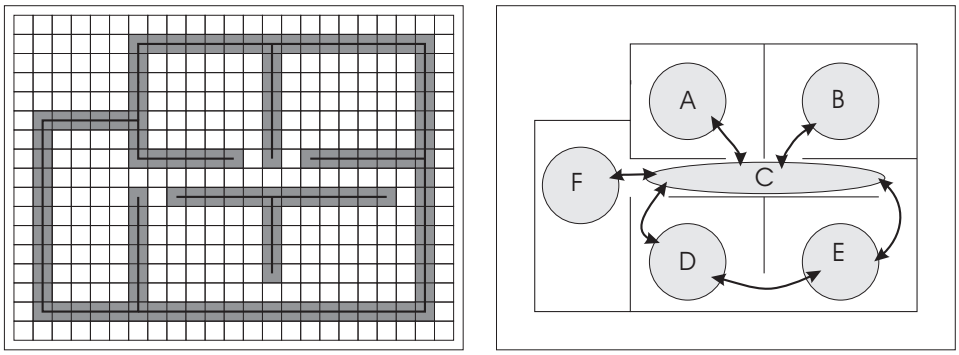


Figure 1.2: A floor-plan map of a simple building structure

In this form, the map consists of a 2 dimensional array of cells corresponding to a grid overlayed on the environment. Walls and other objects that present obstacles to the robot are represented as occupied cells; each array element stores the probability that the corresponding cell is occupied. As shown in Figure 1.3(a), using a 1m square grid on the example floor-plan, the dark cells indicate walls and obstacles that should be avoided, and the light cells indicate open areas for navigation. The grid representation is suitable for localization using range sensors; route planning can be performed by finding obstacle-free paths through adjacent vacant cells.



(a) Occupancy grid map (b) Topological graph map

Figure 1.3: Two representations of the example building floor-plan

A *topological map* represents the connectivity of regions in the environment as a graph structure. As shown in Figure 1.3(b), nodes represent open regions, and edges indicate obstacle-free paths between regions; The direct representation of connectivity makes the graph structure ideally suited to path planning.

Since different representations are well suited to different aspects of navigation, multiple maps may be used within one planning-based architecture. For example, an occupancy grid map may be used for localization, while a topological representation (a Voronoi map for example [9]) may be derived from the occupancy grid and used for motion planning.

1.3 Localization

Localization algorithms can generally be classified by two features: *position tracking versus global localization*, and *absolute versus relative* estimation. The most suitable localization algorithm for a particular system depends on the requirements of the particular application.

1.3.1 Position Tracking versus Global Localization

Position tracking algorithms estimate motion from a previously estimated pose (position and orientation), and are suitable for maintaining an accurate estimate given a good initial estimate. These algorithms may be used when the robot is always placed in a known starting position, or can be given its initial coordinates by a human operator.

Global localization, however, makes no assumption of an accurate initial estimate in order to estimate the pose of the robot in a map. As a result, a global localization algorithm is typically more capable of recovering from large localization errors. Unlike position tracking localization, global localization can potentially solve the *kidnapped robot* problem [80], that is, where the robot has a confident estimate of its pose, and is then moved to a new position in the map without any knowledge of the relocation.

1.3.2 Absolute versus Relative Localization

Absolute localization methods determine the pose of the robot with respect to a map with a pre-defined, world-fixed coordinate system. In contrast, in multi-robot systems *relative localization* determines the pose with respect to the other objects or robots in the team, as in the system developed by Grabowski and Khosla [27]. Relative localization can be useful for coordinating the motion between robots, such as maintaining formations. However, while the positions of other robots can be used to reduce the position uncertainty of an individual, when all of the robots in the team eventually move, the incremental errors result in unbounded uncertainty in the estimated absolute position of the team with respect to the world.

This thesis addresses the problem of *global* and *absolute* localization of a multi-robot team, estimating the position of each robot with respect to predefined real-world coordinates in a known map of the environment. Note, however, that relative localization can be an important step within the absolute localization process, since the relative positioning of robots within the team can be used to incorporate sensor readings from other robots and improve the overall estimate.

1.3.3 Localization Methods

Since localization is fundamentally a state estimation problem, various state estimator algorithms have been applied to mobile robots. The goal of each of these methods is to probabilistically estimate the most likely state, or pose, of the robot, based on its history of motion and sensor inputs.

The Kalman filter method [37] maintains a state estimate using a Gaussian distribution representation. While the Kalman filter gives an optimal estimate for linear systems with known sensor and process noise, extensions such as the *unscented Kalman Filter* (UKF) have been developed to effectively estimate parameters in nonlinear systems [36]. These methods have been successfully applied to the position tracking localization problem, where

the position can be reasonably represented by a single estimate and variance [58]. In *global* localization, however, no initial estimate is given, and many possible estimates may be likely based on the initial sensor inputs. This limitation motivated the use of more general descriptions of the probability distribution of the robot pose.

General Bayesian state estimation methods estimate the probability distribution across the entire state space, and can therefore maintain multiple hypotheses, corresponding to different peaks in the distribution. The Bayesian estimator is capable of tracking these multiple hypotheses until further measurements allow convergence to a single estimate. Unfortunately, it is computationally intensive to maintain a probability distribution across the entire configuration space of the robot [12].

Particle filter state estimation methods maintain the ability to track multiple estimates, but reduce the computational load by maintaining only an approximation of the full probability distribution. The application of particle filters to mobile robot localization is described in detail by Fox [18], and has been successfully used by many other researchers.

1.4 Motion Planning

A motion planning system is used to find an obstacle-free path from the current position (as estimated by the localization system) to a specified goal position. Motion planning for individual robots has been well studied in the literature — refer to Latombe [42] for a detailed review. The A* search algorithm, first presented by Hart [32], has been used extensively in robotics to find the shortest path for a robot through a graph-based map. The A* algorithm is *complete* (guaranteed to find an obstacle-free path between two nodes if one exists), and *optimal* (guaranteed to find the shortest obstacle-free path). It is used as a foundation for motion planning in this thesis.

The complexity of the A* planner increases exponentially with the number of dimensions used to represent the robot pose. For a single robot operat-

ing in a planar map, the pose (or *state* of the robot) is typically represented with either two coordinates of position (x and y), or three coordinates by adding orientation (θ). For motion planning in these two or three dimensions, an A* planner is practical. For a team of multiple robots, a straightforward approach is to create a *system state* that includes the coordinates of all robots in the team. However, as the number of robots r increases, the number of coordinates in the system state increases to $3 \times r$, and the exponential complexity of the A* algorithm makes it impractical for more than 3 or 4 robots.

Some popular algorithms for multi-robot planning manage the complexity of the problem by planning trajectories for robots individually and sequentially [4] [29]; such decoupled methods are not guaranteed to find a solution if one exists. For example, if two robots are required to swap positions in a confined environment, the first trajectory planned for one robot may create an obstacle for the other, and vice versa. Other approaches use a randomized approach, such as probabilistic roadmaps (PRMs) to find mutually collision-free trajectories through the environment [40]. These are efficient, but again are not guaranteed to find a solution. In contrast, Chapter 3 describes the development of a multi-phase approach to the planning problem that guarantees a solution for a pre-defined number of robots in a common environment.

1.5 Task Allocation

Given a set of tasks to accomplish, and a team of multiple robots available to perform the tasks, a system is required to assign each task to a particular robot. The goal of such a system is to allocate the tasks in an optimal manner, minimizing a cost function such as the total time to complete all of the tasks, or the total energy expended by all of the robots.

A common mechanism for achieving a suitable allocation of tasks is based on an economic, or *market-based* model proposed by Smith [76]. The system auctions off each task to the individual robots, which supply bids based on

their estimated cost (in time, distance, or energy) to accomplish the task. Refer to [26] for an example implementation of the approach with mobile robots.

In Chapter 4, the problem of task allocation is investigated in the context of motion planning in tunnel and corridor environments, such as underground mines and office buildings. The tasks in this case are defined by locations in the environment that must be visited by a robot, and the task allocation problem is to direct each robot to an appropriate sequence of task locations. This is similar to the well-studied traveling salesman problem [61], applied to many robots simultaneously. In such problems, coordination between robots is required for effective task allocation and planning; the cost for one robot to reach a particular goal may depend significantly on the motion of other robots occupying nearby tunnels.

1.6 Multi-robot Systems

Systems of multiple autonomous mobile robots can provide significant benefits over individual robots working independently, such as increased redundancy and robustness, simultaneous presence in multiple locations, and the potential to perform cooperative tasks, such as moving objects too heavy for a single robot. Numerous approaches have been proposed and demonstrated to achieve these goals, many of them discussed in a thorough survey by Cao, Fukunaga and Kahng [6].

Multi-robot system architectures can typically be categorized by the nature of cooperation in the system (reactive or planning-based), the independence of decision making by individual robots (centralized or distributed), and whether the robots in the team are distinguishable from one another (homogenous or heterogeneous).

1.6.1 Reactive versus Planning-Based Systems

In the same way that behaviour-based control systems generate complex behaviour in individual robots from a simple set of rules, complex multi-robot system behaviour can emerge from a set of simple robots. This emergent behaviour can be compared to the complex group dynamics observed in biological systems, such as ant colonies and insect swarms [6]. Reactive flocking algorithms can maintain a formation of many simple robots without explicit communication and coordination [67]. Group behaviours of ant colonies, such as clustering, dispersing, and following a leader [53] [25], or guiding the motion of nanobots in medical applications [45] have also been developed based on rule-based systems. The *ALLIANCE* architecture allocates tasks among behaviour-based heterogenous robots by enabling and inhibiting sets of behaviours as robots become aware of their teammates [62].

Reactive algorithms are attractive due to their low computational and communication requirements, and apparent simplicity. However, a significant challenge in the development of such systems is the design of the underlying rules that will produce the desired group response. "Decomposing swarm actions into individual behaviors is a daunting task" [52], limiting reactive systems to applications where the required group behaviour can be encoded in stimulus-response rules.

The alternative planning-based approach explicitly coordinates the actions of many robots to accomplish one or more tasks. The difference between reactive and planning-based systems may be understood by analogy to two types of biological group behaviour, *eusocial* and *cooperative*, observed by McFarland [51] [6]. *Eusocial* behaviour is the group behaviour that emerges from genetically-driven actions that are necessary for survival of individuals (typically insects in a colony). This is comparable to the emergent group behaviour in a team of reactive robots. In contrast, *cooperative* group behaviour is observed in interactions between higher vertebrates, and "is not motivated by innate behavior, but by an intentional desire to cooperate in order to maximize individual utility" [6]. A planning-based approach to multi-robot

systems is analogous to this "intentional" model of cooperation; an algorithmic process is followed to cooperatively and efficiently achieve a well-defined set of tasks.

With a planning-based approach to control, the introduction of multiple robots in the same environment creates the possibility of (and possibly requirement for) interactions between individual robots at each layer of the hierarchical structure shown in Figure 1.1. Considering the problem of potential collisions between independent robots operating in a common environment:

- at the lowest level, robots may use simple signaling protocols or reactive obstacle avoidance to avoid one other;
- at the execution level, trajectories may be adjusted by varying the velocities of each robot to reduce the likelihood of trajectories crossing;
- at the highest level of planning, tasks may be allocated to robots in such a way as to direct robots to different regions of the environment, further reducing the probability of potential interactions or collisions between robots.

At higher levels of the control structure, higher level planning can more effectively coordinate the motion of robots. However, coordination at higher levels often requires more sophisticated logic, to avoid or resolve potential deadlock conditions, as presented in detail in Chapter 3.

1.6.2 Centralized versus Distributed Systems

Multi-robot control architectures can be described as centralized, where planning and coordination is performed at a central processor, or distributed, where each robot navigates independently of the rest of the group. Centralized architectures can benefit from having the state and goals of all robots available at one processor; this allows for more globally optimal planning and task allocation. Distributed approaches can benefit from less reliance

on reliable communication networks, and greater scalability since each robot performs its own navigation processing [10]. Cao et al. note that "it is not clear whether the scaling properties of decentralization offset the coordinative advantage of centralized systems" [6].

Many implementations involve a combination of both architectures, by using a centralized processor to perform high level planning for largely autonomous individuals, such as [59], or by making certain individuals "leaders" within a distributed system [79]. Chien et al. give a comparison of three different task allocation systems, with varying degrees of distributed processing, with simulations of a Mars rover exploration application [8].

1.6.3 Homogeneous versus Heterogeneous Systems

A group of robots can be considered homogenous if all robots are functionally equivalent, and heterogenous otherwise. Homogeneity is assumed (though often not explicitly stated) in many multi-robot systems [45] [52] [72] [69]. This assumption often simplifies the problems of task allocation and motion planning, and is practical in applications involving physically similar robots. While in practice multiple robots are generally not identical due to variations in construction and damage to individuals, the "functionally equivalent" definition allows minor variations to be ignored.

Some frameworks such as *ALLIANCE* [62], *MURDOCH* [23], and more recently *DEMIR-CF* [72] have been developed to explicitly consider heterogeneous groups. By allocating tasks to the robots most capable of performing them, overall efficiency of the system can be improved.

Localization systems can also benefit from heterogeneity, by taking advantage of the characteristics of different sensors available on different robots. This has been applied to outdoor autonomous navigation [48], and with teams of small cooperative robots operating indoors [57] [41]. For example, one robot in the team may be equipped with an absolute positioning system (such as GPS). The other robots, using a relative positioning system (such as vision) can estimate their absolute position based on their pose relative

to the first robot.

In this thesis, the localization system developed in Chapter 2 assumes homogeneous robots, all of which have only simple range-finding sensors. The system does not depend on any one member of the team with an absolute positioning system, which would create a single point of failure. Chapter 4 investigates task allocation and cooperative motion planning for both homogenous and heterogenous teams.

1.6.4 Loosely versus Tightly Coupled Teams

The degree of cooperation and coordination between individual robots in a team is dependent on the nature of the tasks to be performed. A problem involving navigation of several robots in a large open area, where the probability of crossing paths is low, requires a much lower degree of coordination than the navigation of many robots in a confined tunnel environment (the subject of Chapters 3 and 4). This difference can be captured by the descriptions *loosely coupled*, referring to teams where robots coordinate only during task allocation, and *tightly coupled*, where coordination is required during task execution [14].

Dias et al., in a survey of market-based task allocation systems, note that "Tight coordination is extremely challenging: teams cannot easily take advantage of the distributed planning and execution that make loose coordination tractable, and they are rarely fault-tolerant since task success depends on the simultaneous success of multiple teammates" [14].

The planning and task allocation systems developed in this thesis qualify as tightly coupled; successful plan execution requires each robot to follow its trajectory accurately in both space and time. This tight coupling is a requirement for efficient operation in tunnel environments, where coordinated planning is a necessity. The challenges of tractability are handled by use of the centralized and scalable planner developed in Chapter 3, while fault-tolerance is achieved through obstacle avoidance at the lowest level of the control hierarchy and rapid replanning at the highest level if individual robot

failures occur.

1.7 Thesis Outline

This thesis presents the development and validation of algorithms for cooperative localization, planning, and task allocation by a team of mobile robots, suitable for applications involving a large number of robots operating in a shared environment. The contributions of this research are summarized in an outline of the following chapters:

Chapter 2: *A distributed cooperative algorithm for multi-robot localization* presents a new approach for teams of mobile robots to globally localize themselves in a known environment. By combining the sensor data from multiple robots, including measurements of the relative pose between pairs of robots, conventional particle filter localization algorithms can be improved to reduce the computational cost (or conversely, improve the accuracy) of absolute pose estimation for each robot. Further, by estimating the position of multiple robots and distributing the best estimates, members of the team can work cooperatively to localize one another.

The localization method developed in Chapter 2 is well suited to a distributed implementation, where each robot can operate fully autonomously of the rest of the network, but can take advantage of other robots within its local area when possible. This leads to a multi-robot system with significantly better localization performance than individuals operating in isolation, and is robust to failures of individual robots and the communication network.

In contrast to some existing methods, the algorithm presented uses only measurements of the distance between pairs of robots; it requires only range measurements, rather than a system that can estimate the full relative pose of two robots. Sensor measurements of all robots are shared and combined to more efficiently localize all robots in the team. Performance is also improved over existing algorithms by sharing the best pose estimates among the team members.

Chapter 3: *A complete and scalable multi-robot motion planning system* addresses the problem of finding collision-free trajectories for many robots moving towards individual goals in an environment of corridors or tunnels. Most popular algorithms for multi-robot planning manage the complexity of the problem by planning trajectories for robots individually; such decoupled methods are not guaranteed to find a solution if one exists.

In contrast, this chapter describes a multi-phase approach to the planning problem that uses a graph and spanning tree representation to create and maintain obstacle-free paths through the environment for each robot to reach its goal. The resulting algorithm guarantees a solution for a pre-defined number of robots in a common environment.

Chapter 4: *A multi-robot task allocation algorithm* investigates the allocation of a specific type of task, where robots are required to visit certain locations (tasks) in an environment composed primarily of narrow corridors or tunnels, such as underground mines. The multi-phase planner developed in Chapter 3 is applied to this problem, and simulation results demonstrate the practicality and scalability of the task allocation system.

Chapter 5: *A Multi-Robot System Implementation* presents a real-world cooperative multi-robot system, consisting of several mobile robots equipped with scanning laser range finders for localization, and a wireless communication network. The system is used to demonstrate a physical instantiation of the coordinated planning and task allocation algorithms developed in Chapters 3 and 4.

Chapter 6: *Conclusions* summarizes the developments and contributions presented in this thesis, and suggests directions for future research.

Chapter 2

Cooperative Localization for Teams of Robots with Simple Sensors

2.1 Introduction

Cooperating teams of robots can add simultaneous presence, force multiplication, and greater robustness to a robotic mission. In particular, teams of small robots are valuable in a variety of applications such as space exploration, where weight must be minimized to reduce transportation costs.

The autonomy and intelligent behaviour of small robots is typically limited by two factors: computational resources and sensor capabilities. The computational limitations have been addressed by at least two different approaches. One is to use behavior-based control strategies [5], which often have lower computational requirements. However, such strategies navigate without a world model, making them less practical for applications involving map-based navigation. An alternative approach is to use small robots within a hierarchical team. In such a system, larger robots integrate sensor information from smaller robots and assist with the higher level computations required for localization and path planning [28]. By centralizing some

functions of the team control, however, some of the benefits of redundancy inherent in a multi-robot system are lost.

This chapter addresses the problem of localization of a team of robots using only simple range-measurement sensors. Localization (estimating the robot's position in the environment) is necessary for map-based navigation and exploration, but is a challenging task using only a few range measurement sensors. By sharing the limited sensor measurements among the robots in the team, a cooperative approach can more effectively solve the localization problem.

Localization has two different sub-problems: position tracking from a known starting location, and (the more challenging) global position estimation where no estimate of the initial location is given. This focus of this chapter is the problem of performing global localization of multiple robots, with sensors that are limited in size, power, and number. This chapter presents a method of distributing the problem of global localization across a team of robots, where sets of three robots (referred to as a "triad") work cooperatively. This method extends traditional particle filter algorithms in two ways. First, an alternative state representation for a team of three robots is proposed, effectively reducing the number of variables to estimate. Secondly, the particle filter calculations are distributed across the team, and the best position estimates are shared at each iteration of the algorithm. By sharing their limited sensory data and computational resources, the team is capable of achieving global localization that cannot be accomplished by an individual robot. This development is followed by an investigation of the scalability of the algorithm, using different methods of selecting triads within a large team of robots.

2.2 Literature Review

The problem of localization has been addressed for a wide variety of different problems, such as structured and unstructured environments, varying types

of sensors, and varying constraints on computational resources. This has led to the development of several different methods that can be used, depending on the application requirements. However, virtually all methods have a commonality in their use of a probabilistic approach, which is generally necessary due to the presence of sensor measurement noise. Refer to Thrun et al. [82] for an overview of the development of localization as a probabilistic process.

2.2.1 Localization Methods

Gutmann and Fox give an overview and comparison of several methods of localization [30]. The common goal is to determine the most likely pose of the robot (s_t), based on the available sensor data (d^t). The data d^t includes both sensor measurements of the environment z, and control inputs u, from the first measurements to the current time: $d^t = z_0, u_0, z_1, u_1, ...z_t, u_t$. This can then be formalized as an estimation problem to find the state s_t that maximizes the conditional probability $p(s_t|d^t)$.

As derived in [81], Bayes' rule can be used to compute this posterior distribution of the robot state by integrating over ds_{t-1}, the state space at the previous time step:

$$p(s_t|d^t) = \eta p(z_t|s_t) \int p(s_t|u_{t-1}, s_{t-1}) p(s_{t-1}|d^{t-1}) ds_{t-1} \tag{2.1}$$

where η is a normalizing factor, z_t is the sensor measurements at time t, and u_{t-1} is the commanded motion of the robot from the previous to current time step.

This Bayesian estimator gives an update rule to compute the posterior distribution using only the distribution and the input and sensor measurements from the previous time step; measurements made prior to $t-1$ can be discarded. Unfortunately, the integral term makes calculation of the full posterior distribution computationally expensive, which has led to the development of a variety of methods that approximate the probability distribution more efficiently.

Kalman Filter Methods

One approach is to use a Gaussian approximation, which allows the use of a Kalman filter [37] to estimate the robot pose and a confidence in the pose estimate (the *covariance matrix*). A primary feature of the Kalman filter is that it generates an optimal state estimate for linear systems where the sensor and process noise have known Gaussian characteristics. Smith, Self and Cheeseman first proposed the use of Kalman filter methods to estimate the relative positions of objects measured with noisy sensors [75].

Since the linear system and Gaussian noise assumptions rarely hold for real-world localization applications, variations on the method are generally required. For a nonlinear system model, the extended Kalman filter (EKF) linearizes the system equations about the current estimate to propagate the uncertainty estimate between time steps. The unscented Kalman filter (UKF) more accurately models a nonlinear system by propagating the covariance estimate through a set of sample points (called *sigma points*) through the nonlinear system model [36]. These Kalman filter based methods have been successfully applied to the robot tracking problem, by efficiently fusing multiple uncertain sensor readings to maintain a single estimate of the robot pose as a robot moves from a known initial position [31]. However, because Kalman filter methods maintain only a single hypothesis of the robot position, they are not well suited to the problem of global localization where many possible initial positions must be considered.

Particle Filter Methods

In contrast, particle filter methods, such as Monte Carlo Localization (MCL) proposed by Fox [20], maintain multiple hypotheses of the current robot state. Particle filters approximate the full Bayesian distribution with a set of *particles*, each of which represents one estimate, or guess, of the robot position. A weight is associated with each particle, representing the confidence in that particle's estimate. To initialize the filter, m particles are selected from the configuration space with a uniform distribution, and the weights

are set to $1/m$. At each iteration of the algorithm, the set of particles are updated with the following processes:

- *Sampling*:
  Particles are drawn from the previous set with probability proportional to their weights.

- *State Update*:
  The state of each particle is updated to account for the robot motion (estimated from odometry) for the current time step.

- *Weighting*:
  Weights are computed for each particle, as a function of the difference between the robot sensor measurements z and the predicted measurements $\hat{z}$ based on the estimated position and map data.

This approach creates a higher density of particles near the best estimates of the robot position, making effective use of the computational resources. By maintaining estimates across the entire configuration space, particle filters can perform global localization and address the *kidnapped robot* situation, where a correctly localized robot is picked up and moved to a new position without any information about the movement [80].

2.2.2 Multi-Robot Relative Localization

The ability of a robot to estimate its position relative to other robots in a team is a significant aspect of cooperative multi-robot localization — determining the positions of robots relative to one another is one step of the global localization process presented in this chapter. Relative localization requires a mechanism for determining the range and/or bearing to other robots, as well as an algorithm for combining the measurements into relative position estimates. The algorithm developed in this thesis determines the relative position of three robots using only the distances between them. A review of

prior research suggests some current methods for estimating relative robot poses.

Kato presents a method of identifying other robots and determining their relative positions using omnidirectional vision sensors [39]. By using known shapes and distinctive colors, the bearing to other robots can be determined through image processing. Kato uses the bearing information from two robots to determine the relative location of a third.

Grabowski presents a method using omnidirectional sonar sensors to estimate the distance between each robot pair, and uses trilateration from three stationary robots to determine the relative positions of others [27]. The algorithms presented here differ from Grabowski's, as they allow for independent motion of all robots rather than maintaining three as fixed beacons. The omnidirectional sonar ranging system used by Grabowski, described in greater detail by Navarro [56], is a relatively straightforward mechanism for measuring inter-robot distances.

Rekleitis and Dudek also describe a method for improving the performance of localization by measuring relative position between robots [66]. In their system, one robot in the team is used as a *tracker* that observes the position of the other. The relative position is determined using a camera on the tracker, and a distinctive helical pattern on the roaming robot. This approach solves the problem of accumulated odometry measurement errors for the roaming robot, since the tracker robot can always provide a relative position estimate from its stationary position.

Howard presents a particle filter-based method of cooperatively estimating relative positions of robots in a team [35]. It is assumed that the relative pose of other robots can be measured using fiducials (easily identified markers) affixed to each robot, as well as a camera and laser range finder. A derivation of the posterior updates is given based on inter-robot observations, taking into account the potential for over-estimated certainty that could result from the circular dependencies between estimates (that is, if Robot A estimates its position relative to Robot B, which estimated its position rela-

tive to Robot A, *etc.*).

2.2.3 Multi-Robot Global Localization

Global localization requires the additional ability to estimate the robots absolute position in a known map with no prior knowledge of the robot position.

Roumeliotis presents a Kalman filter-based distributed localization approach they call *collective localization* [69]. The filter performs fusion of sensor data from multiple robots based on the concept of a *group organism*, consisting of all of the sensors available to the group, connected by virtual links between individual robots. The lengths and joint angles of the virtual links represent the relative robot positions, and allow the use of the standard Kalman filter equations to optimally fuse measurements to improve the localization estimate of all robots in the team. When robots are within sight of each other, their relative pose is measured, and their Kalman filter state estimates are combined. The method was demonstrated experimentally to maintain an improved estimate of all robot positions.

Fox [19] describes a Monte Carlo based method for cooperative global localization that synchronizes the beliefs of robots when they detect and recognize one another. The merging of beliefs provides a dramatic improvement in performance over individual localization, assuming that the robots have sensors capable of accurately locating and identifying other robots in the group.

Madhavan applies a distributed extended Kalman filter (EKF) based algorithm for localization and mapping using a team of heterogeneous robots operating in outdoor terrain [48] [49]. The algorithm is demonstrated using a variety of sensors including GPS, scanning lasers and cameras. The use of GPS allows absolute localization and the use of a common reference frame for all robots; however, it also restricts the algorithm to outdoor environments where GPS signals are available.

Rynn developed a solution to the problem of global localization of multiple robots using low cost sensors in structured environments [71]. In that

solution, a CMOS camera is used to estimate the distance between robots, and geometric features (such as straight walls and corners) are identified using infra-red sensors on a rotating base. The method depends on the ability to identify particular geometric features, such as corners, to generate a set of possible positions of each robot in the environment. The relative distances between robots can then be used to determine a unique solution for the position estimation.

Cooperative localization has been investigated by Roumeliotis, Mourikis and Hidaka, with an examination of the optimal formations of robots [33] and optimal use of limited sensors [55] to maximize localization performance of a multi-robot team. Their analytic and simulation results suggest that cooperative localization is optimized when robots create formations of small equilateral triangles.

2.3 Cooperative Localization of Three Robots

The method developed in this chapter first considers the problem of global localization of a group of three robots (a "triad") within a known map of their environment. It is assumed that the robots can sense the environment (through sonar range sensors for example) as well as their own motion (though wheel encoders or inertial sensors for example). For the simulations used in this investigation, each robot is modeled as having two or four fixed-position range sensors, a compass to sense orientation, and a measure of odometry using wheel encoders. In addition, each robot has a mechanism to measure the distance, but not direction, to the other robots (such as an omnidirectional acoustic range sensor similar to that described by Navarro [56] for example). The task is to estimate the global position of all three robots in the given map. This method is then extended to consider cooperative localization in a team of many robots by applying the method to dynamically formed sub-groups of three robots within the team.

2.3.1 Overview

Particle filter localization uses a large number of particles – i.e., state estimates – to approximate the probability distribution of the robot being at any location in the environment as described in Section 2.2.1. The algorithm presented in this chapter uses a similar particle filter approach, but rather than estimate the position of a single robot, it estimates the pose of a triangle with 3 robots at the corners. That is, each particle represents an estimate of the pose of a triad, defined by the variables $\{x_c, y_c, \theta_c\}$ giving the global position of the centroid and the orientation of the triangle. From the estimate of the position and orientation of the centroid of the triangle, and the measured distances between each pair of robots, the estimated position of each robot in the triad can then be computed. The weight of each particle in the particle filter represents the belief in a particular configuration of all three robots.

2.3.2 State Representation

The full configuration space for the three robots is defined by the 9-dimensional space of $\{x_1, y_1, \theta_1, x_2, y_2, \theta_2, x_3, y_3, \theta_3\}$. If the headings $\{\theta_1, \theta_2, \theta_3\}$ are determined solely by a compass on each robot, the remaining variables to estimate are the global position variables $\{x_1, y_1, x_2, y_2, x_3, y_3\}$.

If the distances between the robots are known, the state can be more compactly represented in three variables, $\{x_c, y_c, \theta_c\}$, where the subscript c identifies a reference frame C, the centroid of a triangle with the robots at the corners. θ_c defines the orientation of the reference frame, where the x-axis is aligned with one (arbitrarily selected) median of the triangle. By reducing the dimension of the state space from six to three variables, the computational complexity of the state estimation problem is significantly reduced.

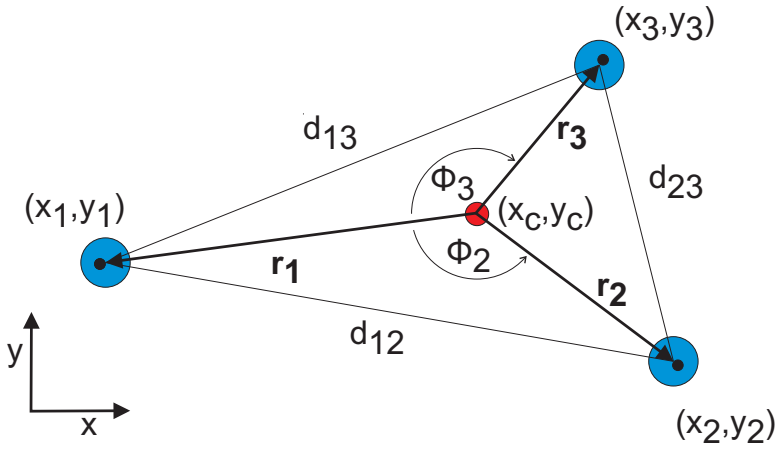


Figure 2.1: Graphical representation of robot positions given the centroid reference frame at (x_c, y_c) and the measured distances between robots $d_{i,j}$.

As derived in [7] for a general triangle, the distances between each robot pair $d_{i,j}$ can be used to calculate the distance r_i from the centroid C to robot i:

$$r_i = \frac{1}{3}\sqrt{2d_{i,j}^2 + 2d_{i,k}^2 - d_{j,k}^2}. \tag{2.2}$$

The x-axis of the centroid frame is aligned with the vector to the first robot. The angles of the vectors to the robots are given by

$$\phi_1 = 0 \tag{2.3}$$

$$\phi_2 = \pm \arccos\left(\frac{r_1^2 + r_2^2 - d_{1,2}^2}{2r_1 r_2}\right) \tag{2.4}$$

$$\phi_3 = \mp \arccos\left(\frac{r_1^2 + r_3^2 - d_{1,3}^2}{2r_1 r_3}\right). \tag{2.5}$$

Note that two symmetrical solutions are possible from the geometry. The selected solution is determined by a random binary variable that is set at initialization of the particle.

The goal of the particle filter algorithm is now to estimate the values of the reduced set of state variables x_c, y_c, θ_c, from which the estimated absolute

position of robot i can be computed:

$$x_i = x_c + r_i \cos(\theta_c + \phi_i), \tag{2.6}$$

$$y_i = y_c + r_i \sin(\theta_c + \phi_i). \tag{2.7}$$

2.3.3 Distributed Algorithm Processes

The randomized nature of the particle filter algorithm makes it suitable for a distributed, parallel implementation on multiple robots. Each robot can apply the algorithm to an independent set of particles, each of which estimates the pose of the centroid of a triad. However, to make effective use of the best estimates found by each robot, the particles with highest weights must be shared amongst the team. At each iteration then, every robot begins with a set of particles including those with the highest weights selected from all three robots.

The sequence of processing and communication involved in the algorithm is shown in Fig. 2.2. The following sections describe the variations to the three steps in the particle filter algorithm required to integrate the sensor readings from all three robots into a cooperative position estimate of all robot positions.

Sampling

At each iteration of the algorithm, a sampling process is required to select a set of particles to propagate forward from the previous iteration. Particles may be selected with probability proportional to their weights. However, a variety of alternative methods can be used to improve the particle selection. As suggested in [20], this method uses a mixture of particles, with a fraction sampled with probability equal to the weights, and a fraction sampled from the best estimates from the most recent sensor measurements. This leads to a denser representation of the belief state in the region of the highest likelihood.

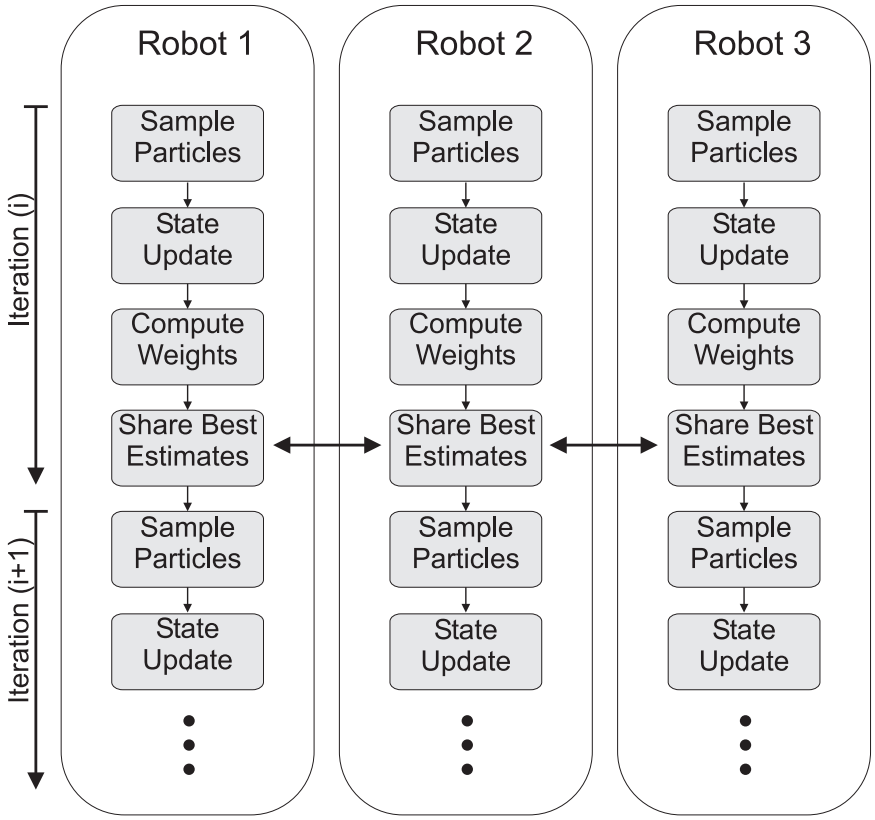


Figure 2.2: The sequence of particle filter processes is executed on each robot, and synchronized by the exchange of the best state estimates after each iteration.

As well, as suggested in [18], a small number of particles are added from a uniform distribution of the state space to aid in global localization if the robots become lost after acquiring a confident estimate.

State Update

At each iteration of the particle filter algorithm, the state of each particle is updated to reflect the motion of the robots since the last iteration of the algorithm, based on odometry measurements. In a single robot application, the position would typically be updated based on the measured odometry and

kinematics of the robot. In this method however, the update must reflect the motion of the centroid reference frame C.

As shown in Fig. 2.3, the motion of C can be computed by first estimating the previous position of the robots, $(x_i, y_i, \theta_i)_{t-1}$, given the previous estimate of the centroid $(x_c, y_c, \theta_c)_{t-1}$ and Equations (2.6 - 2.7). The updated robot positions, $(x_i, y_i, \theta_i)_t$, are estimated by propagating the previous position through h, the kinematic equations of motion for the robot with the measured odometry o_{i_t}:

$$\{x_i, y_i, \theta_i\}_t = h\left(\{x_i, y_i, \theta_i\}_{t-1}, o_{i_t}\right). \tag{2.8}$$

Robot orientations are estimated from compass readings. The updated estimate of C is then computed as the average of the robot coordinates:

$$x_c = \frac{1}{3}\sum_{1}^{3} x_i \tag{2.9}$$

$$y_c = \frac{1}{3}\sum_{1}^{3} y_i. \tag{2.10}$$

The state update also requires an update of the orientation of the frame C, which is computed as the angle of the vector from C to the robot at (x_1, y_1):

$$\theta_c = \arctan\left(\frac{y_1 - y_c}{x_1 - x_c}\right). \tag{2.11}$$

Weighting

For each particle, a weighting is applied representing the degree of belief in the position estimate of the particle.

Using the known map of the environment, the expected sensor readings from each robot are predicted, and these are compared to the actual sensor readings. This comparison is used to assign weights to each particle, inversely proportional to the difference between actual and expected sensor readings, as

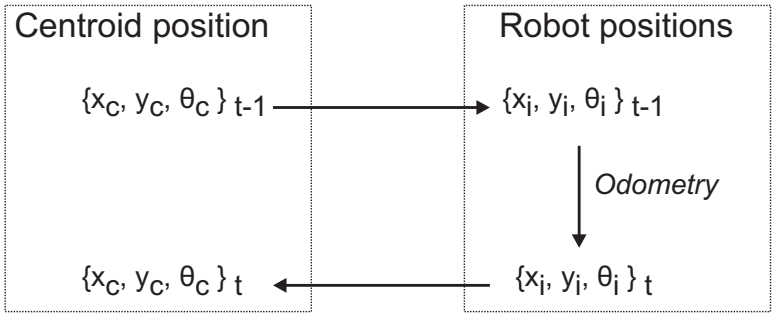


Figure 2.3: The state update process computes the new position of the centroid by first transforming the estimate to the robot coordinates, estimating the robot motion, and transforming back to the centroid coordinates. The upper arrow transformation is defined by Equations (2.6-2.7), and the lower arrow by Equations (2.10-2.11)

.

defined by Equation (2.12). The particles defining the best estimates (those with the highest weights) are then shared between all the robots. Each robot selects a new set of particles using probabilities proportional to the weights, with a fraction selected around the best estimates, and a fraction randomly selected throughout the configuration space.

This weighting W_p is determined as a function of the error between the predicted IR sensor readings that would be measured from the estimated robot positions and the actual measurements from all three robots:

$$W_p = \frac{1}{\sqrt{\sum_{r=1}^{3}\sum_{s=1}^{2}\left(z_{r,s} - \hat{z}_{r,s}\right)^2}} \tag{2.12}$$

where $z_{r,s}$ and $\hat{z}_{r,s}$ are the measured and predicted values of sensor s on robot r respectively. The weights are then normalized such that $\sum W_p = 1$.

2.3.4 Results

The algorithm was implemented and evaluated in a simulation of a group of 3 robots, each with two fixed-direction infra-red (IR) range sensors, odometry

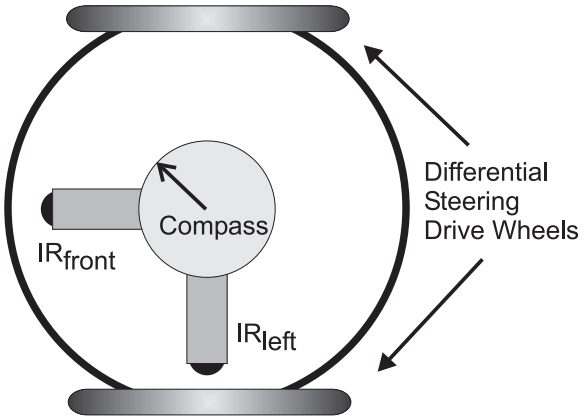


Figure 2.4: The simulated robot configuration for the single-triad simulations of Section 2.3.4: The robot is driven with differential steering, and has a compass and two IR sensors oriented at 90° to sense the environment.

sensors, and a compass (as indicated in Figure 2.4), operating in a confined area including obstacles and walls. Obstacles and walls are detected by the IR sensors, but are considered transparent for the inter-robot range measurements; this simplification was removed for further simulations described in Section 2.4. Each robot begins in a random location in the known map. They then create a randomized trajectory by driving forward until they reach an obstacle, after which they turn through a randomly selected rotation. The models of the range sensors, compass, and odometry measurements include injection of Gaussian noise based on typical sensor performance.

During the simulation, each robot performed two localization algorithms. The first was an implementation of the particle filter running in isolation on each robot, using only data from its own sensors. This was used as a baseline for comparison of results. The second algorithm was the distributed method described in this thesis, using the combined sensor readings from all robots and sharing the best estimates.

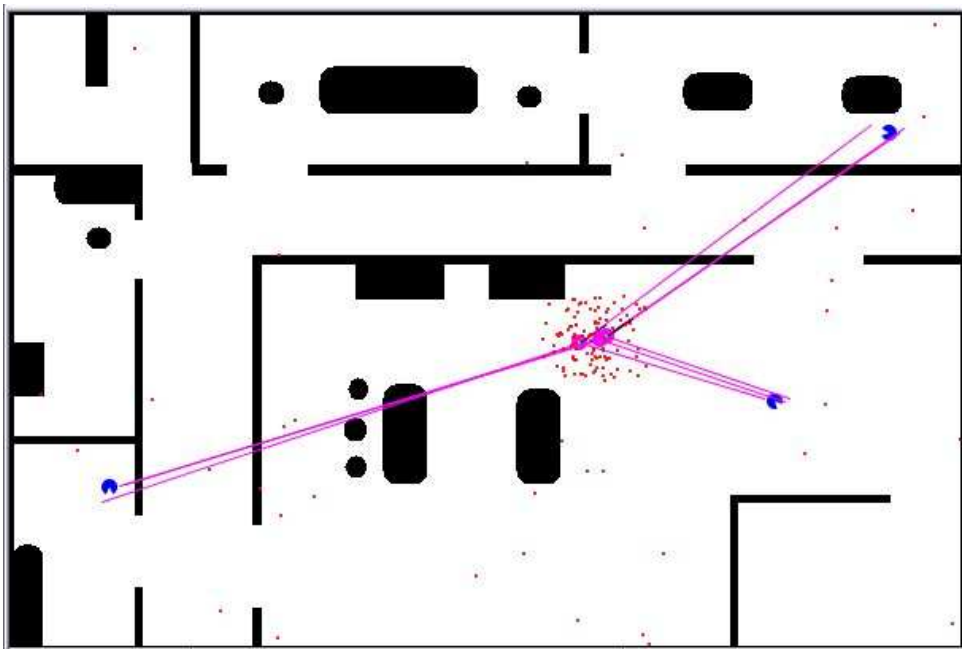


Figure 2.5: The simulation environment, a cluttered $6m \times 4m$ area, showing the system state after successful localization.

Simulation Environment

The simulation environment is shown in Fig. 2.5. The small dots indicate the position of particles maintained in the particle filters. The lines radiating from the central cluster represent vectors from the best estimates of the centroid position to the estimated positions of the three robots. The three circles near the end of the lines indicate the actual positions of the robots. Note that a higher density of particles is maintained around the estimated centroid after completing the localization.

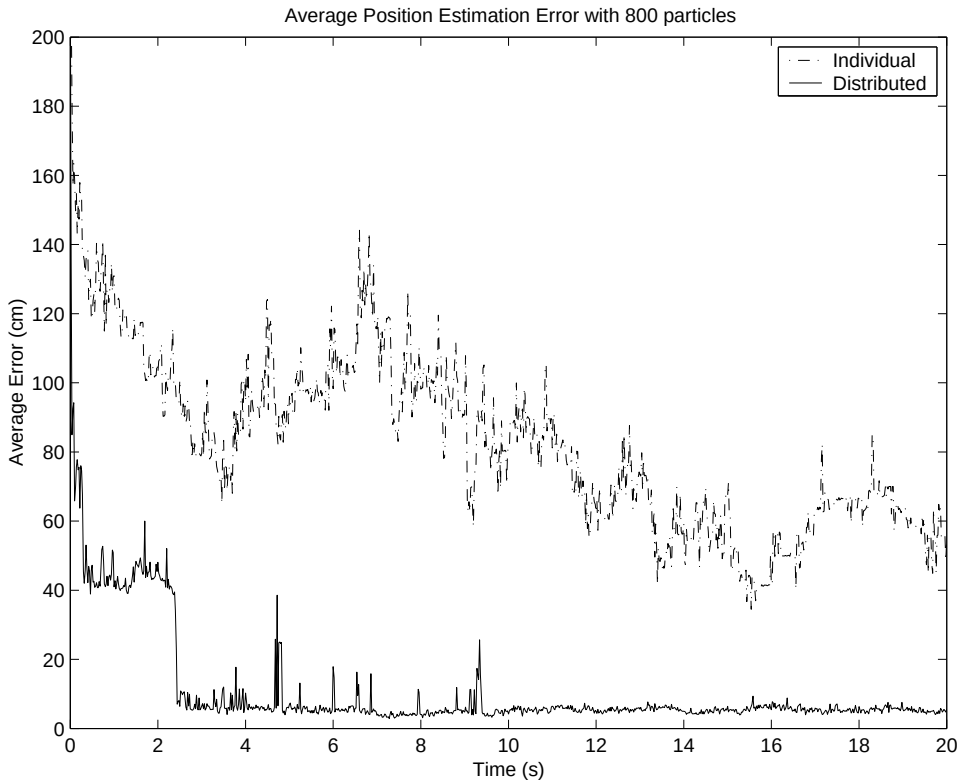


Figure 2.6: Comparison of position error over time for the individual particle filters (dotted line) and the cooperative distributed particle filters (solid line). The results are averaged over 10 consecutive simulations of 20 seconds each.

Position Estimation Performance

In Fig. 2.6, the average position estimation error is plotted over time for a set of 10 simulations using 800 particles for each robot. On average, after the first 3 seconds of the simulation the distributed localization has converged to the correct estimation of the centroid and robot positions (as shown for example in Fig. 2.5). This is in contrast to the performance of the individual particle filters operating in isolation, also with 800 particles in each filter, which on average do not converge to the correct solutions for all three robots within the 20 second simulation time.

As shown in Fig. 2.7, the error between the predicted and actual sensor readings are consistently smaller using the individual particle filters. Operating in isolation, the individual position estimates can lead to many solutions that give sensor readings similar to those from the actual robot position. In contrast, in the distributed method the larger discrepancies between predicted and actual sensor readings reflect the increased constraints on the possible position estimates, imposed by the trilateration calculations. Only those estimates that satisfy the inter-robot distance measurements are considered, leading to less freedom to minimize the sensor prediction error, and a better overall estimate of position.

Figure 2.8 shows the performance of the algorithm over a range of sizes for the particle filter, simulating 50 iterations per second on each robot. The vertical scale indicates simulation time, corresponding to the number of iterations of the algorithm with a fixed time step. While the number of iterations required to successfully localize the team decreases as the number of particles increases, the computational cost of each iteration of the algorithm is proportional to the number of particles. This is particularly significant for an algorithm intended for small robots with limited processing capabilities, as the frequency of the algorithm execution will be constrained by the number of particles used. The ideal size of the particle filter for a particular application will be determined based on the computational resources available, the size of the environment, and the required performance of the localization system. The size of the particle filter can be tuned by performing an evaluation of localization performance in the target environment, as in the example shown in Figure 2.9, and selecting the minimum number of particles corresponding to the required convergence rate. For this example, a filter of 200 particles is a suitable choice, averaging approximately 20 time steps and 25000 particle evaluations to converge to a solution. Using less than 100

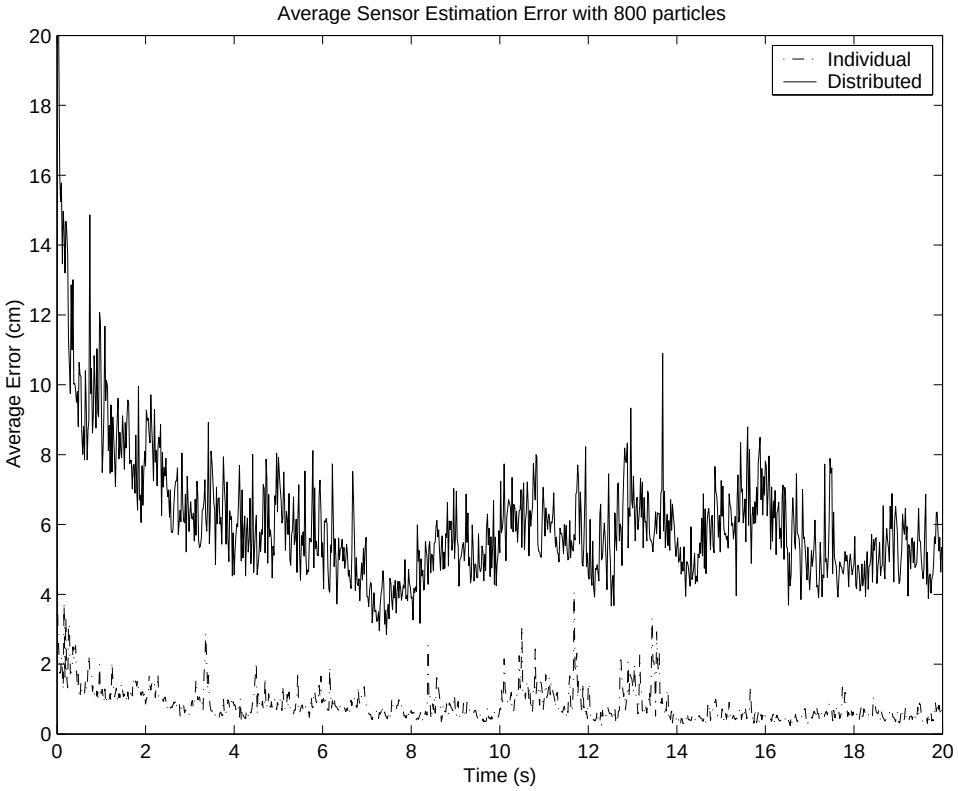


Figure 2.7: Comparison of sensor prediction error over time for the individual particle filters (dotted line) and the cooperative distributed particle filters (solid line) for one representative simulation run.

particles greatly increases the time to convergence, while using more than 400 greatly increases the number of particle evaluations required.

2.4 Cooperative Localization of Many Robots

The triad-based localization method presented above can be extended to improve localization in a large groups of robots, by dynamically selecting sets

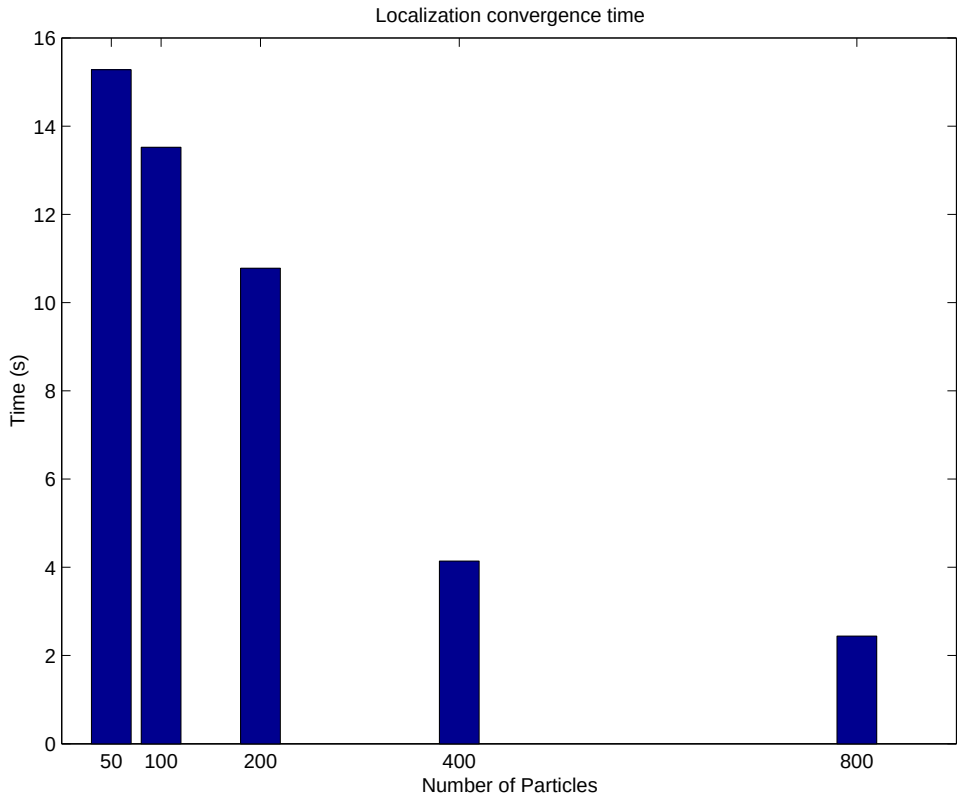


Figure 2.8: Comparison of the rates of convergence for varying number of particles.

of three robots to cooperatively localize within the larger group. Each robot can then generate an estimate of the positions of itself and two neighbors, using the trilateration method described above.

In this case, each particle filter estimates the position of the centroid of a triangle of robots, but the particular three robots (and the estimate of their centroid) may be different for each particle filter. In the *sharing best estimate* process, therefore, the position of the centroid cannot be shared, as it is only relevant to one triad. Instead, the estimated absolute positions of individual robots must be computed and shared. The absolute positions can then be transformed back to the centroid representation required for

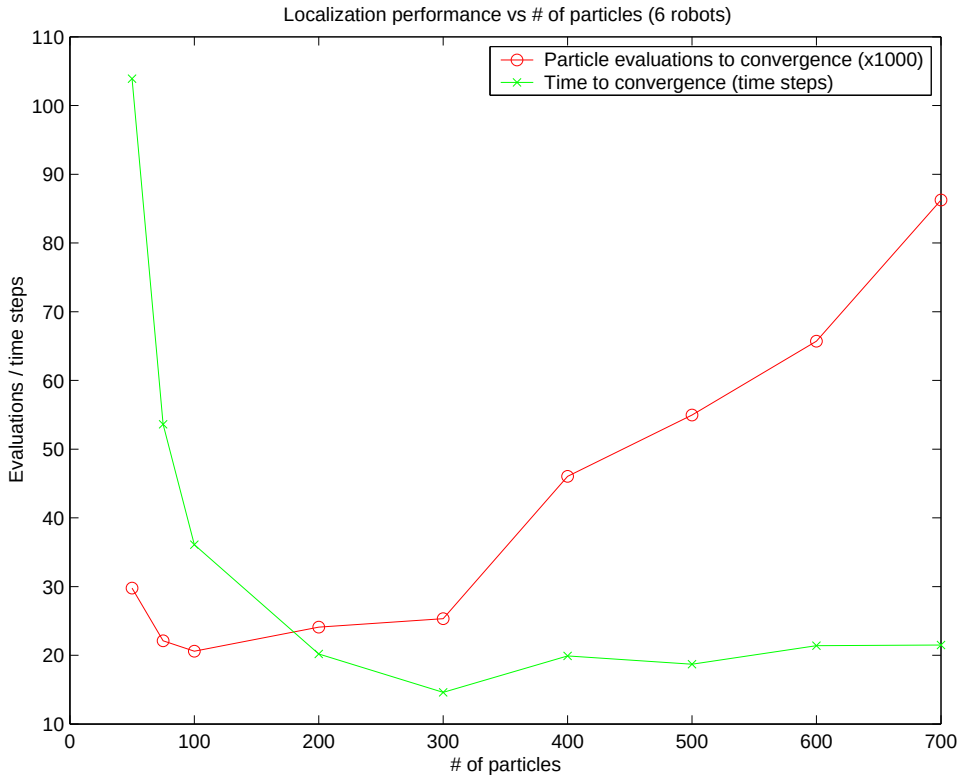


Figure 2.9: Comparison of the computational cost (measured in the number of particle evaluations before convergence) and the time to convergence (measured in the number of time steps) for six robots using a varying number of particles.

each local particle filter. Simulations are used to demonstrate the process, and investigate the effect of different approaches to selecting triads of robots with the team.

2.4.1 Localization Process for Many Robots

Applying the three-robot method to a team of many robots, each robot executes the following process at each time step:

1. *Prediction:* The best position estimates from the previous iteration are

updated to reflect the estimated motion of the robots using odometry.

2. *Range estimation:* Distances to other robots are measured from range sensors.

3. *Data exchange:* The sensor measurements and best position estimates are broadcast to all other robots within range.

4. *Triad selection:* Two other robots within sight are selected for the cooperative localization process. If two other robots are not within sight, the robot relies on odometry to update its own position estimate.

5. *Particle generation and Weighting:* A set of particles is generated based on the estimated poses received from other robots in the *data exchange* step, and the ranges between robots determined in the *range estimation* step. A weight for each particle is computed by comparing the sensor measurements of each robot to those predicted by the particle's estimated poses.

Prediction

In the prediction step, each particle of the filter is updated based on the current motion of the robot. The current motor control outputs (or odometry sensor measurements if available) are applied to a model of the robot motion, to determine the change in position and orientation since the previous time step. The pose estimate represented by each particle is then updated based on the estimated motion. In simulation, the motion is estimated based on commanded wheel rotations of a differential-drive robot, with Gaussian noise added to the simulated motion.

Range Estimation

Each robot transmits a beacon, such as a time-synchronized sonar pulse, and receives the corresponding beacons from other robots in the same area.

Based on a time-of-flight calculation, the distance to other robots in the area can be determined. In a physical implementation, range measurements may be acquired using an integrated system such as the *Cricket* location system developed by Priyantha et al. [65] [64] and available from Crossbow Technologies. In simulation, the range measurements are generated only if an unobstructed line-of-sight is available between robots, using the simulated distance between robots and the addition of Gaussian noise.

Data exchange

Each robot maintains a best estimate of the current position of itself and the other two robots in the triad, based on the particle with the highest weight in the previous iteration. These best position estimates, and the confidence of the estimate (the particle weight) is broadcast to all other robots, and the current best estimates from all other robots within range are received and stored. Sensor measurements, range measurements, and planned velocities are also broadcast to all other robots, for motion prediction in the next timestep and evaluation of the particle weights. In the simulation, sensor measurements of the environment include only four range values, representing readings from four sonar or IR sensors on each robot.

Triad selection

To combine the position estimates and range measurements using the trilateration process discussed above, each robot selects two other robots to use in the cooperative localization process. The selection criteria can take several considerations into account:

- *Confidence:* When selecting a triad, it may be beneficial to select the two other robots with the greatest confidence in their own estimates, effectively performing a relative localization to two well-localized beacons. However, this greedy approach may not be optimal for the team; if three robots are well-localized with a high confidence in their own

estimates, it may be more globally beneficial for them to form triads with other poorly localized robots rather than with each other, in order to improve the overall estimation of the team.

- *Geometry:* The impact of measurement errors depends significantly on the shape of the triangle formed by the triad. The positions and angles computed in Equations (2.2 - 2.5) are more sensitive to small changes in the range measurements for an obtuse triangle than an equilateral triangle. The variance in sensitivity can be seen by considering the derivative of Equation (2.2), with respect to $d_{j,k}$, for each vector r_i. Setting all of the derivatives to zero to minimize the variance requires equating $d_{1,2} = d_{1,3} = d_{2,3}$, corresponding to an equilateral triangle. Triads may therefore be selected by minimizing the difference in ranges between robots, forming approximately equilateral triangles when possible.

- *Persistence:* Since robots share the estimates of the positions of all robots in their triad at each iteration, there may be benefit to maintaining persistent triads. That is, once a triad is selected, change the selection only if necessary (if any pair of robots in the triad cannot make inter-robot range measurements for example). Maintaining a persistent triad allows the particle filter to converge to a solution for a set of three robots. A continually varying triad selection will require the filter to converge on a localization solution for a larger number of robots (and a corresponding larger search space).

The effects of these factors on the performance of the overall team localization are not obvious, and are investigated in the simulation results presented in Section 2.4.2.

Particle generation and Weighting

Unlike a conventional particle filter, the set of particles cannot be simply propagated from the previous iteration. Since the particle represents the

centroid of a triangle of three robots, and the particular three robots may change dynamically between iterations in the *triad selection* step, a new set of particles must be generated for the current set of robots in the triad.

To generate a suitable set of particles, an initial set of seeds are generated using the current position estimates of each of the three robots in the triad. Each seed particle represents the pose of a triad, and is determined using the estimated positions of 2 robots, and the range measurements to the third. The position estimate of the third robot is estimated using the intersection of two circles, centered at the estimates of the first 2 robots, and with radii equal to the range measurements from the first 2 robots to the third. Using this method, given two well-localized robots, at least one seed will correctly estimate the position of the third robot. Weights are then computed for each of the seeds, by comparing the measured sensor values from each robot to those predicted by the particle's estimated poses.

A complete set of particles is generated using a weighted random sampling from these seeds, with Gaussian noise added to each estimate. A weight for each particle is computed by comparing the sensor measurements of each robot to those predicted by the particle's estimated poses. The best position estimates of the three robots in the triad (based on the particle with the highest weight) are stored for the following iteration.

2.4.2 Simulation Results

To investigate the performance of the trilateration-based localization in a large team of robots, a simulation was created in an artificial environment, shown in Figure 2.10. In this environment, the black circles represent obstacles in the map that can be detected by the range sensors; these are also included in the robot's map of the environment, and are used for the prediction of range measurements in the localization process. The obstacles also block inter-robot range measurements, limiting the combinations of robots that can form cooperative triads. The simulated positions of five robots are indicated by pie shapes, and the straight lines indicate inter-robot range

measurements forming two triads. The localization process is defined to have converged to a correct solution for a robot when the error between the estimated and simulated position of the robot is less than twice the radius of the robot. The entire team of robots is considered localized when the process has converged to a correct solution for all robots in the team simultaneously.

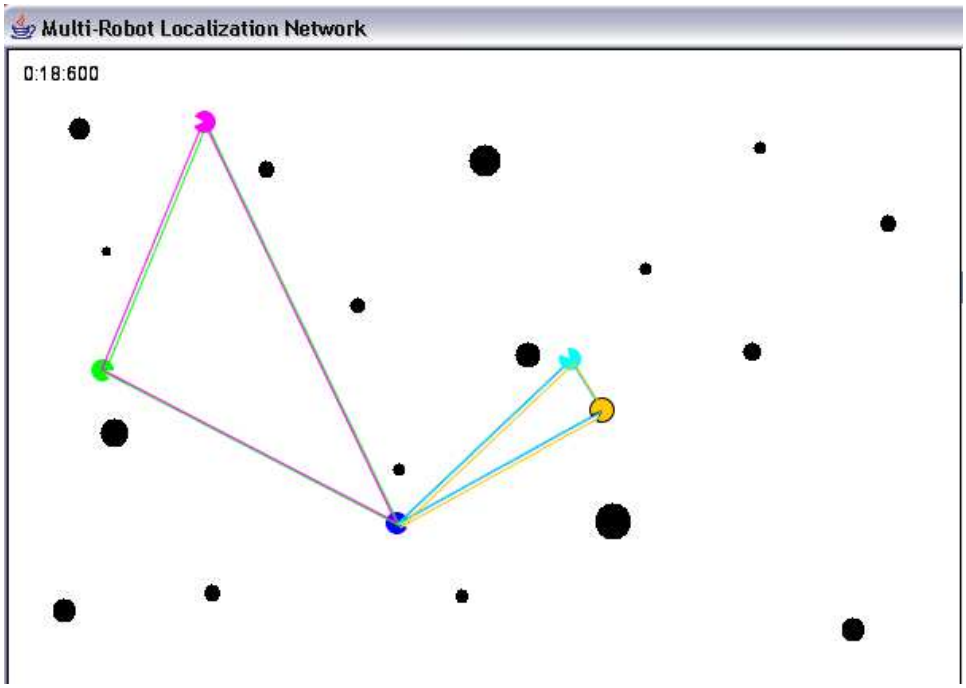


Figure 2.10: The multi-robot team localization simulation environment.

To investigate the performance of the localization system as the total number of robots increases, simulations were run varying the number of robots between 3 and 18. The time required for the system to localize to the actual positions was measured, and averaged over 10 simulations with random initial positions of all robots. To investigate the effect of different triad selection criteria, results were generated using different methods. In

each case, a triad is only formed between 3 robots that all have an obstacle-free line-of-sight between each another.

Baseline: Static Triad Selection

As a baseline for comparison, triads are statically defined between sets of 3 robots at initialization of the simulation. The triad selection remains fixed for the duration of the simulation. This is similar to the simulation in Section 2.3.4, but with multiple independent teams of 3 robots.

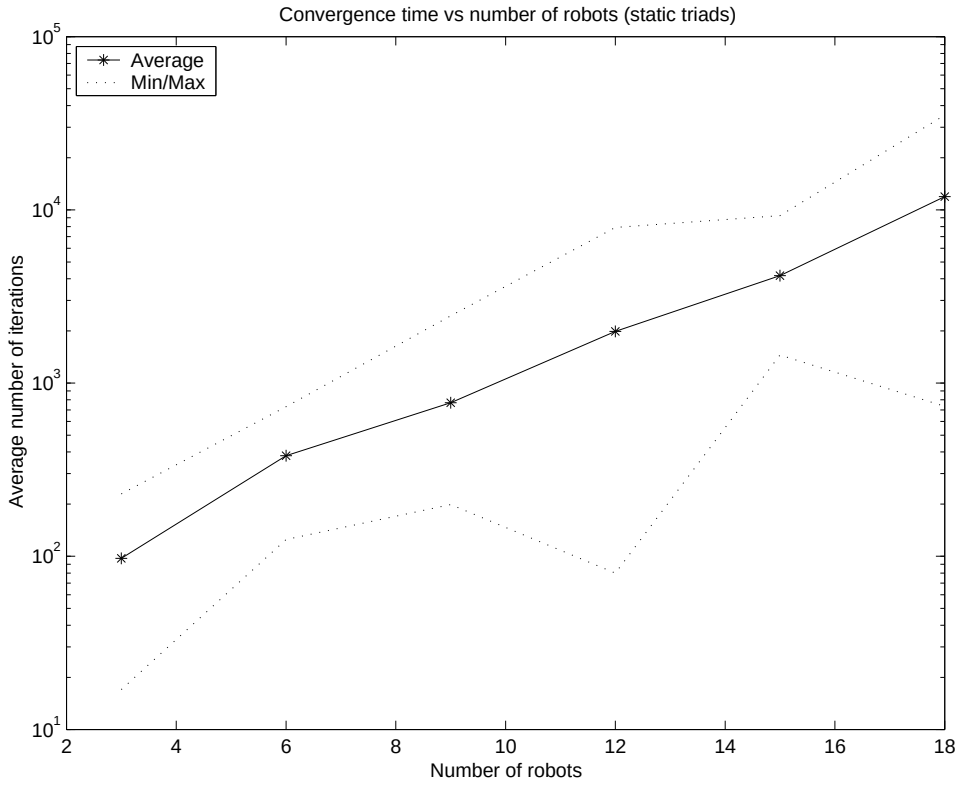


Figure 2.11: Baseline convergence rates for static triad assignment.

Figure 2.11 shows that the number of iterations required for convergence increases exponentially with the number of robots in the team, using a static allocation of triads (note the logarithmic scale on the y-axis). This trend is due to the decreasing likelihood of all teams simultaneously achieving line-of-sight for inter-robot range measurements as the number of sub-teams increases. With statically allocated sub-teams, the robots do not dynamically create triads with other visible team-mates. Instead, when visibility between the assigned sub-teams is lost, the robots rely on odometry to predict position, and the individual estimation error grows without bound until the obstacles are passed and visibility is regained. Complete localization of the team only occurs when all sub-teams cooperatively localize by achieving visibility within the pre-assigned triad; this becomes increasingly improbable as the number of robots increases.

Best-estimate Dynamic Triad Selection

Dynamic selection of triads based on line-of-sight visibility at each time-step allows a robot to use the trilateration localization when it has visibility of any other two robots that also have visibility of each other. As discussed in Section 2.4.1, there are several possible approaches for selecting the two other robots to form the triad (if more than one potential triad is available). One possibility is selecting the two visible robots with the highest confidence in their current estimate. This corresponds to the minimum difference between predicted and measured range sensor readings in the prior iteration. The goal of this greedy approach is to take advantage of any correct localization solutions found by other robots; using the trilateration method, a robot can determine its absolute position based on its range measures to two other well-localized robots.

Figure 2.12 shows the convergence time observed when selecting triads based on the robots with the highest confidence in their estimates, compared

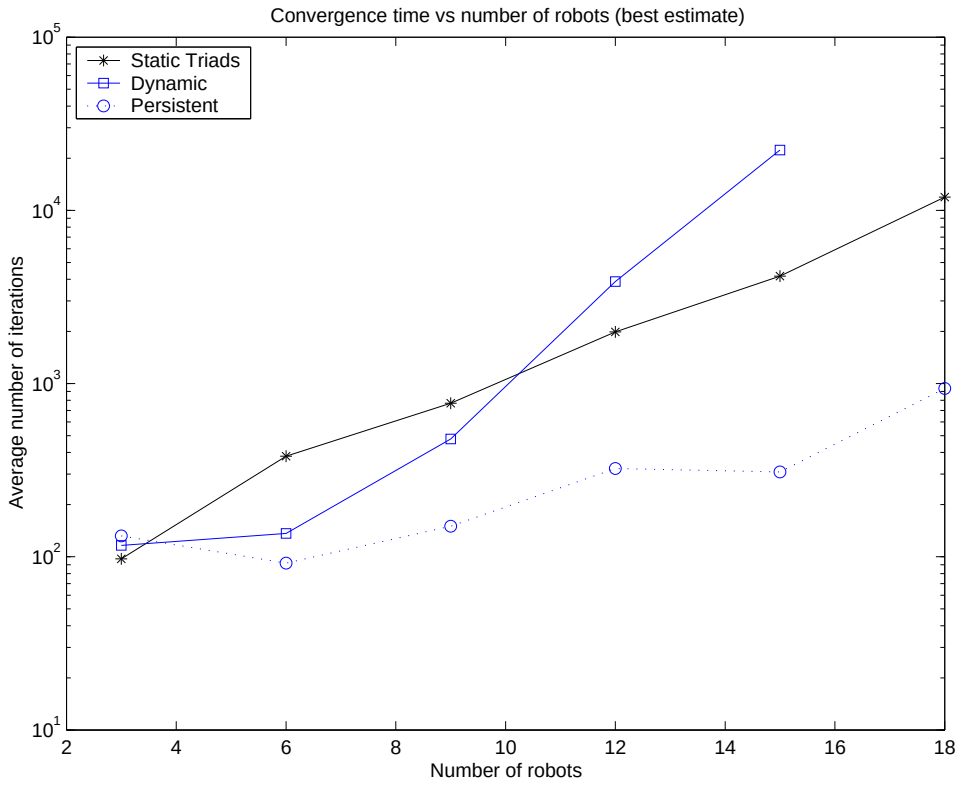


Figure 2.12: Convergence rates for best-estimate dynamic triad selection.

to the baseline allocation. The solid line connecting squares (*Dynamic*) is based on a re-selection of the triad at every iteration using the highest confidence criteria. Between 3 and 6 robots, the performance improves compared to the static allocation. For more than 6 robots however, like the static triad allocation performance, the convergence time increases exponentially. This trend can be attributed to the fact that each robot's particle filter is now effectively attempting to estimate the position of N robots, where N is the average number of visible robots that may be selected into the triad. The dynamic selection converges only when all robots that may be included in the triad are actually correctly localized; this requires estimation of $2 \times N$ coordinates, instead of the 6 variables involved for a static triad. The increase in

the dimensionality of the search space is reflected in the exponential increase in convergence time.

The dotted line connecting circles (*Persistent*) is based on a re-selection of the triad using the same criteria, but only when the current triad fails (visibility between a pair of robots is lost due to an obstacle). By maintaining existing triads when possible, the filter can converge to a position estimation for 3 robots, rather than estimating the position of whatever visible robots have the highest confidence at each iteration. This corresponds to a substantial reduction in the convergence time compared to the fully dynamic best-estimate selection approach.

In summary, the greedy approach of forming triads with well-localized robots fails, because in the initial state no robots are correctly localized. Instead of converging more quickly, the random changing of triads makes the process more difficult; this is overcome to some degree by maintaining existing triad selections when possible.

Biggest-Area Triad Selection

An alternative approach to selecting triads is based on geometric considerations. As discussed in Section 2.4.1, by selecting robots that form a triangle that is close to equilateral, rather than obtuse, the impact of measurement errors on position estimation can be minimized. In addition, by forming larger triangles, the relative significance of errors in range measurements is reduced. In the simulation, range measurement noise is modeled as a constant Gaussian addition to actual distances, assuming a model based on typical sonar sensor measurement noise.

Figure 2.13 shows the convergence time when selecting triads based on the visible robots that will form a triangle with the largest area. The area can be determined using Heron's formula in Equation (2.13) for the area of

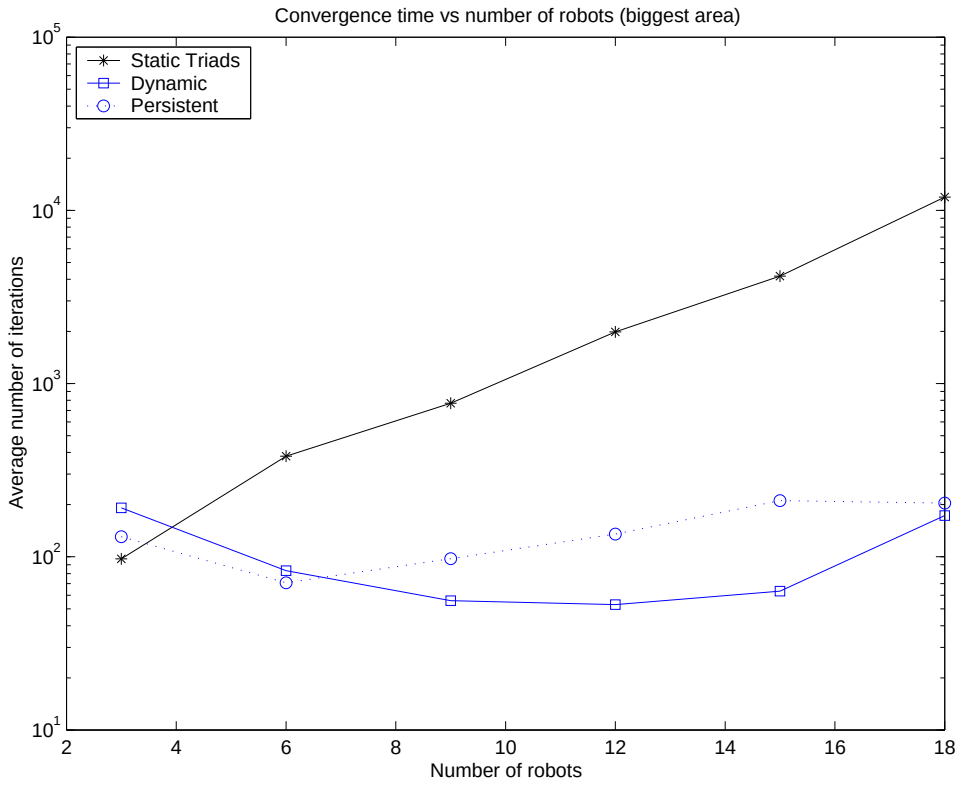


Figure 2.13: Convergence rates for biggest-triangle dynamic triad selection.

a triangle based on the range measurements a, b, and c between robots:

$$A = \sqrt{s(s-a)(s-b)(s-c)} \tag{2.13}$$

where $s = \frac{1}{2}(a+b+c)$.

The dashed line connecting squares in Figure 2.13 (the *Dyanamic* selection, where new triads are selected at every iteration) shows a significant improvement over the best-estimate triad selection approaches shown in Figure 2.12. By dynamically forming triads that minimize the effect of measurement errors in the position calculations, the convergence time *decreases* significantly as the number of robots increases from 3 to 12. As the number of robots increases further, the convergence time begins to increase again; this is likely due to the high density of robots in the environment. With a large

number of robots, it becomes increasingly difficult to form triads covering a large area, since the robots themselves become line-of-sight obstacles between other robots. However, even with 18 robots in the small environment, the average convergence time is approximately the same as for 3 robots, demonstrating effective scalability using the biggest-area triad selection approach.

The dotted line connecting circles in Figure 2.13 (the *Persistent* selection, where new triads are only selected when the current triad is broken) demonstrates that the benefit of the geometric structure is of greater benefit than maintaining the current triad selection. In comparison to the best-estimate approach, the biggest-area selection is inherently less dynamic. The robots forming the largest triangle are much less likely to change from iteration to iteration, compared to the robots with the highest confidence in their estimates.

To illustrate the convergence trends over time, Figure 2.14 shows the convergence over time for one representative simulation using both the static triad allocation and the biggest-triangle dynamic selection approach. In the upper plot, the number of robots localized at each iteration rises and falls many times over a period of over 500 iterations. This is due to the divergence of individual estimates as robots depend on odometry when they do not have visibility with their pre-assigned triad. In contrast, using the biggest-area triangle approach, the number of robots correctly localized increases relatively steadily over a relatively short time (22 iterations). Once localized, individuals effectively maintain correct estimates by cooperating with other robots that are in suitable positions to minimize estimation error.

2.5 Summary

This chapter presented the development of a cooperative method of global localization for mobile robots. The algorithm for a triad of robots uses trilateration to determine the possible relative positions of three robots, forming a

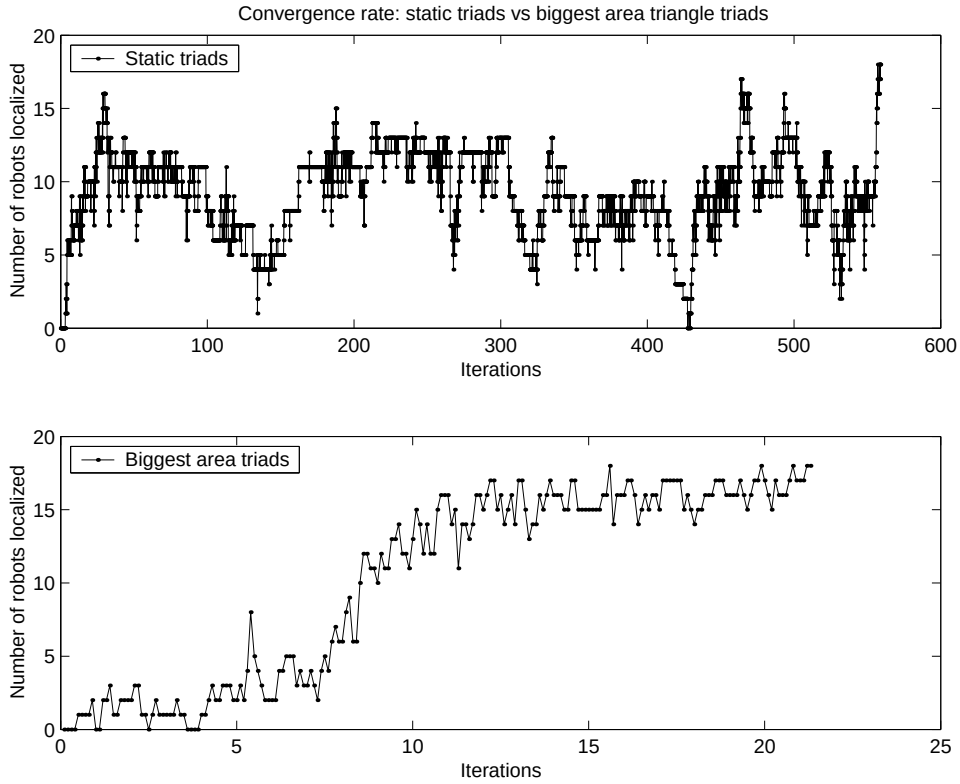


Figure 2.14: Convergence over time for static vs biggest-area triad selection.

triangle with the relative position of one robot at each corner. A particle filter is then used to estimate the centroid and orientation of the triangle, using range sensor measurements of the environment from all robots to evaluate the particle weights. Simulation results demonstrate the performance of the algorithm with only simple range sensors on each robot, while the number of particles used is varied over a range of relatively small values. Three robots are able to localize themselves in an environment where isolated particle filters on each robot failed to converge.

Cooperative localization of a team of many robots was investigated by extending the triad-based algorithm to sub-teams of three robots. The challenging aspect of this extension is the selection of triads for the sub-teams;

the overall performance, measured as the time required to converge to a correct position estimate for all robots in the team, is highly dependent on the triad selection criteria used. Statically assigning robots to fixed triads, or dynamically selecting triads based on confidence estimates leads to an exponential increase in the convergence time as the size of the team increases. However, using a geometric selection criteria of maximizing the area enclosed by the robots in the triad leads to a decrease in the convergence time as the size of the team increases up to 18 robots in the simulation environment. By dynamically assigning triads based on the current visibility and network connectivity between robots, the system is also robust to individual robot or network failures.

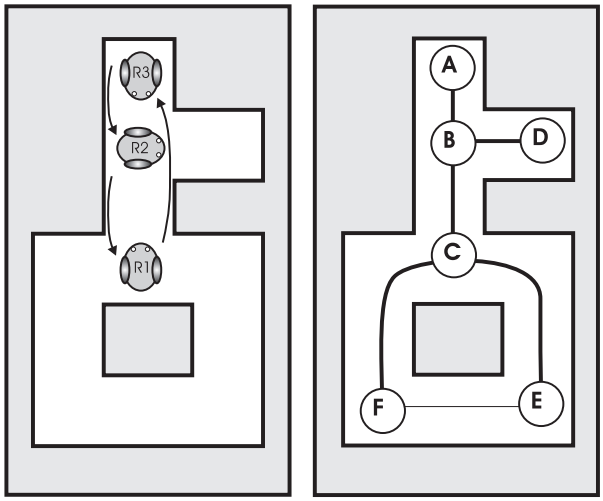
Chapter 3

Multi-Robot Motion Planning

3.1 Introduction

The use of multiple mobile robots in a common environment is valuable for the automation of many operations, such as underground mining and warehouse management. In such applications, multiple vehicles are required to drive autonomously between different locations, preferably taking the shortest possible route while avoiding collisions with static objects and other vehicles. This requires first the localization of all vehicles, as discussed in Chapter 2, followed by path planning, the selection of a path from the current position to the goal location. This chapter presents an algorithm for efficiently determining collision-free paths for many vehicles in environments composed of tunnels or corridors, as may be found in these applications. The problem addressed by this research is demonstrated by the multi-robot planning task pictured in Figure 3.1(a).

In this scenario, the environment is constructed of corridors or tunnels that are wide enough for only a single robot to travel, and we assume differential drive robots that can rotate in place. The objective in this example is to shift the position of each robot, such that robot R_1 moves to the initial position of R_3, R_3 to the position of R_2, and R_2 to the position of R_1. Our goal is to find an algorithm that is scalable to a large number of robots



(a) A planning problem (b) Graph-based map

Figure 3.1: A multi-robot planning problem requiring coordination of 3 robots, and a graph-based representation of the environment.

(> 100) densely situated in a large environment, and can solve problems that specifically require coordinated planning, such as that shown in Figure 3.1(a).

3.2 Literature Review

Many methods have been proposed for planning the motion of one or more robots; refer to Latombe [42] and LaValle [43] for detailed reviews. The methods are differentiated by the map representations they use (and how they generate their maps), and by the search method used to find a connected obstacle-free path from the robot to its goal. The methods can be evaluated in terms of completeness (whether they are guaranteed to find a solution if one exists), complexity, and optimality.

3.2.1 Map Representations

As introduced in Section 1.2, map representation is a significant factor in the efficiency of motion planning algorithms. Occupancy grids are a common map representation for robot navigation, and are easily derived from range sensor measurements [15]. Optimal and resolution-complete algorithms (guaranteed to find a solution subject to the spatial resolution of the occupancy grid) have been presented. This approach creates a graph representation with a node for every unoccupied cell, and edges between all adjacent unoccupied cells. The A* search algorithm [32] is then used to find the shortest path to the node of the goal cell. Improvements have been made on this common method, such as D*, which efficiency updates paths when the map changes dynamically [77].

For motion planning problems, graph representations such as topological graphs (or *roadmaps*) are often more efficient than high resolution grid maps. By abstracting the structure of the environment to a set of open spaces (nodes) connected by corridors or tunnels (edges), a graph representation reduces the number of possible states of the system, and therefore reduces the complexity of the search for collision-free paths.

Roadmaps define a set of admissible collision-free paths (graph edges) connecting points (nodes) within the environment. The selection of these paths depends on the environment, to ensure that stationary obstacles are avoided. The roadmap also depends on the kinematics of the robot, since only admissible paths (that the robots can actually follow) should be included.

Several methods of generating roadmaps have been developed for different applications. *Deterministic roadmaps* can be generated using methods such as Voronoi graphs [9] and visibility graphs. These combinatorial methods require a polygonal representation of obstacles in the map, but give complete representations of the connectivity of the environments. This full representation enables complete and optimal path planning, but can be difficult to generate and impractical for higher dimensional configuration spaces.

In contrast, *probabilistic roadmaps* (PRMs) are generated by randomly

selecting *milestones*, which are points in the robot configuration space, and connecting pairs of milestones that have obstacle-free paths between them [40]. Due to their random nature, PRMs do not include all obstacle-free paths through the configuration space, so do not result in complete or optimal planning solutions. However, they are most effective in very large configuration spaces for which deterministic roadmaps would be impractical.

The planning algorithm presented in this chapter requires such a roadmap, in the form of a topological map, but is independent of the particular method used to generate it.

3.2.2 Multi-Robot Planning

Motion planning algorithms for multiple robots are typically based on those developed for individual robots. For example, the A* algorithm can be applied directly by creating a configuration space including the coordinates of several robots, and probabilistic roadmaps can be generated for multiple robots simultaneously [11]. However, two particular challenges need to be addressed in motion planning for multiple robots. First, the dimensionality of the configuration space increases with the number of robots in the system; for a complete search algorithm, the complexity increases exponentially with the number of robots. The second consideration is that the robots themselves become dynamic obstacles in the environment. This requires a multi-robot planner to consider the motion of all robots as *trajectories* (positions as functions of time), rather than simply time-independent paths that lead to the goal. The planner must generate mutually collision-free trajectories that drive all robots through the configuration space-time to their respective goals.

Most multi-robot planning algorithms that achieve this fall into one of two categories, *coupled* or *decoupled*. *Coupled* algorithms, such as [78], plan the trajectories of all robots in the environment concurrently. By combining the states (poses) of the individual robots together into a system state representation, a sequence of state transitions can be found that will move

all robots to their respective goals. Using complete search methods, such as A* [32], coupled algorithms can achieve completeness and optimality (the shortest path solution), and can solve the problem shown in Figure 3.1(a). Coupled algorithms depend on a centralized architecture, where all of the state information is available to a single processor. Their limitation is in searching the large configuration space that grows in dimension as each additional robot is added to the environment. A direct application of the A* search would guarantee a resolution-complete solution. However, since the size of the configuration space (the number of possible states of the system) grows exponentially with the number of robots ($O(k^r)$ for r robots), the computational complexity of the A* search also increases exponentially and quickly becomes intractable. Hopcroft et al. have shown the general motion planning problem for multiple moving objects to be PSPACE-hard [34]. One approach to reducing the size of the search space is to create probabilistic roadmaps (PRMs) through the environment; this method was shown in [78] to be probabilistically complete and demonstrated in simulation for up to 5 robots. Another approach is to decompose a large map into subgraphs, and plan paths between subgraph segments before coordinating motion within each subgraph [70].

Decoupled methods plan for the motion of individual robots, rather than planning the motion of all robots simultaneously. One approach is to decouple path planning from mutual collision avoidance, by first finding obstacle-free paths, then adjusting velocities of individual robots to avoid collisions [38] [63] [29]. Alternatively, a *coordination-diagram* [60] approach can be used to combine independently generated paths of many robots while avoiding collisions [73].

Decoupled methods may use a decentralized architecture, allowing independent planning based methods such as maze-searching [47] or potential fields [3] [21], or they may use a centralized architecture planning for all robots with a single processor. Centralized decoupled planners typically determine individual trajectories sequentially and combine the plans of all

robots to avoid collisions. Plans may be combined by iteratively adding new plans as obstacles into the configuration space-time [16]; however, this inherently involves assigning priorities to robots to determine the order in which plans are added, which affects the quality of the resulting plan. This can be addressed by considering all different combinations of priorities (for up to 3 robots, demonstrated in [2]), or running an optimization process on the priority assignment [4]. In a more dynamic paradigm, the plans of individual robots can be merged into the global coordination plan as new goals are assigned [1].

By planning the motion of robots sequentially, decoupled methods have lower complexity and greater scalability than a coupled planner; however, this comes at the cost of completeness and optimality. The problem in Figure 3.1(a) for example cannot be solved by a sequential planner. By selecting the optimal plan for any robot independently, an obstacle is created in the space-time map that cannot be avoided by the other two robots.

This chapter presents an alternative *multi-phase* planning method that can solve these coordinated planning problems, and is scalable to a large number of robots in a large environment. For the tunnel and corridor environments considered here the segments are only one lane wide, reducing the complexity of a suitable topological map generation process compared to the general case. A multi-phase planning approach then takes advantage of the properties of the graph and spanning tree to create and maintain obstacle-free paths while robots move to their respective goals.

3.3 Multi-Phase Planning Algorithm

The multi-phase planning algorithm depends on a topological graph representation of the environment, and the selection of a spanning tree for the graph. These structures must be generated, as described in Section 3.3.1, only once for a given environment. The planning process, described in Sections 3.3.2 to 3.3.6 that follow, is executed repeatedly whenever the robot

goals are changed.

3.3.1 Graph Generation and Tree Selection

For the example of Figure 3.1(a), a topological graph G can be constructed by hand as shown in Figure 3.1(b), consisting of $N = 6$ nodes and $E = 6$ edges. Each node is an obstacle-free region of the workspace, at least as large as any of the robots. Edges are created between each pair of adjacent nodes where there exists an obstacle-free path at least as wide as any of the robots. We assume that the initial and goal positions of all robots lie on the nodes of the graph; in this representation, the goal positions of robots R_1, R_2, and R_3 are nodes A, C, and B respectively.

Given the graph representation, we can also select a spanning tree T^* in the graph, that is, a subset of edges connecting all nodes without forming any loops. A given spanning tree has L *leaf nodes* (nodes with only one incident edge), and $N - L$ *interior nodes*. A suitable spanning tree for the example is shown in bold in Figure 3.1(b), and redrawn in a tree form in Figure 3.2. Node C, closest to the geographic center of the map, has been selected as the root, and node B is the root of a subtree. Selecting all edges except for $E - F$ into the spanning tree as shown gives $L = 4$ leaf nodes, A, D, E, and F, and two interior nodes, B and C.

In general the spanning tree is not unique, and a heuristic approach for tree selection was used that tends to maximize the number of leafs and minimize the distance between leafs. Finding the tree with the maximum number of leaves for an undirected graph is an NP-complete problem, but approximate algorithms have been presented [46]. An simple but effective approach used here is to iteratively add edges to the tree that lead to the nodes with the maximum number of incident edges, starting from the root node. Again, the planning algorithm requires the selection of a spanning tree, but is independent of the tree selection method used.

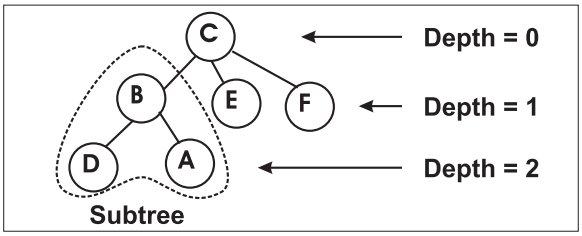


Figure 3.2: A spanning tree T^* for the graph representation of the environment rooted at node C, and a subtree T_B rooted at node B.

3.3.2 Algorithm Overview

The multi-phase algorithm finds a feasible solution to the multi-robot trajectory planning problem by breaking the problem into a sequence of four sub-problems. Each phase can be solved in time proportional to the number of robots by taking advantage of the graph and spanning tree structures developed above.

A plan is first found that moves the robots to the leafs of the spanning tree (Phase 1 of the algorithm), requiring that the number of robots r is less than the number of leaves L. We then use the following observations to plan a sequence of paths to drive each robot to its goal. For a system with $r < L$ robots:

Lemma 1: When all robots occupy leaf nodes, any robot can move to any interior node in the graph G.

Lemma 2: When all robots occupy leaf nodes, any two robots can swap positions.

Lemma 1 is clear since an obstacle-free path can be found between any two nodes through the spanning tree T^*, and no robots remain as obstacles on the interior nodes of the tree. Lemma 2 follows, since with $r < L$ robots, there is always one unoccupied leaf N_{tmp} in the spanning tree. Robots R_i and R_j at nodes N_i and N_j can swap positions by moving R_i to N_{tmp}, R_j to N_i, and R_i to N_j.

Note that these lemmas guarantee that there exists at least one path through the spanning tree. However, a shorter path may exist using graph edges that are not in the tree (e.g., moving from E to F in Figure 3.1(b)). Where an A* search is used in the following steps, the entire graph is searched, and the shortest paths will be selected.

As described in detail below, a plan is constructed by first building a sequence of individual paths, or *segments*, in which one robot moves between 2 nodes (as shown in Figure 3.3). Once all robots have been moved to the leafs of the tree in Phase 1, the lemmas above guarantee that the robots can be arranged in the graph such that every robot will have an obstacle-free path to its goal. This is accomplished in Phase 2 by moving each robot to a node within a subtree of its goal. In Phase 3, we can then move each robot in sequence to its goal. Finally, in Phase 4, the time and distance required to complete the sequence of individual robot movements can be reduced by removing redundant motions and moving robots concurrently whenever possible.

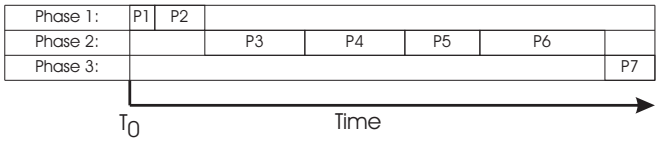


Figure 3.3: Each path segment P_i indicates the motion of one robot. In Phases 1-3, individual collision-free segments are planned and concatenated in time.

The pseudo-code below assumes the following functions are available:

$currentNode(robot)$ returns the node occupied by *robot* at the current timestep of the plan.

$freeLeafNode()$ returns an unoccupied leaf node of the spanning tree. The $freeLeafInSubtree(node)$ and $freeLeafNotInSubtree(node)$ functions perform the same search, restricted to the subtree of *node*, or the subset of the graph not in the subtree of *node*, respectively.

$astarPath(start, end)$ returns the shortest connected sequence of nodes between nodes *start* and *end*, assuming no obstacles in the graph.

$freeAstarPath(start, end)$ returns the shortest connected sequence of nodes between nodes *start* and *end*, avoiding any already occupied nodes.

$findObstacleRobot(path)$ searches for an occupied node in the *path* sequence, in reverse order from the end to start. A reference to the first robot found occupying a node (if any) is returned.

$addPath(path, robot)$ adds the sequence of nodes in *path* as a new sequence for *robot* in the plan, and updates the current position of *robot* to the last node in *path*.

$planRobotToNode(robot, goal)$ uses $freeAstarPath$ to find the shortest obstacle-free path from the robot's current position to the goal, and adds this new trajectory segment using $addPath$.

$subTreeContains(root, node)$ returns true if *node* is in the subtree of *root* within the spanning tree.

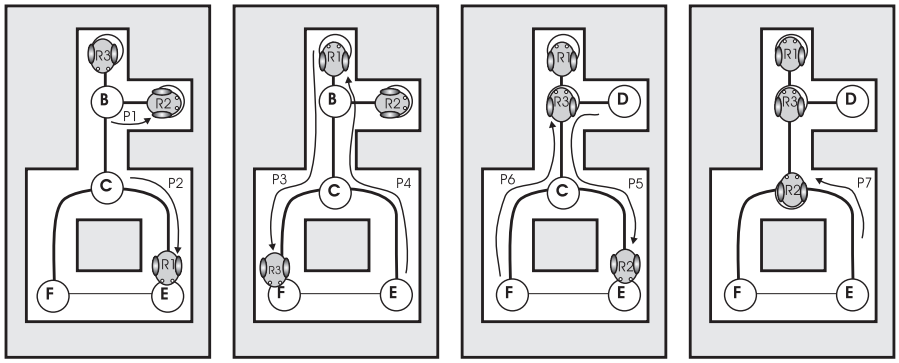
$getBlockedRobot(node)$ searches for robots currently within the subtree of *node*, whose goal is outside of the subtree of *node*.

$sortRobotsByDepthOfGoal()$ orders the robots according to the depth of their goal nodes, from deepest to shallowest, in the spanning tree. This order is applied in the following $for\ each\ robot...$ loop.

The process is shown graphically in Figure 3.4 for the example problem, and each phase is described in detail in the following sections.

3.3.3 Phase 1: Reaching Leaf Nodes

In Phase 1, we develop a plan that will move all robots to leaf nodes of the spanning tree. This is accomplished by repeatedly selecting a robot R_i that



(a) Phase 1 (b) Phase 2-a (c) Phase 2-b (d) Phase 3

Figure 3.4: A multi-phase solution to the planning problem of Figure 3.1(a)

.

is not currently on a leaf node (lines 2-6 of the pseudo-code shown in Figure 3.5), and selecting an unoccupied leaf node L_i (line 7). This is guaranteed to succeed, since there are L leaf nodes, and $r < L$ robots to occupy them. A heuristic may be used to select a leaf node close to the robot or its goal. In the example in Figure 3.1(a), node E may be selected as the leaf node for robot R_1.

An A* search is then used to find a path (sequence of nodes) P_i, from the initial position of robot R_i to the target leaf node L_i, *ignoring all other robots in the system* (line 8). The path P_i is then examined for robots occupying any nodes of the path (line 9). If the path is clear, the path moving R_i to the leaf node is added to the plan (line 11). Otherwise, let R_j be the robot on a node of P_i that is closest to L_i. In this case, we plan for R_j to move to L_i instead, using the obstacle-free subpath of P_i that connects R_j to L_i (lines 13-15).

In Figure 3.4(a), since robot R_2 is an obstacle between the selected robot R_1 and leaf node D, a path P_1 moving R_2 from node B to D is added instead. Continuing the process, R_1 remains to be moved to a leaf node, and either node E or F may be selected, indicated by path P_2.

```
1 function plan_phase1()
2   for each robot
3     start = currentNode(robot)
4     if isLeafNode(start)
5       continue
6     end if
7     leaf = freeLeafNode()
8     path = astarPath(start, leaf)
9     obstacle = findObstacleRobot(path)
10    if (obstacle not found)
11      addPath(path, robot)
12    else
13      start = currentNode(obstacle)
14      path = astarPlan(start, leaf)
15      addPath(path, obstacle)
16    end if
17  end for
18 end function
```

Figure 3.5: Pseudo-code for Phase 1

3.3.4 Phase 2: Sorting Robots by Depth of Goals

In Phase 2, we move all robots into positions where they can reach their goals without creating an obstruction for another robot. The need for this arrangement step can be seen in Figure 3.4(a): robots R_2 and R_3 have goals on the interior nodes C and B respectively, and if either moves directly to its goal, it will create an obstacle for the other. For a general algorithm to resolve this potential deadlock, we consider the problem in terms of robot positions relative to their goals within the spanning tree structure.

Let T_{G_i} be a subtree of the spanning tree with root at the goal node G_i of robot R_i. A deadlock condition occurs only if

- when G_i is occupied, another robot R_j is *inside* the subtree of T_{G_i} and is blocked from reaching its goal *outside* the subtree, or
- when G_i is occupied, another robot R_j is *outside* the subtree of T_{G_i}, and is blocked from reaching its goal *inside* the subtree.

We can prevent these conditions by:

- moving robots to nodes within the subtree of their goal nodes, and
- ordering the depth of the robots within the subtree based on the depth of their goals.

To accomplish this task, we process robots in the order of the *depth* of their goals, that is, the distance from the goal node to the root of the spanning tree (refer to Figure 3.2 and lines 20-21 of the pseudocode in Figure 3.6). For each robot R_i, we determine whether it is already in T_{G_i}, in which case the requirements are already satisfied (lines 24-25). If not, we test whether filling the goal G_i will create an obstacle for any robots in the subtree T_{G_i}, and if so, select the deepest positioned such robot R_j (line 27). The blocked robot R_j can be moved out of the subtree if an unoccupied leaf is available outside of the subtree (lines 30-33). Otherwise, the free leaf must be within the subtree; the depth ordering condition can be achieved by moving R_i to the

available leaf within subtree T_{G_i}, and moving R_j to the original goal node G_i (lines 34-38). This phase achieves the two conditions required above to avoid deadlock conditions when filling interior node goals.

The total path length can be reduced by only partially completing the swap in some cases:

- If the temporary unoccupied leaf used for swapping is not in T_{G_i}, robot R_j may remain at that leaf rather than completing the swap to the previous position of R_i.
- If R_j is the only robot that would be blocked into the subtree, robot R_i can fill its goal node immediately after robot R_j has been moved.

In the example, R_1 has the deepest goal node A, so is processed first. The subtree of the goal consists of only the node A, and contains the robot R_3, which must be moved to avoid the deadlock condition (line 27). R_3 is therefore moved to the unoccupied leaf node F (line 32), before moving R_1 to its goal node A (line 33), shown by paths $P3$ and $P4$ in Figure 3.4(b).

The goals of robots R_2 and R_3 are interior nodes C and B, with C being the root of the spanning tree T^*. R_3 has the deeper goal node B, so is processed first. Its goal node B is the root of the subtree containing nodes A and D, as shown in Figure 3.2, so we must check for robots that would be blocked into the subtree (line 27). Referring to Figure 3.4(b), R_2 at node D is such a robot. We therefore move R_2 to an unoccupied leaf node E (line 32), then plan robot R_3 to its goal node (line 33), indicated by paths P_5 and P_6 in Figure 3.4(c). This leaves R_2 and R_3 in subtrees of their goal nodes, and in the same depth order as their goals, as required.

3.3.5 Phase 3: Filling Remaining Goals

In Phase 3, we move any robots to the remaining unfilled goals. If we plan for robots with goals closest to the top of the tree first (line 47 in Figure 3.7), an obstacle-free path for each robot is guaranteed by the arrangement

```
19 function plan_phase2()
20   sortRobotsByDepthOfGoal()
21   for each robot
22     start = currentNode(robot)
23     goal  = goalNode(robot)
24     if subTreeContains(goal, start)
25       continue
26     end if
27     blockedRobot = getBlockedRobot(goal)
28     if (blockedRobot found)
29       blockedNode = currentNode(blockedRobot)
30       leaf = freeLeafNotInSubtree(goal)
31       if (leaf found)
32         planRobotToNode(blockedRobot, leaf)
33         planRobotToNode(robot, blockedNode)
34       else
35         leaf = freeLeafInSubtree(goal)
36         planRobotToNode(robot, leaf)
37         planRobotToNode(blockedRobot, goal)
38         continue
39       end if
40     else
41       leaf = freeLeafInSubtree(goal)
42       planRobotToNode(robot, leaf)
43     end if
44   end for
45 end function
```

Figure 3.6: Pseudo-code for Phase 2

determined in Phase 2, where the robots are sorted in order of the depth of their goals. For the example scenario, this requires planning robot R_2 to its goal at node C (line 51), resulting in the desired goal configuration shown in Figure 3.4(d).

```
46 function plan_phase3()
47   reverseSortRobotsByDepthOfGoal()
48   for each robot
49     goal = goalNode(robot)
50     if (robot is not at goal)
51       planRobotToNode(robot, goal);
52     end if
53   end for
54 end function
```

Figure 3.7: Pseudo-code for Phase 3

3.3.6 Phase 4: Building a concurrent plan

The plan determined in phases 1-3 consists of a sequences of segments or paths P_i, in which only one robot moves at any time, as shown in Figure 3.3 for the example problem. The sequence of paths guarantees that all robots reach their goal positions without collisions with other robots. However, the sequence of paths is generally very sub-optimal in terms of time and total distance required to reach the goal positions, compared to a decoupled planning solution (if one is possible). Since travel time and distance are often significant evaluation criteria in practical applications, a number of methods may be applied to generate a more optimal solution from the sequence of segments. This stage introduces a tradeoff between solution optimality and computational complexity; the ideal method will depend on the scale of the application (number of robots and size of the map), the computational resources available, the requirements for real-time performance, and the relative importance of optimality in the trajectory solution.

Because the algorithm first moves robots to leaf nodes of the spanning tree (a process that is required to guarantee completeness, but is often unnecessary in the final solution), a significant reduction in total distance traveled can typically be gained by finding and removing any redundant motion. This can be found for each robot by checking all cases where the robot returns to a node it previously visited. If the node was not occupied in the intervening time, the robot may simply remain at that node for the duration.

The result of this first optimization is that there may be steps of the trajectory where no robots are moving; these can be simply removed to reduce the total trajectory execution time.

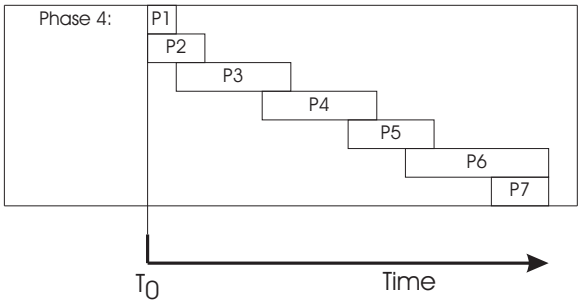


Figure 3.8: Individual path segments are overlapped in time whenever possible while avoiding collisions.

Concurrency by overlapping segments

An additional step is then to allow multiple robots to move concurrently by overlapping the individual segments in time as much as possible without introducing any collisions, as shown in Figure 3.8. Each successive segment of the original plan is added to a concurrent plan by first considering it appended to the end of the plan. The start position of the segment is then moved earlier in time until the motion in the new segment would create a collision between robots in the concurrent plan. The motion of the robot in the new segment is then incorporated into the concurrent plan. This approach was used in generating the simulation results of Section 3.4, involving up to 40 robots

operating in a map of several hundred nodes, with sub-second computation times.

Concurrency by a space-time search

An alternative approach to generating a concurrent plan is to generate a concurrent plan for all robots using a sequential A* search in time and space, based on the method described for general multi-body motion planning by Erdmann and Lozano-Perez [16].

- For each segment of the plan, consider the initial and final positions of the moving robot in each segment.
- Perform an A* search, in a space-time map. This map is based on the topological node-based map used in Phases 1-3, but extended in the time dimension with resolution corresponding to the movement of a robot between two adjacent nodes. The initial and final states for the A* search are the initial and final states of the robots in the trajectory segment.
- Add the A* solution trajectory for the moving robot to the space-time map as an obstacle to be avoided in future searches.

Note that each A* search is guaranteed to find a solution, due to the conditions and ordering of the sequences established in Phases 1-3. In the worst case, for each segment, the initial state will correspond to the final state of the space-time map generated so far, and the A* search will append the same motion as found in the original trajectory segment. Typically, however, the space-time search will find a solution where the motion of the one moving robot can be at least partially concurrent with the motion of previously added segments.

This approach of a full space-time search for each trajectory segment is very effective, as the shortest possible paths are found for each required robot motion, and the maximum concurrency of motion is obtained. However, this

comes at a substantial computational cost, since the space-time map adds an additional dimension to the A* search space, and the length of the time dimension grows with the number and length of individual segments of the original plan. The method was found to be practical for up to 20 robots in simulation, and was used in the real-world implementation described in Chapter 5.

3.3.7 Complexity Analysis

The plan completed at the end of Phase 3 will move all robots to their respective goals, as required for a *complete* planner. In each of the 3 phases, we iterate once over the set of r robots, and require at most 3 (in the case of swapping) A* plans for each. Each A* search has a fixed complexity C that depends on the size of the graph and the heuristic used, but remains independent of the number of robots in the environment. The total computational complexity of the first 3 phases is therefore $O(r \cdot C)$ for r robots.

As discussed above, the complexity of Phase 4 depends on the method used, and the degree of optimization required. In the method of overlapping segments, for example, as each segment overlaps the concurrent plan by one additional step in time, a "collision check" is required for the moving robot at each state in the segment. The worst case complexity of the operation, given a trajectory of s states is $O(s^2)$.

3.3.8 Hybrid Planning

The multi-phase planner is fast and complete; it will quickly generate a solution to the planning problem for a large number of robots in a complex graph. However, the resulting plans are typically sub-optimal, in terms of path length for each robot.

A decoupled planning approach, such as that proposed by Bennewitz [4], can use priority scheduling to consider many different possible plans. Unfortunately, the generation of many plans using different sequence of priorities

is CPU intensive, and may fail to find a solution for complex planning problems. When successful, the resulting plans from the decoupled approach are typically shorter than those found by the multi-phase planner.

To take advantage of the properties of each approach, a hybrid planner was implemented and evaluated. One valid plan is first quickly generated using the multi-phase planner. The decoupled planner is then invoked in an attempt to find a shorter path solution. The decoupled planner may then be terminated at any time, and the most optimal plan selected.

3.4 Simulation Results

The 4-phase planner described above was implemented and evaluated in Monte-Carlo simulations in the underground ("tunnel") mine map shown in Figure 3.4, using between 3 and 40 robots. Refer to Appendix A for an animation video of the simulation. The planner was also evaluated on a map with more open space, shown in Figure 3.4, using between 3 and 150 robots.

For each map, a topological representation was generated from an occupancy grid by finding adjacent circular regions of open space (nodes) and connecting all adjacent nodes by edges. The spanning tree selected for the tunnel map contains 43 leaf nodes, allowing for motion planning of up to 42 robots in the environment. Random initial and goal positions are selected for each robot. For the environment with open spaces, a mesh-like topological structure results, allowing robots to pass each other in the open areas. The resulting spanning tree has 142 leafs, allowing for planning of up to 141 robots.

As expected from the analysis above, the multi-phase planner finds a collision-free plan for every configuration in both maps.

For comparison, a *Decoupled Planner* using a sequential A* planning approach for each robot was also implemented, which randomly selects a priority sequence of robots. This sequential planner finds the shortest collision-free path for each robot through the space-time map, avoiding obstacles includ-

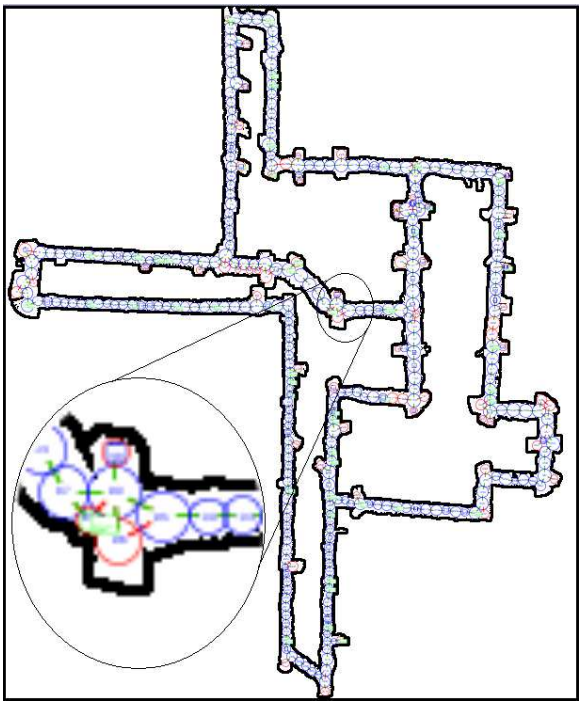
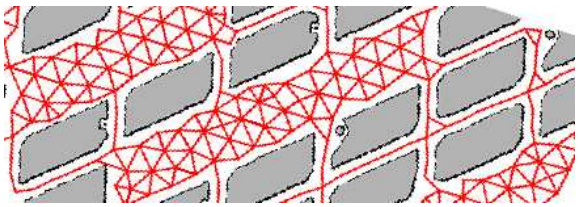


Figure 3.9: Tunnel simulation environment: floor-plan of the Mathies Mine, from http://www.cs.cmu.edu/ 3D/mines/html/map6.html, courtesy of Sebastian Thrun



Open area simulation map of a mine-like environment, created by hand

(a) Open area simulation environment

ing the trajectories of all previously planned robots. The results of such a planner are dependent on the priority sequence used, so up to 100 randomly selected priority sequences were applied for each case in an attempt to find a sequence for which a plan could be found. Finally, the hybrid planner approach described in Section 3.3.8 was used to evaluate the benefit of using a combination of multi-phase and decoupled planning.

The plots in the following sections show the results of applying the algorithms to the same randomly-generated problems in the two different environments.

3.4.1 Planning Success Rate

The first measure of the algorithm performance is the success rate of finding a feasible solution. As expected for a complete algorithm, the success rate of the multi-phase planner is 100% for up to 42 robots given a spanning tree in the tunnel map with 43 leafs. However, the sequential planner failed to find solutions for some randomly generated problems with 14 or more robots, and failed to find solutions for all problems with 25 or more robots.

In the open space map, the spanning tree with 142 leafs guarantees a solution for up to 141 robots using the multi-phase planner. The decoupled planner began to fail for some problems with 25 robots, and failed to find a solution for any problems with 75 or more robots.

The success rate of the sequential planner will increase if more randomly selected priority sequences are tried; however, the planning cost also increases with each additional priority sequence. 100 different sequences was a practical maximum value to run the planner in real-time with less than 10 seconds of CPU time per plan.

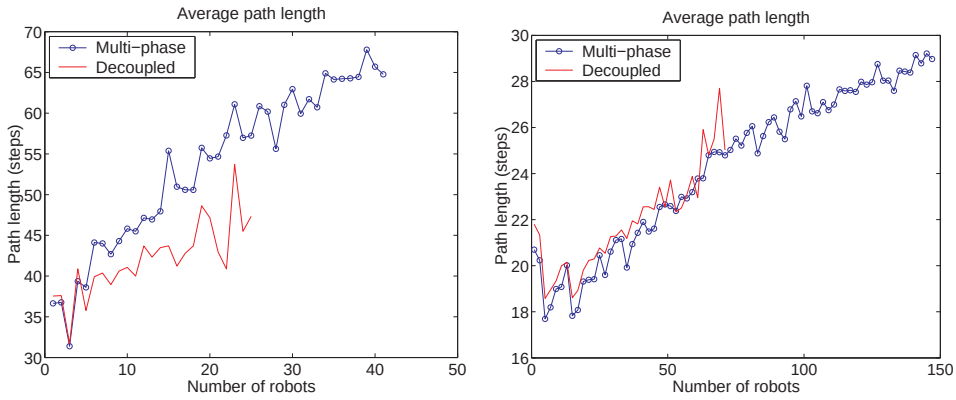
3.4.2 Average Robot Path Length

The average distance required for each robot to travel to reach its goal is plotted in Figure 3.10. The results indicate that in the tunnel map, the multi-

phase planner typically generates longer paths for each robot, particularly as the number of robots increases. This is not unexpected, since the planner first directs robots to positions other than their goals in order to create an obstacle-free path for the final phases of the process.

In the open map, the average path lengths are very similar. This is due to the increased density of leaf nodes in the map; when the multi-phase planner moves robots to leaf nodes, the average additional distance is much less than for the tunnel map.

When the sequential planner begins to fail for some of the randomly generated problems (> 14 robots in the tunnel map and > 25 robots in the open map), the average path length is computed only for those scenarios where a solution was found.



(b) Tunnel Map (c) Open Map

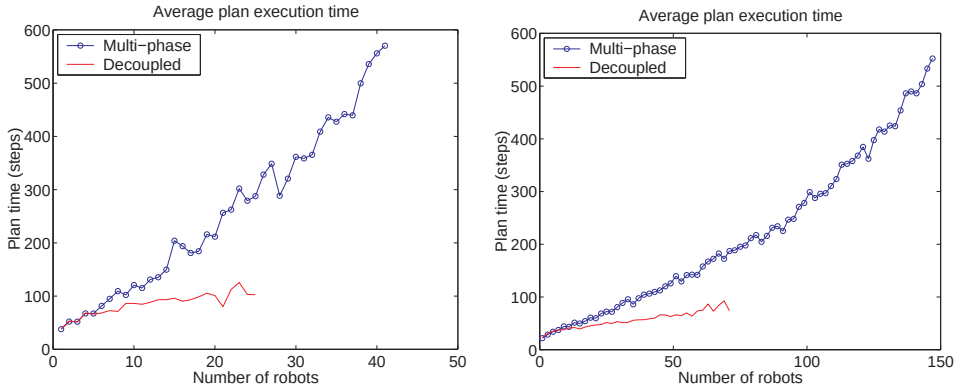
Figure 3.10: Average robot path length generated by each planner

3.4.3 Average Total Execution Time

The average execution time (the number of time steps required for all robots to execute their plans) is plotted in Figure 3.11. The plans generated by the decoupled planner can typically be executed in less time than the multi-phase planner solutions. This is due to the serialized nature of the multi-

phase planner path generation, where the plan is constructed of a sequence of individual robot movements. The execution time of the multi-phase planner is reduced in Phase 4 by executing multiple segments concurrently; however, improving the concurrency involves greater computational complexity.

In contrast, the decoupled planner attempts to immediately move all robots toward their goals from the first time step. This results in greater concurrency, and a shorter execution time. However, this gain comes at a cost of complexity and loss of completeness. In the open-space map, the decoupled planner failed to find solutions for some random scenarios of 25 robots. In the tunnel environment, it failed to find solutions for *any* trials with 25 or more robots.



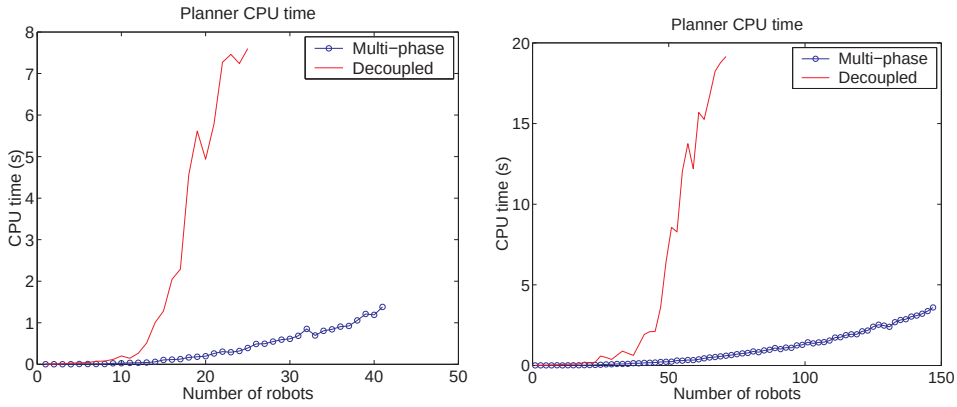
(a) Tunnel Map (b) Open Map

Figure 3.11: Average execution time for paths generated by each planner

3.4.4 Search Cost

The search cost is a measure of the complexity of the planning algorithm, or the time required to complete the search for a feasible solution. Figure 3.12 shows the CPU time required by each algorithm; the processing time has been normalized by the number of robots in the plan, and shows the exponential growth in complexity of the decoupled planning method. The

values indicate the time required to find a feasible solution given the graph representation, and not the (one-time) cost of generating the graph and tree.



(a) Tunnel Map (b) Open Map

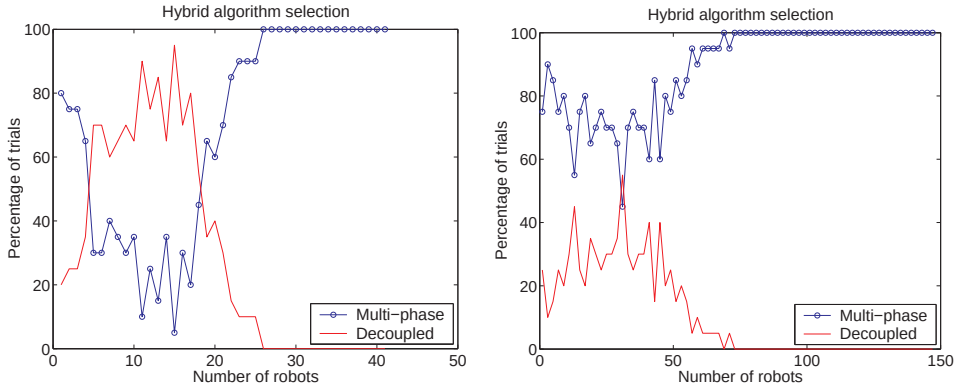
Figure 3.12: Average CPU time used by each planner

These results demonstrate that while a decoupled approach can find shorter paths for simpler planning problems, the multi-phase planner involves much less computational cost. The cost of the sequential planner grows exponentially, since it requires many attempts with different random priority sequences to find a solution. The cost of the multi-phase planning algorithm, however, increases close to linearly with the increase in number of robots. For 100 robots in the open-space map, feasible plans were computed by the multi-phase planner in less than 1.5 seconds using a 1.5 GHz Pentium M processor.

3.4.5 Hybrid Planner

The graph of the algorithm selection in the hybrid scheme, shown in Figure 3.13, indicates the algorithm behaviour as the number of robots in the system increases. The plots show the percentage of time the results of each planner are selected, indicating how often the multi-phase planner generates a more optimal result (a shorter total travel distance) than the decoupled planner.

For very small numbers of robots, the multi-phase planner results are often better than the decoupled planner results. The randomly selected order used by the decoupled planner is typically suboptimal, required longer paths to be generated for some robots. As the number of robots increases, the decoupled planner can often find shorter path solutions. However, beyond a threshold, the decoupled planner fails to find any solutions, and the multi-phase planner results are required.



(a) Tunnel Map (b) Open Map

Figure 3.13: Hybrid Planner Selection

3.5 Discussion and Summary

This chapter presented a multi-robot planning algorithm that is based on a topological graph and spanning tree representation. By breaking the planning algorithm into several phases, it is shown that the algorithm guarantees a solution to the planning problem, and is scalable with linear increase in complexity for up to $r < L$ robots given a spanning tree with L leafs.

In this development, maps of tunnels and corridors were considered specifically, since they present a challenging environment for the coordination of a large number of robots, and occur in practical applications environments such as buildings and underground mines. For more general cases, including

arbitrary obstacles and non-holonomic motion constraints, the generation of a suitable roadmap or graph representation can be a challenging problem in itself. However, once a suitable graph is created, the multi-phase algorithm can be applied directly. For this development, a graph was created by connecting nodes of adjacent circular regions in the obstacle-free workspace. This straightforward approach can be made robust to robot failures by marking the areas around disabled robots as obstacles in the environment, and re-generating the graph representation.

Considering the performance comparison between the sequential planner and the multi-phase planner, it may be advantageous to consider a hybrid approach, taking advantage of the features of both algorithms. By first generating a plan using the multi-phase planner, a feasible solution can be generated very efficiently. To search for a more optimal plan, a sequential planner could then be applied to the same problem, and permitted to run within the time bounds of the application.

In comparison to a decoupled sequential planning algorithm, the multi-phase planner typically produces longer paths, but at a much reduced computational cost when planning for many robots. A hybrid algorithm demonstrated the value in using both the multi-phase and decoupled planning methods when planning for a variable number of robots in an environment. A real-world implementation of the planning algorithm with physical robots is presented in Chapter 5, which demonstrates the practicality of the multi-phase planner in real-world applications.

Applying this planning algorithm to a multi-robot task allocation problem is the topic of the following chapter.

Chapter 4

Multi-Robot Task Allocation in Corridor Environments

Given a set of tasks to accomplish, and a team of multiple robots available to perform the tasks, a system is required to assign each task to a particular robot. The goal of such a system is to allocate the tasks in an optimal manner, minimizing a cost function such as the total time to complete all of the tasks, or the total energy expended by all of the robots.

This chapter considers a specific type of task, where robots are required to visit certain locations in their environment, and the environment is composed primarily of narrow corridors or tunnels. A task is defined as the node in the graph representation of the environment that is closest to a location of interest. This type of problem arises in a number of practical applications. For security applications, a team of robots may need to patrol a building, periodically visiting a set of rooms and monitoring for intruders. Another example includes mining applications, where loose rock and ore must be picked up from multiple locations by autonomous earth-moving vehicles and delivered out of the mine. In each of these cases, a tunnel or corridor environment can limit the ability of robots to pass one another at arbitrary locations; this creates a need for appropriate task allocation and coordinated motion planning.

A *class* is associated with each task, corresponding to the type of operation to be performed at the task location. Each robot also has an associated class, corresponding to the type of operations it can perform. For the building patrol application, the particular robot that visits each location is not relevant; the robots in the team are typically considered functionally equivalent, and any of the robots can perform the observation operation required at any task location. We define this as a *single-class* task allocation problem, where all tasks and all robots are of the same class. The task allocation problem is then to find an efficient path for all of the robots through their environment, such that at least one robot visits each of the task locations. Depending on the particular application, the objective may be to minimize the travel time of all robots, or to minimize the time between visits to each task (for maximum coverage in the building security example).

For the autonomous mining application, all vehicles are not functionally equivalent; empty trucks can be assigned certain tasks (picking up fresh material), and trucks that are full of material can be assigned other tasks (delivering the material at the mouth of the shaft). We define problems of this type as *multi-class* task allocation problems. This chapter discusses the different requirements for *single-class* and *multi-class* task allocation systems, and presents algorithms for both scenarios, suitable for many robots operating in a confined environment.

4.1 Literature Review

Several approaches have been applied to the general problem of allocating tasks between multiple robots in a team. A more specific problem, in the context of this thesis, is the allocation of tasks that are the visiting of specified locations (task points) within the environment. To gauge the complexity of the problem, this section first reviews a similar problem, the *traveling salesman problem*, which has been well studied in combinatorial optimization (refer to [44] for a thorough discussion). The remainder of the review focuses

on market-based approaches, which have been applied to the multi-robot task allocation problem in numerous applications.

4.1.1 Traveling Salesman Analogies

One goal of the task allocation problems addressed in this chapter is to minimize the total travel time or distance of the robots. This objective is closely related to the commonly studied traveling salesman problem (TSP). A review of research on the TSP can provide some insights into the complexity of the problem, and suggest directions toward an effective solution. In the standard traveling salesman problem, one salesman has a set of cities to visit, and the goal is to find the shortest route that visits each of the cities exactly once and returns to the starting location. A brute-force search of all possible permutations will yield the optimal answer; however, there are $N!$ permutations for a problem of N cities. The TSP has been shown to be NP-complete [61], and exact solutions can be found by algorithms with exponential complexity $O(2^N \times N^2)$. In practice, heuristics and approximate solution methods (see [68] for examples) are typically used to find solutions for large values of N, or if computation time is limited (as for real-time robot motion planning).

The multi-robot planning problem described here includes a number of variations from the standard TSP:

- The allocation of tasks between multiple robots must be considered. For N tasks and a team of R robots, $N \times R$ different allocations are possible, before considering the order of the task execution for each robot. This is often termed a multiple traveling salesman problem, or MTSP.

- The constraints of motion within the environment must be considered; while the standard TSP assumes a fully connected graph, where the salesman can travel directly between any two cities, the robot task allocation system involves motion planning using only the open areas

of the environment. As a result, unlike the standard TSP, the same points may be visited multiple times in one solution. As tasks are completed, new tasks may be added, creating a dynamic allocation and planning problem requiring a real-time algorithm.

- The task allocation must take into account the motion of other robots working in the same environment. This is particularly significant in constrained environments of tunnels and corridors, where there may not be room for multiple robots to move simultaneously, and explicit coordination is required to avoid collisions.

- For this application, we require tasks to be visited repeatedly, while the standard TSP is complete after a single visit to each task point.

These variations from the standard TSP definition (in particular, the requirement for coordinating collision-free motion of all robots in the solution) increases the complexity of the problem and prevents the direct application of existing TSP solution methods. However, some of the common TSP solution heuristics, such as selecting a nearest neighbour task, can be used to guide the multi-robot task allocation process.

4.1.2 Market-Based Methods

The concept of using an economic model to allocate tasks between agents was proposed by Smith [76], in a system called *Contract Net*. Contract Net defines a protocol for negotiation communication, allowing some agents (acting as managers) to announce tasks, and other agents (acting as contractors) to respond with bids. Contracts are established by managers comparing bids received from contractors, and allocating tasks to the lowest cost bid.

This approach has been applied to many task allocation problems, summarized in a survey by Dias et al. [14]. In that survey, market-based systems are characterized by the following 5 features:

- The task to be accomplished can be subdivided into subtasks, which can be allocated for individuals or sub-groups to perform. A limited set of resources are available to be shared by the team members.
- The quality of solutions to the task problem can be quantified by a global *objective function*.
- An individual *utility function* quantifies a robot's relative ability to perform a subtask.
- A mapping function relates the individual utility functions and the global objective function, quantifying how well an individual's contribution to a subtask helps achieve the overall team task.
- Individual subtasks and resources can be allocated to individuals within the team, using a mechanism such as auction system.

This general definition can be applied to many task allocation problems, including the multi-robot navigation problem considered in this chapter. The navigation problem involves determining which robots should attend to which task locations, as well as determining mutually collision-free trajectories for all of the robots to reach those tasks. In terms of the characteristics proposed above, the components of a market-based solution to the coordinated multi-robot motion planning and task allocation system are:

- The global task consists of one or more sets of locations (task sites) to be visited by the robots. The set of resources are the open spaces of the environment which must be shared for navigation over time.
- The solution quality is measured as a function of the total distance traveled by the robots, or the time between visits to the task points.
- A robot's ability to achieve a task may be measured in terms of the distance between the task and the robot's currently planned trajectory.

- The mapping function determines the change in the objective function based on a robot's new trajectory including the new task location.
- Assigned robot trajectories, as functions of time, determine the allocation of the available resources (the open space) and tasks amongst the robots.

Considering the challenge of task allocation and trajectory execution, Dias et al. note that "[planning] coordination between teammates during task execution becomes necessary when robots interfere with each other during execution... It is also necessary in domains where teammates continuously constrain each others actions" [14]. These are the situations commonly encountered in the problem of task allocation among many robots in a tunnel environment.

4.1.3 Task Allocation Solutions

TraderBots is a "market-based approach for resource, role, and task allocation in multirobot coordination", developed by Dias and Stentz [13]. The architecture is fundamentally distributed, gaining the advantages of scalability and robustness. However, it opportunistically forms sub-groups to take advantage of the performance benefits of centralized task allocation.

Simmons et al. present a system for coordinating multi-robot planning for exploration and mapping [74]. In this application, value is based on the amount of information gained, measured by the number of *frontier cells* (unexplored area) that will be observed. Cost is measured by the total distance traveled.

4.1.4 Inter-Robot Coordination

Some approaches to task allocation require no explicit coordination between robots. Parker's *ALLIANCE* architecture for allocation of tasks among heterogenous robots [62] is one such example. Two primary goals of the *AL-*

LIANCE architecture are robustness and adaptive control of the robot team; these are accomplished with a distributed, behaviour-based approach to task allocation. In this approach, the activation and inhibition of behaviours in one robot is triggered by the selection of tasks by other robots.

Auction-based approaches, also requiring no explicit coordination, have been applied to the task allocation problem, as presented by Gerkey and Mataric [22], and Sariel and Balch [72]. In these systems, each robot bids on a task, based on the perceived cost of performing the task. Each task is assigned to the robot offering the lowest bid, and the tasks are then executed by the robots. Within this framework, various strategies can be applied to tune the algorithm for a particular problem. Mataric et al. present an investigation of four different auction-based task allocation strategies [50]. The varying strategies considered the effect of commitment (whether a robot completes a task before accepting another, or opportunistically switches tasks) and coordination (whether tasks are exclusively assigned to individual robots) on the system performance. The conclusion from Mataric's work is that no single strategy is optimal for all task allocation scenarios. The most suitable strategy depends on factors that can change dynamically in a real-world environment, such as the amount of sensor noise in the system.

The specific task allocation scenarios considered in this thesis involve the movement of multiple robots to task locations within a corridor environment. As discussed in previous chapters, planning for collision-free motion of multiple robots within such an environment may require the explicit coordination of motion between robots to avoid collisions and deadlocks. A suitable task allocation strategy for this problem should therefore take into account the explicit coordination and trajectory planning that may be required to execute the allocated tasks within a corridor environment.

4.2 Single-Class Task Allocation

In this section, an auction-based task allocation approach is considered, specifically for problems involving multi-robot navigation in corridor environments, such as the building patrol robot scenario described above. For this problem we assume:

- All robots and tasks are of a single class. That is, all robots are equivalently capable of performing any of the tasks.
- A communication network is available between each robot and a central processor. No communication is required directly between individual robots.

The auction-based approach considers each task in turn, and requests a bid from each robot, which includes the trajectory that the robot will follow to reach the task point. The cost of the bid is determined as the average time between visits to all of the currently assigned tasks (though this objective function may be varied depending on the particular application). The task is then assigned to the robot producing the lowest-cost bid. The trajectories of all robots are updated based on the new trajectory of the winning robot. The process is then repeated for each remaining unassigned task, as summarized in the pseudo-code in Figure 4.1.

The pseudo-code below assumes the following functions are available:

$currentNode(robot)$ returns the node occupied by *robot* at the current timestep of the plan.

$astar\_search(robot, start, end)$ returns the shortest obstacle-free path for *robot* between nodes *start* and *end*, taking into account as obstacles all trajectories currently assigned to other robots.

$assign\_task(robot, task)$ allocates the specified task to the robot, and updates the current set of trajectories to include the trajectory generated for the winning bid by that robot.

eval_bid(*trajectory*) computes the time between visits for all task nodes in the trajectory, according to Equation (4.1).

4.2.1 Bid Generation

To generate a bid for a task, a robot must find a trajectory that:

- passes through the currently auctioned task point, as well all of the robot's previously assigned tasks, and
- avoids collisions with other robot, based on their current trajectories.

To generate the optimal bid, the trajectory should minimize the time between successive visits to all of the robot's tasks. This time between visits includes the time since each task point was last visited (such that tasks which haven't been visited recently should be included close to the beginning of the trajectory), and the time until the task will be reached (such that the length of the trajectory should be minimized, to reach all task points as quickly as possible).

The bid cost B_r for robot r is evaluated at time t according to Equation (4.1):

$$B_r = \sum_{j=0}^{N} ((t - t\_prev_j) + (t\_next_j - t)) \tag{4.1}$$

where j iterates over the N tasks assigned to robot r, and t_prev_j and t_next_j are the times of the previous and planned visits to task j respectively.

For each bid, all possible permutations of the task order could be considered, to ensure that the optimal possible trajectory is found for each robot. However, as the number of task points increases, the exponential computational complexity (as with the standard TSP problem) makes this approach impractical. Instead, a fixed sequence of current tasks is maintained by each robot, and the auction task is considered for insertion before each task in the current list, and at the end of the list.

```
1 function single_class_auction()
2   for each auction_task
3     for each robot
4       bid_value = bid(robot, auction_task)
5       if bid_value < best_bid_value
6         winning_robot = robot
7       end if
8     end for
9     assign_task(winning_robot, auction_task)
10  end for
11 end function
12
13 function bid(robot, task)
14  for each allocated_task in task_list
15    insert(task_list, task)
16    trajectory = astar_search(robot, task_list)
17    if trajectory not found
18      continue
19    bid_value = eval_bid(trajectory)
20    if bid_value < best_bid_value
21      best_bid_value = bid_value
22      best_trajectory = trajectory
23    end if
24    remove(task_list, task)
25  end for
26  return best_bid_value
27 end function
28
29 function astar_search_list(robot, task_list)
30  initialize empty trajectory
31  start = current_node(robot)
32  for each task in task_list
33    path = astar_search(robot, start, task)
34    if path not found
35      return not found
36    append(trajectory, path)
37    start = task
38  end for
39  return trajectory
40 end function
```

Figure 4.1: Pseudo-code for single-class task allocation

For each insertion point, an A* search is used to generate the shortest obstacle-free path through the environment that moves the robot from its current position to each task location in sequence. Each solution is then checked for collisions with other robots on their currently selected trajectories, and rejected if a collision would occur.

4.2.2 Completeness, Optimality and Scalability

The auction-based approach for single-class task allocation is particularly suited to problems requiring coordinated trajectory planning in confined environments. A significant feature of this approach over other methods is its completeness — it is guaranteed to find a feasible, collision-free solution that reaches all of the task points within a connected graph.

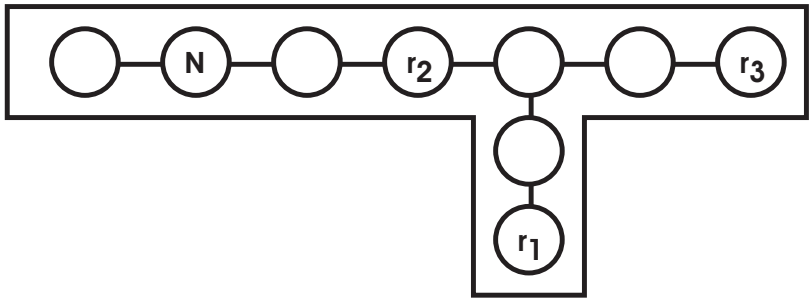


Figure 4.2: Illustrative graph for single-class task allocation.

To show the completeness of the algorithm, we first consider the task of planning for one of the robots to reach a specific task node in a connected graph map. A simple graph is shown in Figure 4.2 for illustration, with a task node N and robots occupying the three nodes indicated by r_1, r_2, and r_3.

Lemma 1 For every node n in a connected graph, there is an obstacle-free path between node n and at least one robot r in the map.

To see that this is true, consider a path generated by A* from node n to a candidate robot r_1. If no other robots lie on the path, we have the solution using robot r_1. However, if another robot lies on this path, is it not an obstacle-free path. In this case, consider the first robot r_2 which lies along the path from n to r_1. Since all robots are of the same class, we can consider r_2 as the candidate robot instead. Since r_2 is the first robot on the path from n to r_1, and the A* path to r_2 is a subset of the path to r_1, r_2 must have an obstacle-free path to node n. Thus, an obstacle-free path can always be found between any node n and one of the robots.

Considering the task allocation process, as every task is auctioned, every robot submits a bid if it finds a collision-free trajectory to the task node. By Lemma 1, at least one robot will have an obstacle-free path from its final position to the task node, guaranteeing that at least one bid will be submitted as each task is auctioned. This process guarantees that a collision-free solution will be found for any set of assigned tasks, as required for a complete algorithm.

The use of an A* search and the consideration of each possible position of new tasks into the existing task lists generates shortest-path solutions at each step. Since the algorithm doesn't consider re-ordering or re-assignment of previously assigned tasks, the solution is not globally optimal. The globally optimal solution would determine the shortest possible path of all robots that would pass through each of the tasks. As discussed above, such an approach would involve exponential complexity. Instead, the computational complexity of this method increases linearly with the number of robots and the number of tasks assigned, so the method is practical for real-time planning applications with many robots.

4.2.3 Algorithm Behaviour

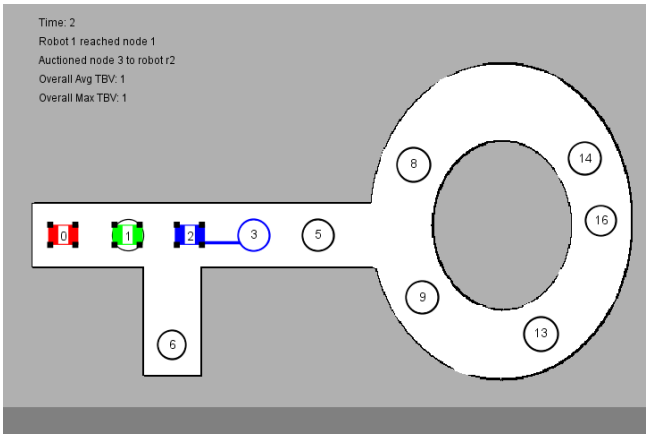
To demonstrate the behaviour of the single-class task allocation, three robots are shown working in a small environment (a graph of 16 nodes), where 8 nodes are defined as the task points to be monitored. Figures 4.3 and 4.4

shows a sequence of images from the task allocation process. The positions of robots r_0, r_1, and r_2 are indicated by the red, green, and blue rectangles respectively. The planned trajectory of each robot is indicated by a solid lines of the corresponding colour. The task nodes are indicated by circles, which are the colour of the robot assigned to the task, or black if no robot is assigned.

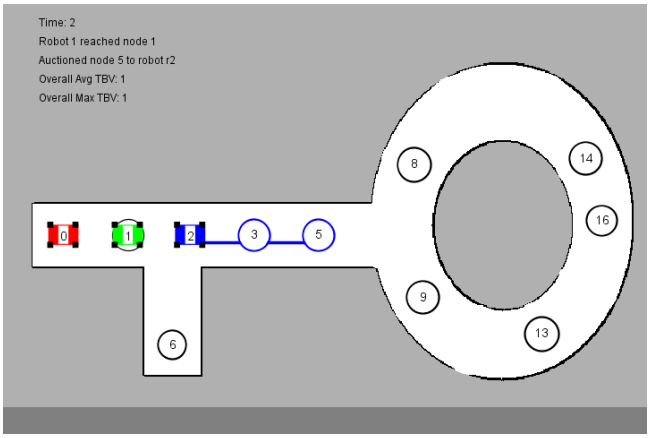
In the sequence of Figure 4.3, each of the tasks is auctioned in sequence before the robots begin to move. In (a), robot 2 is first assigned to node 3 — no other robot has a collision-free path to node 3, so robot 2 will offer the only bid for that task. Likewise, in (b) and (c), robot 2 has the only collision-free path to nodes 5 and 8, and is assigned those tasks as well.

Node 9 is then auctioned in (d), and assigned to robot 1. This is a result of the current planned trajectory of robot 2, which gives robot 1 the shortest collision-free trajectory to node 9. Sub-figures (e)-(h) show the continuing process for the remaining nodes.

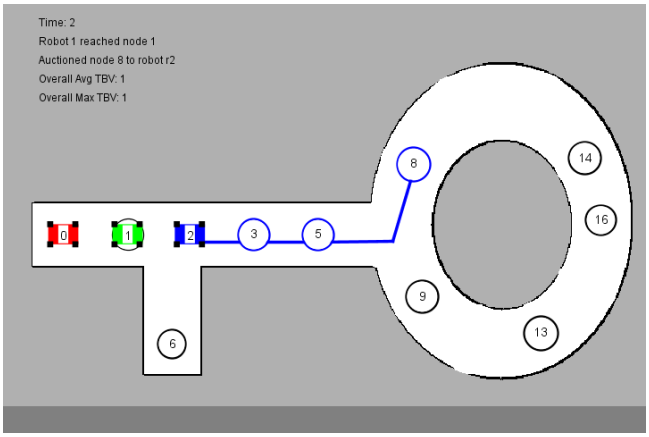
After all nodes have been auctioned, the robots begin following the assigned trajectories. As each robot reaches a task node, the node is marked as un-assigned and is re-auctioned at a subsequent time step. Figure 4.4 shows the following eight auction events as trajectories are dynamically updated.



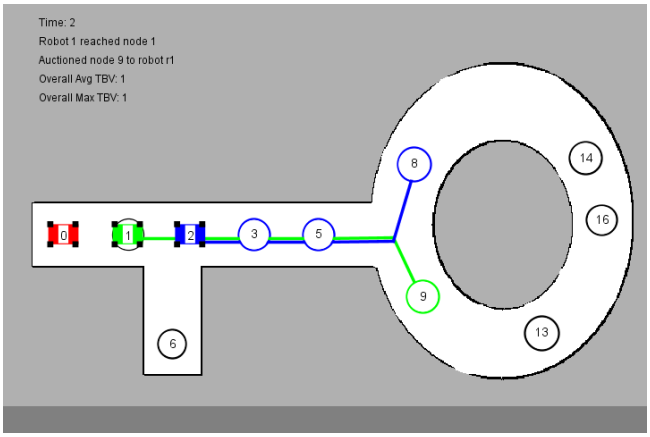
(a) Node 3 is assigned to robot 2



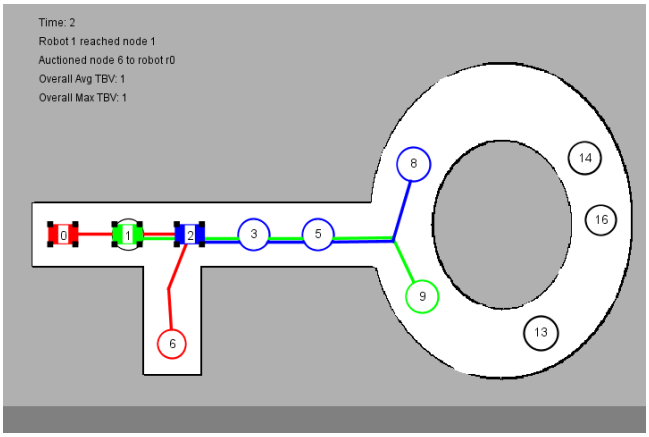
(b) Node 5 is assigned to robot 2



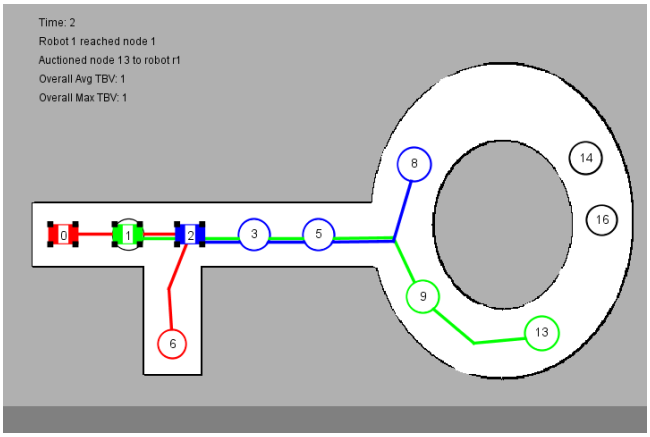
(c) Node 8 is assigned to robot 2



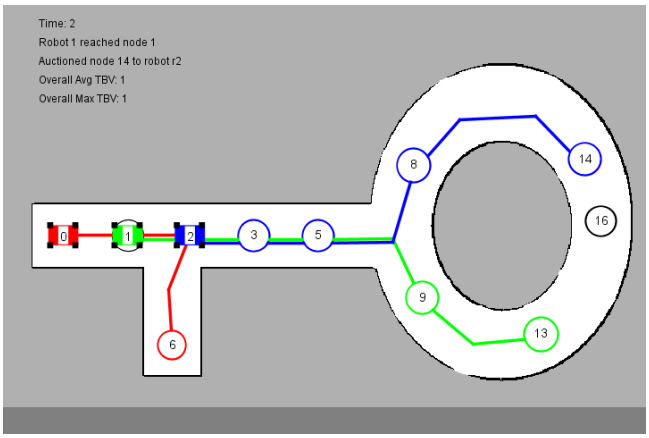
(d) Node 9 is assigned to robot 1



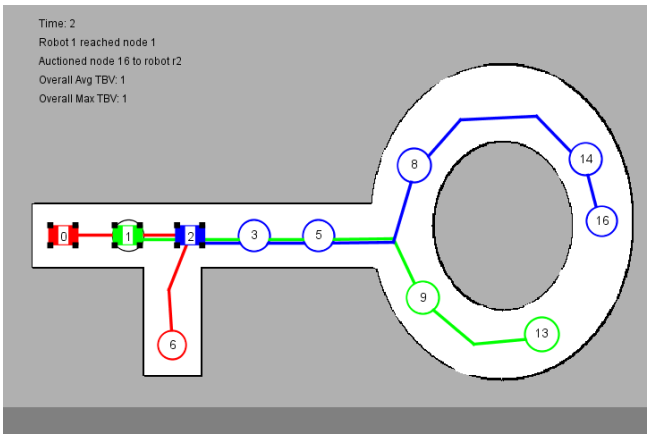
(e) Node 6 is assigned to robot 0



(f) Node 13 is assigned to robot 1

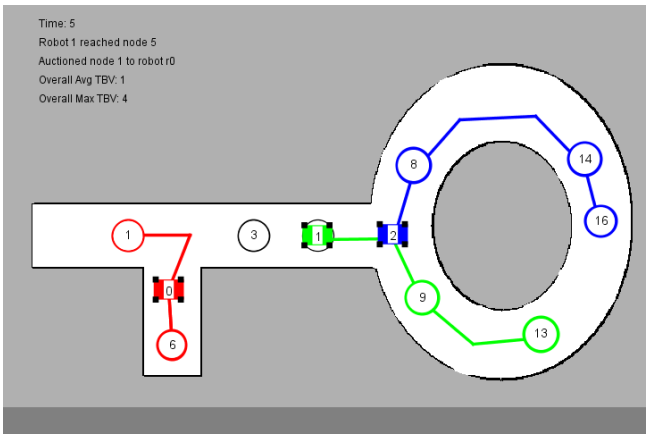


(g) Node 14 is assigned to robot 2

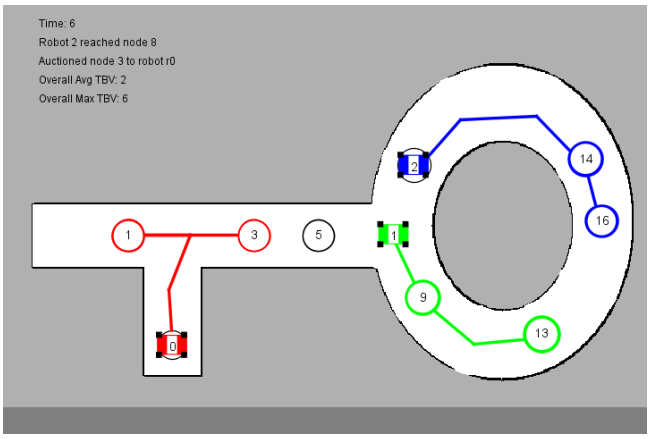


(h) Node 16 is assigned to robot 2

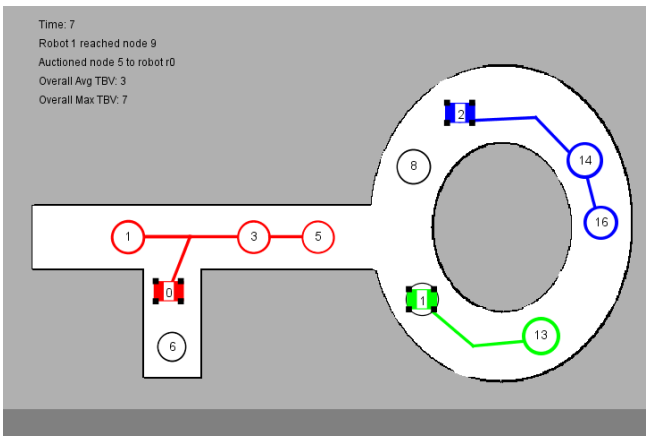
Figure 4.3: Initial task allocation sequence for 3 robots and 8 tasks



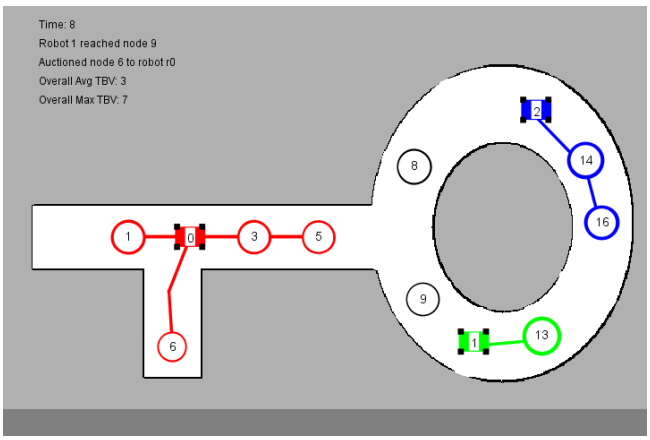
(a) Node 1 is assigned to robot 0



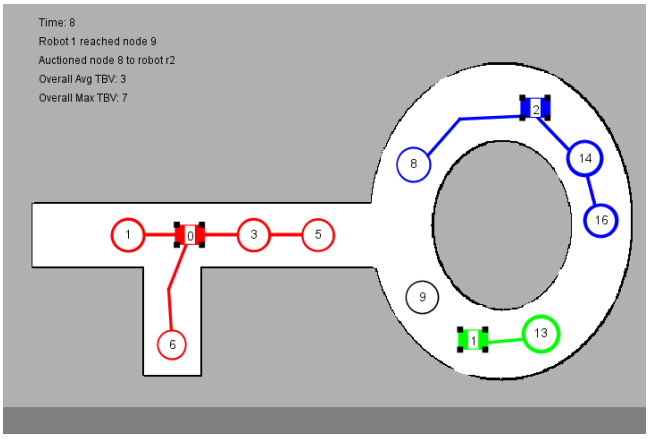
(b) Node 3 is assigned to robot 0



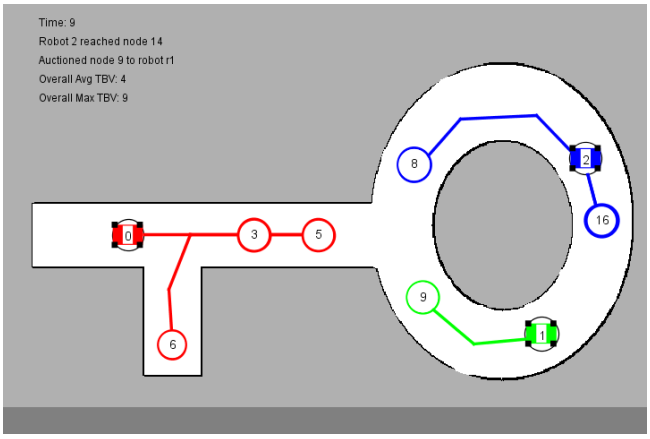
(c) Node 5 is assigned to robot 0



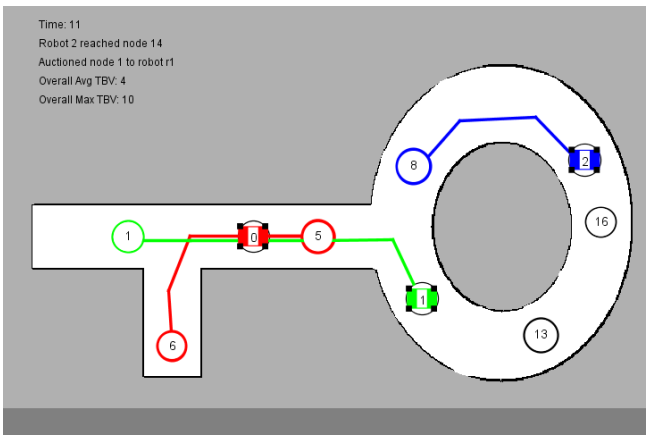
(d) Node 6 is assigned to robot 0



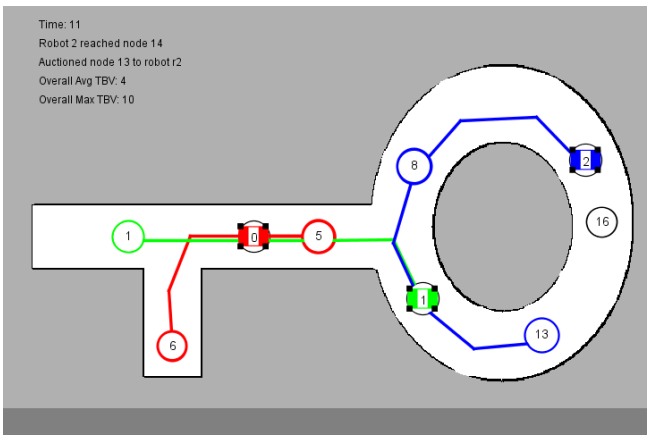
(e) Node 8 is assigned to robot 2



(f) Node 9 is assigned to robot 1



(g) Node 1 is assigned to robot 1



(h) Node 13 is assigned to robot 2

Figure 4.4: Task allocation sequence for 3 robots and 8 tasks

4.2.4 Simulation Performance Results

The performance of the algorithm can be quantified by a metric corresponding to the objective for the particular problem. For the building patrol application, we use the average time between visits of each task as the metric — a lower value indicates better performance.

For a single robot, the minimum average time between visits corresponds to the optimal solution of the corresponding traveling salesman problem, if each task is only visited once in the circuit. For the problem defined here, the average time between visits may be lower, since some tasks may be visited multiple times as a robot completes a circuit. As more robots are added to the team, the number of tasks allocated to each robot decreases, and the optimal average time between visits is expected to decrease.

To investigate this performance in a larger environment, the algorithm was run in simulation with the map shown in Figure 4.5 [1]. The number of robots was varied between 1 and 50, while holding the number of tasks fixed at 50.

In this type of tunnel environment, the shortest path to repeatedly visit a set of randomly selected nodes is frequently a circuit, or closed path, which avoids backtracking over tasks that have already been visited. One circuit around the tunnel traverses approximately 230 nodes. We can therefore expect an average time between visits to each node of approximately 230 time steps using a single robot.

As robots are added, the average time between visits should reduce to approximately $230/R$ for R robots if the algorithm effectively allocates the tasks amongst the robots. The average time between visits will decrease as expected if either

- all of the robots follow the same circuit, with each robot visiting all of the tasks in turn, or

[1] Map source used with permission, courtesy of Sebastian Thrun, available from http://www.cs.cmu.edu/~thrun/3D/mines/groundhog/loops/map-2nd-corr.png.

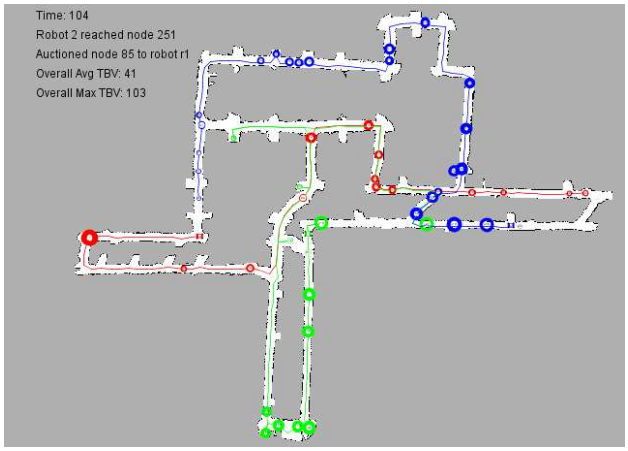


Figure 4.5: The tunnel simulation environment, with 50 tasks assigned to three robots. Three different classes of tasks are indicated by the red, green, and blue circles.

- the set of tasks is divided amongst the robots, resulting in a shorter circuit (of approximately $230/R$ steps) for each robot.

The latter case is the pattern that emerges in the example shown in Figure 4.5.

The resulting average time between visits as the number of robots changes is shown in Figure 4.6. As predicted, the time between visits is approximately $230/R$ for R robots, indicating that the algorithm effectively distributes the tasks amongst the robots. However, the performance remains approximately constant for $R > 30$ robots.

Two factors contribute to the smaller incremental performance improvement as R increases. First, the impact of additional robots decreases as predicted by the $230/R$ curve. In addition, increased congestion in the environment reduces the incremental system performance as the number of robots increases. This effect is examined further in the following section (refer to Figure 4.9).

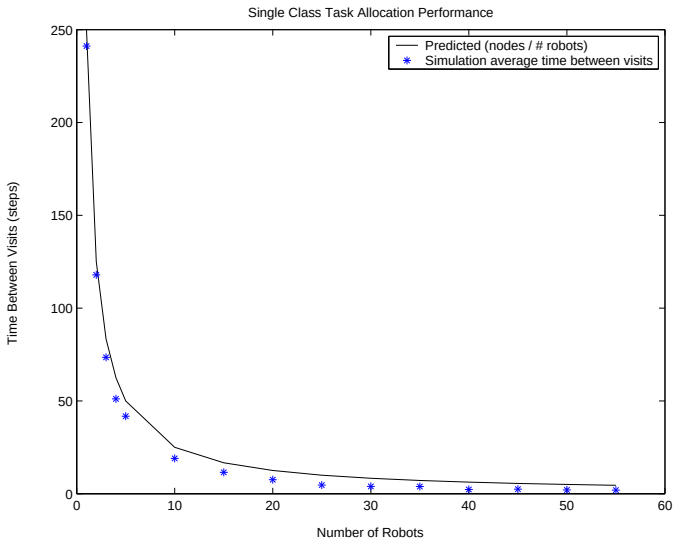


Figure 4.6: Simulation performance results in the tunnel environment, showing the average time between visits to each node.

4.2.5 Computational Complexity

A primary design goal of this algorithm is scalability; the method should be suitable for real-time planning of many robots operating simultaneously in a confined environment. The real-time performance of the algorithm can be seen in Figure 4.7, showing the CPU time required for the simulations with up to 50 robots and 50 tasks in the tunnel environment. The simulation was run on a single 1.4GHz Intel Pentium M processor.

Figure 4.7 shows an interesting result; the computational cost increases initially, as the number of robots increases, then decreases as the number of robots approaches the number of tasks. To understand this trend, consider the elements that contribute to the complexity: the number of auctions performed, and the number of bid evaluations required in each auction.

First, as the number of robots increases, the number of auctions required at each time step increases proportionally, as shown in Figure 4.8. This corresponds to the performance results in Figure 4.6; as the number of robots

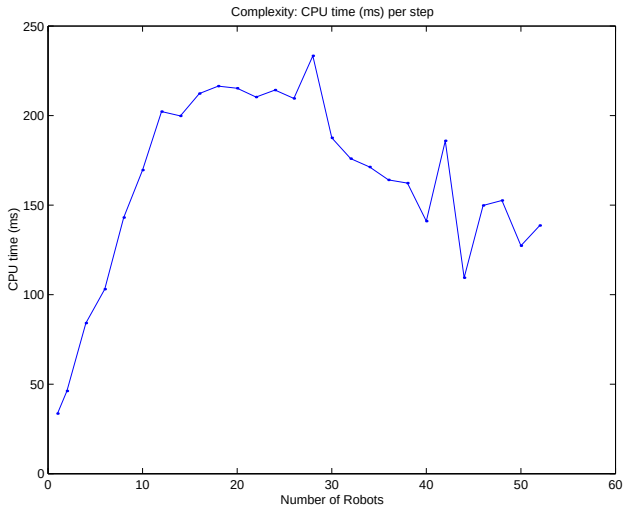


Figure 4.7: Real-time CPU usage (ms) per simulation time step.

increases, the average time between visits to task nodes decreases, requiring the tasks to be re-auctioned more frequently. As the number of robots increases beyond $R = 30$, the time between visits becomes relatively constant, as does the number of auctions required.

Each auction requires a number of evaluations of each potential solution. This evaluation is performed for every plan that is generated as robots attempt to insert a new task into their existing task list. The number of evaluations required per auction decreases as the number of robots increases, as shown in Figure 4.9. This trend is due to the increasing congestion in the map; the greater the number of robots, the fewer obstacle-free paths can be found between robots and tasks, and the fewer bid evaluations are required.

4.2.6 Observations

From the simulation results a number of observations can be made:

- The algorithm effectively distributes tasks for the patrol-type application among a team of R robots, achieving a time between visits of less

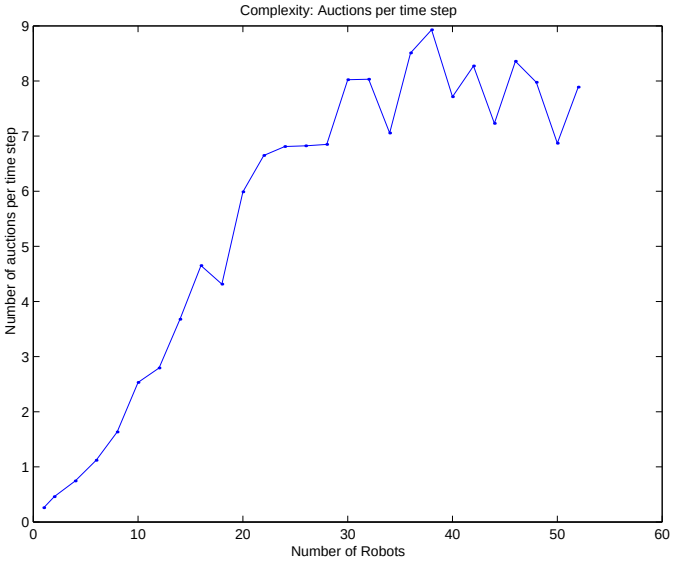


Figure 4.8: Number of auctions per time step

than N/R where N is the number of steps required for a single robot to reach all tasks in a circuit.

- As the number of robots increases, the performance reaches a minimum time between visits due to the lower incremental impact of each robot, and increased congestion in the environment; in the example scenario, this occurs at approximately $R = 30$ robots.
- The algorithm is suitable for real-time performance, even with a centralized implementation; in the example scenario, solutions require less than 300ms at each time step using a single 1.4GHz processor.

4.3 Multi-Class Task Allocation

For some practical applications, tasks and robots can be categorized into multiple classes, where certain robots can address only a subset of the tasks,

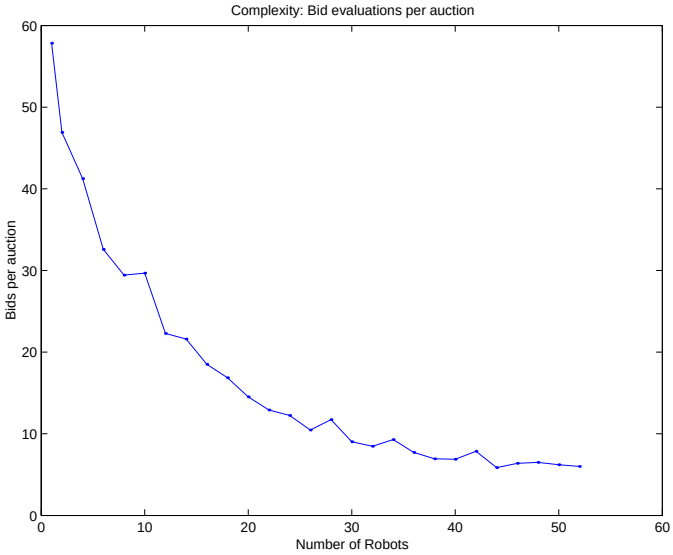


Figure 4.9: Number of bids per auction

and other robots can address a different subset. We define these as multi-class task allocation problems.

The example of autonomous mining vehicles can be formulated as a multi-class task allocation problem, by considering vehicles that are full of material as one class, which are assigned tasks of dumping the material they are carrying at the mouth of the shaft. Vehicles which are empty are considered as a separate class, and are assigned the tasks of picking up material from the face where it is being extracted.

In such cases, the approach described for single-class task allocation cannot be used, because robots of one class may present obstacles to the movement of robots of another class. Lemma 1 no longer holds because, unlike the single-class case, tasks cannot be swapped between robots if they are of different classes. This results in scenarios where a simple iterative A* approach may not be sufficient to find a set of collision-free trajectories to reach all of the assigned tasks.

For example, as shown in Figure 4.10, if robot r_A follows a direct path

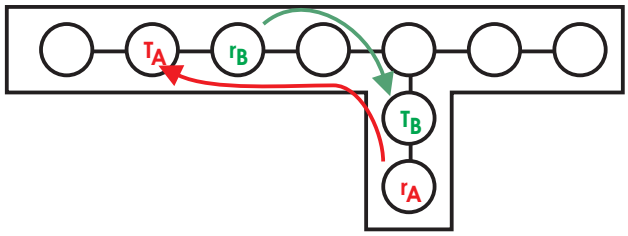


Figure 4.10: A simple multi-class task allocation example.

to task T_A, it will create an obstacle for robot r_B to reach task T_B, and vice versa. In such cases, a coordinated motion planning and task allocation approach, considering the required trajectories of all of the robots in the system simultaneously, is required to guarantee that a solution is found.

4.3.1 Coordinated Task Allocation and Path Planning

To address the multi-class task allocation problem, the planning algorithm presented in Chapter 3 can be applied to find coordinated collision-free trajectories for each robot in a given set of task assignments, and an optimization search method can be used to determine the most suitable task assignment. An initial solution is found by assigning each robot one task of its class, and using the multi-phase planner to find a trajectory for all robots from their current positions to their assigned tasks. The planner guarantees that a solution will be found for any initial and goal conditions, and generates fast solutions suitable for real-time planning applications.

The results of the multi-phase planner can then be evaluated as an optimization objective function, based on total distance traveled or total time required to complete the plan for the selected set of tasks. Continuing the optimization process, alternative task allocations can then be generated, evaluated, and selected if found to be better than prior solutions.

The generation of alternative task allocations in the optimization process can be based on a complete search, a heuristic search (if some a-priori knowl-

edge of the specific application is available), or other optimization methods such as genetic algorithms.

This optimization approach allows for a distributed implementation, where each robot can evaluate a set of different task allocations for the entire group, and the best solution found within the team is selected. Note that, since the multi-phase planner guarantees a fast solution to the initial task allocation selection, this is an anytime algorithm, making it suitable for real-time applications. The optimization process can be run at every timestep, and the best trajectories available at the end of each time-step can be assigned to the robots, as summarized in the pseudo-code in Figure 4.11.

The pseudo-code below assumes the following functions are available:

$current\_node(robot)$ returns the node occupied by *robot* at the current timestep of the plan.

$multi\_phase\_plan(start\_state, end\_state)$ generates, using the multi-phase planner, a set of trajectories moving the robots from their current positions ($start\_state$) to the final position defined by the task allocation (end_state).

$assign\_trajectories()$ assigns the trajectories most recently computed by $multi\_phase\_plan()$ for the robots to execute.

$random\_robot()$ randomly selects one robot from the team.

$random\_boolean()$ randomly returns $true$ or $false$.

$assign\_random\_task(allocation, robot)$ updates *allocation* with a randomly selected task for *robot*, from the set of all tasks of the same class as *robot*.

$swap\_random\_task(allocation, robot)$ updates *allocation* by swapping the tasks of *robot* and another randomly selected robot of the same class.

tasks_achieved(*plan*) returns the number of task points reached in the *plan*.

total_travel_time(*plan*) returns the total number of steps that will be traveled by all robots if the *plan* is fully executed.

```
1 function multi_class_task_allocation()
2   current_value = evaluate(current_allocation)
3   while time_remaining > 0
4     new_allocation = update(current_allocation)
5     if evaluate(new_allocation) > evaluate(current_allocation)
6       current_allocation = new_allocation
7       assign_trajectories()
8     end
9   end while
10 end function
11
12 function update(allocation)
13   robot = random_robot()
14   if random_boolean() == true
15     new_allocation = assign_random_task(allocation, robot)
16   else
17     new_allocation = swap_random_task(allocation, robot)
18   end
19   return new_allocation
20 end
21
22 function evaluate(allocation)
23   plan = multi_phase_plan(current_state, allocation)
24   visits = tasks_achieved(plan)
25   distance = total_travel_time(plan)
26   objective = value/distance
27   return objective
28 end function
```

Figure 4.11: Pseudo-code for multi-class task allocation

The behaviour of the algorithm is shown in Figure 4.12 with a simple example of three robots, and two classes of tasks:

- 6 'fill' locations indicated by blue rings, on the right hand half of the map.
- 3 'dump' locations indicated by purple rings, on the right hand half of the map.

The colour of the robots indicates their current class, corresponding to the class of goals to which they should be assigned. Blue robots are empty, and are assigned to 'fill' task locations to pick up material. When an empty robot reaches its assigned fill task, it changes colour to purple, indicating the robot is carrying material, and should be assigned to a dumping task location (indicated by the purple rings).

When a full robot reaches its assigned dumping task, its color changes back to blue, indicating that it is empty, and is then assigned once again to one of the filling tasks. The currently assigned tasks of the three robots are indicated by the goal indicators $G0$, $G1$, $G2$, above the task circles.

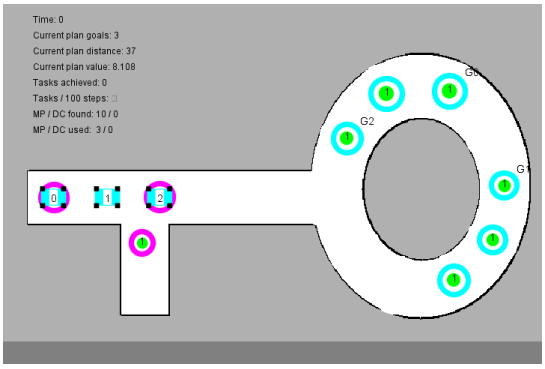
Note that when robots take material from a fill location, the number of units available at that task (indicated by the number within the blue task circle) is decremented. When the number of units available reaches zero, the dot at the center of the task is changed from green to red, indicating that the task is idle, and will not be assigned to a robot. The number of units available at each fill task is periodically incremented, simulating the production of more material at each task to be moved.

The task selection optimization used for this example is straight forward, since the number of tasks and robots is quite small. At each time step, for each task, a trajectory is generated using the multi-phase planner, sequentially assigning each robot (within the appropriate class) to the task. The trajectory that reaches the maximum number of goals with the minimum total distance traveled is selected.

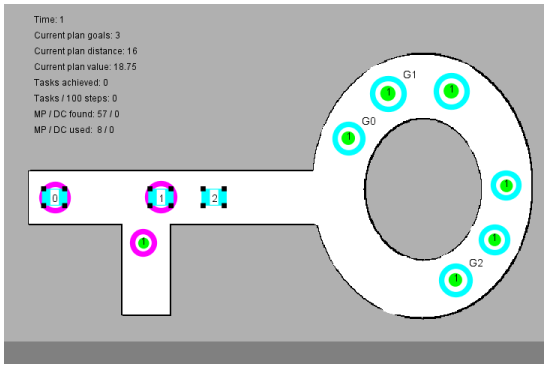
For a larger team of robots operating in a larger environment, a more sophisticated algorithm is required to effectively sample the search space of possible task assignments. One approach evaluated here is to consider

random modifications to the goal state of the current plan at each time step, in a manner similar to the mutation step of genetic algorithms.

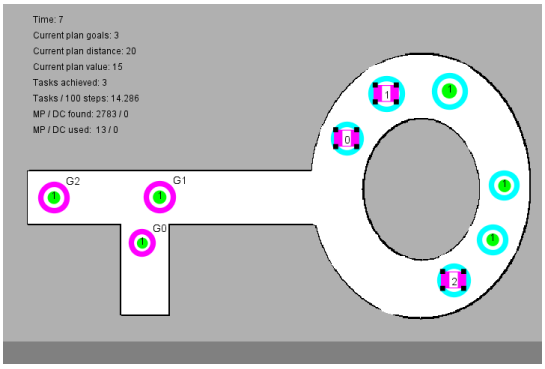
A multi-robot plan is found to the new goal state, and if the objective function for the new plan is better than the current plan, the current plan is replaced with the new plan.



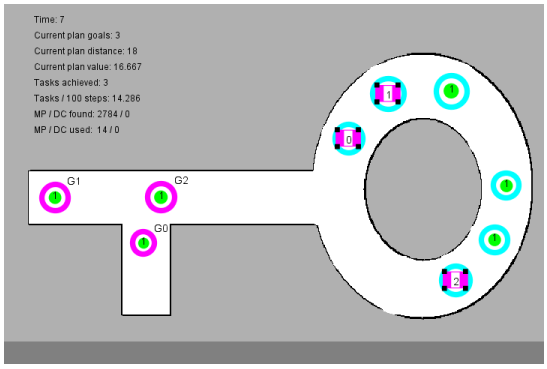
(a) 'Fill' tasks are assigned to all robots



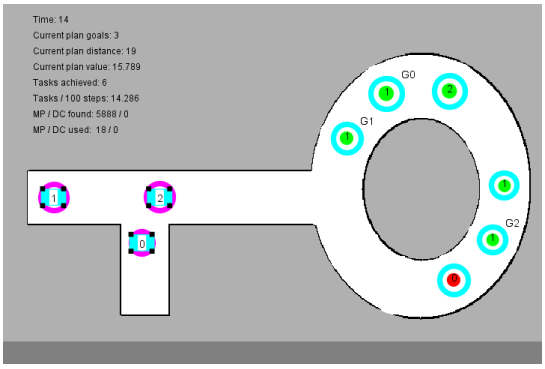
(b) As robots are traveling, a more efficient task allocation is found and assigned



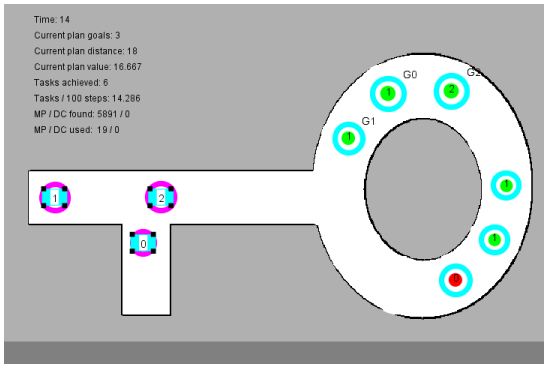
(c) Robots reach their 'fill' tasks, and are assigned 'dump' tasks



(d) a more efficient task allocation is found by swapping goals G1 and G2



(e) Robots reach the 'dump' tasks, and return trajectories to 'fill' tasks are assigned



(f) A more efficient allocation is assigned

Figure 4.12: Multi-class task allocation simulation sequence

4.3.2 Optimization Performance

Performance of the planner is measured by how well a particular objective function is maximized or minimized. For the examples below, the objective is to maximize the number of task points reached per 100 units of distance traveled:

$$J = 100 \cdot \frac{N_{tasks}}{Distance} \tag{4.2}$$

In the results below, the calculation of the objective function for the current plan is based on the number of goals and total distance remaining to travel in the current plan. Note that as a plan is executed, the number of goals remaining in the plan decreases when individual robots reach their goals, decreasing the objective function value. However, the number of steps remaining also decreases (by the number of robots moving during that time step), which increases the objective function value. These two trends result in different optimal behaviour, depending on the relative distance between goals for multiple robots.

- If several robots are approaching their goals at approximately the same time, the optimal solution selected is typically to complete the current plan, waiting for all robots to reach their goals.
- If one robot reaches a goal while others still have a long way to travel, the optimization typically finds a new plan, driving the completed robot to a new goal while the others continue on to their original goals.

Small Environment Performance

To illustrate the performance of the algorithm in a simple environment, performance results are plotted for a simulation using the multi-phase (MP) planner for the scenario shown in Figure 4.13.

Figure 4.13 shows the performance of the algorithm as defined by the objective function in Equation (4.2). The instantaneous objective value, based on the number of goals and number of steps to move in the current

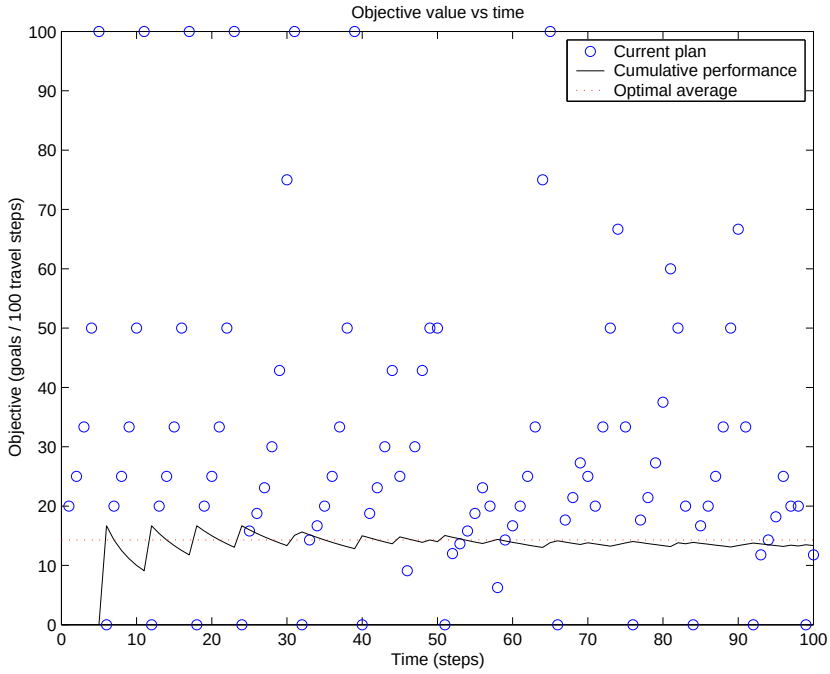


Figure 4.13: Performance results in a small environment: the average system performance approaches the optimal performance of 14.28 tasks per hundred steps traveled.

plan, is shown by circles in Figure 4.13. This instantaneous value fluctuates between zero (no tasks are assigned) and 100 (all robots will reach a task in one step).

The overall objective function, plotted as a solid line, is computed from Equation (4.2) at each time step, based on the actual number of tasks reached and the total distance traveled by all robots up to that time. This cumulative objective function is expected to converge to an average value as the simulation continues.

For simple scenarios, where an optimal solution can be found by inspection, the overall objective function can be compared to the optimal value. For this scenario, considering the time for robots to travel in a complete

circuit from the dumping tasks to the top 3 fill tasks, back to the dumping tasks, to the bottom 3 fill tasks, and back to the dumping tasks, we have

- $N_{tasks} = 12$ (3 robots reaching 4 tasks points each), and
- $Distance = 84$ (3 robots each traveling a total of 28 steps between nodes in the graph).

Evaluating the objective function from Equation (4.2) gives $J = 100 \cdot \frac{12}{84} = 14.28$ which is the overall objective value that would be approached by an ideal planner. This optimal objective is plotted in Figure 4.13 as a dashed line. As expected, the overall objective approaches a steady-state average value, slightly below the optimal solution.

Decoupled and Hybrid Planning

A sequential planning approach to multi-robot planning can also be applied to this problem. While a sequential planner doesn't guarantee a solution for any arbitrary initial and goal states, if a solution is found it is often shorter than than the guaranteed solution found by the multi-phase planner.

By running both planners for each goal configuration considered, a hybrid planner is achieved, where a solution is guaranteed by the multi-phase planner, and a more optimal solution may be found by the sequential planner.

The simple scenario was executed using three different planners: the multi-phase (MP) planner, the decoupled (DC) sequential planner, and the hybrid planner using both the multi-phase and sequential algorithms. The performance of the three planners is summarized by the objective function value computed over time, shown in Figure 4.14.

In this scenario with 3 robots, the three different planners give very similar performance, with cumulative objective function values within 10% of each other. For this example, the optimal task assignment does not typically require the multi-robot coordination provided by the multi-phase planner solution, so the decoupled planning approach is often effective. As is shown in

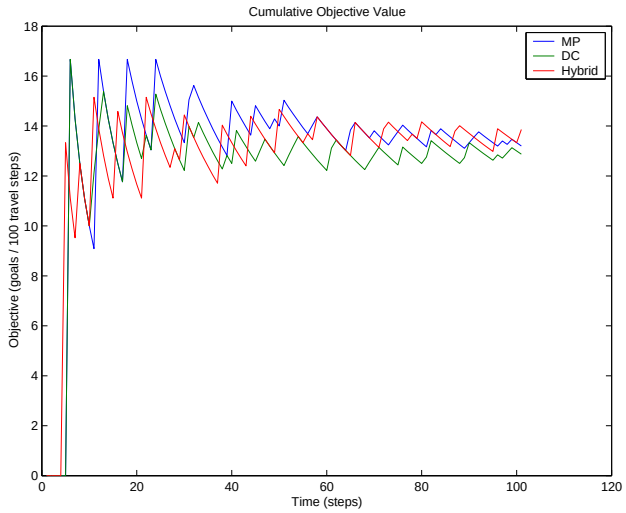


Figure 4.14: Algorithm performance comparison with 3 robots

the following section however, the benefit of the multi-phase planner becomes apparent as the number of robots and tasks increases.

Large Environment Performance

Creating a similar scenario in a larger environment, 20 robots were simulated in the tunnel mine environment shown in Figure 4.5. The scenario was again simulated using the multi-phase, decoupled, and hybrid planners. The performance of the three planners is summarized by the objective function of Equation (4.2) computed over time, shown in Figure 4.15.

In this scenario, the decoupled (DC) planner average objective value converges toward about 2 goals per 100 steps traveled, while the multi-phase planner performs 30% better, achieving about 2.6 goals per 100 steps. The hybrid planner results track the poorer decoupled planner performance.

These results suggest that the multi-phase planning approach to task allocation is more scalable to larger environments with a larger number of robots. To see the reason for this difference in scalability, the plan evaluation

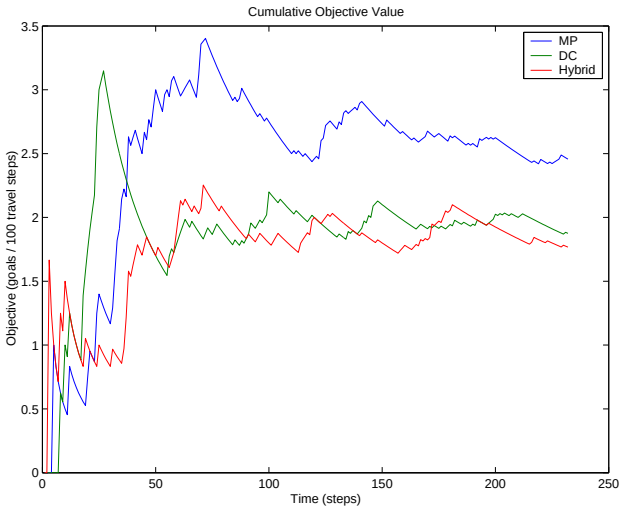


Figure 4.15: Algorithm performance comparison with 20 robots

rate is shown in Figure 4.16 for the simulations in the tunnel environment. The total number of plans evaluated at each time step (or equivalently, the number of different task assignments considered) is plotted over time, for the three different simulations.

The benefit of the multi-phase planner for large environments is clear from this plot. Since the multi-phase planner quickly generates a solution for any task allocation, it allows many more different allocations to be evaluated at each time step. The multi-phase planner evaluates about 10 times as many goal scenarios as the sequential planner; the sequential planner becomes slower as the number of robots increases, because it must consider an increasing number of permutations of planning priorities.

4.4 Summary

For large-scale planning problems, the ability to quickly evaluate alternative task allocations is more significant than the possibly shorter paths that may or may not be found by a decoupled planner. The scalability of the

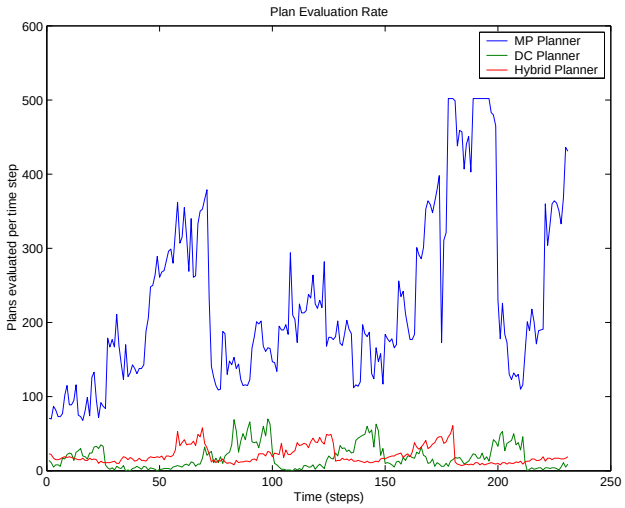


Figure 4.16: Task assignment evaluation rate comparison with 20 robots

multi-phase planner makes it suitable for large real-time task allocation applications in tunnel and corridor environments. Robustness of the system is achieved by dynamically updating the task allocation for the currently active set of robots. The following chapter presents an implementation of the task allocation system with physical robots, demonstrating its practicality real-world applications.

Chapter 5

Multi-Robot System Implementation

5.1 Introduction

To validate the algorithms presented in Chapters 3 and 4, a real-world implementation was developed using a system of three WBR-914 PCBot robots, provided by FrontLine Robotics. The environment for testing the robots was the fourth floor of building E3x at the University of Waterloo, consisting primarily of a 2m wide, 32m long hallway. One of the robots in the environment is shown in Figure 5.1.

This chapter describes the architecture and software used to coordinate the team of robots and execute plans generated by the multi-phase motion planner. The implementation demonstrates the execution of multi-phase plans, requiring the robots to execute coordinated trajectories to arbitrarily assigned goals using the multi-phase planner presented in Chapter 3, and multi-class task allocation, where the trajectories are dynamically updated using the task allocation strategy presented in Chapter 4.

As shown in Figure 5.2, a centralized host PC coordinates the system, communicating with the mobile robots via wireless ethernet. Each robot localizes itself within the hallway using an a-priori map of the floor plan;

Figure 5.1: The PC-Bots in the test environment

the same map is used by the centralized controller for planning the robot trajectories.

5.2 Robot Platform

The system implementation was developed around three WBR-914 PCBot robots. Each robot includes a 2-wheel differential-drive transmission, sonar sensors, motor controllers, analog sensor interfaces, and an on-board PC-based computer, as detailed in Table 5.1.

Motion estimation is accomplished by integrating the commanded speed of the differential drive wheels. This basic odometry works well on the PCBot platform, since the DC stepper motors track desired rotations to a resolution of less than 1°, and the rubber drive wheels and casters result in very little wheel slip. To enable global localization with respect to a world map, a Hokuyo URG series scanning laser rangefinder was mounted on the top of each robot. This sensor provides range measurements from 20mm

Height	54 cm
Weight	25 kg
Mobility	Front and rear roller ball casters Left and right differential drive wheels
Propulsion	Differential drive DC stepper motors
Sensors	8 fixed position sonar sensors
CPU	Intel 2 GHz CPU and 1 GByte of RAM
Storage	80 GB Hard disk
Communication	802.11 wireless networking interface
Power	2 x 12V 9Ah lead acid batteries

Table 5.1: WBR-914 PCBot Specifications

to 4000mm, with a scanning range of 225°, and a resolution of 0.36°. The unit is 50mm square and 70mm high, and interfaces with the on-board CPU through either an RS232 serial or USB interface.

5.3 Control Architecture

The robot control architecture is based on the *Player/Stage* robot server developed by Gerkey, Vaughan, and Howard [24]. *Player* is a network server for robot control. An overview of the architecture is shown in Figure 5.2, and each component is described in more detail in the following sections.

5.3.1 Player Client and Server

It provides an interface to a robot's sensors, motors, and control processes through a TCP/IP network socket. The modular architecture abstracts the details of particular hardware components, and allows new hardware and software components to be integrated into the open-source system. For the PCBot system, 5 Player drivers were required:

WBR914: This module is supplied by WhiteBox Robotics to support the

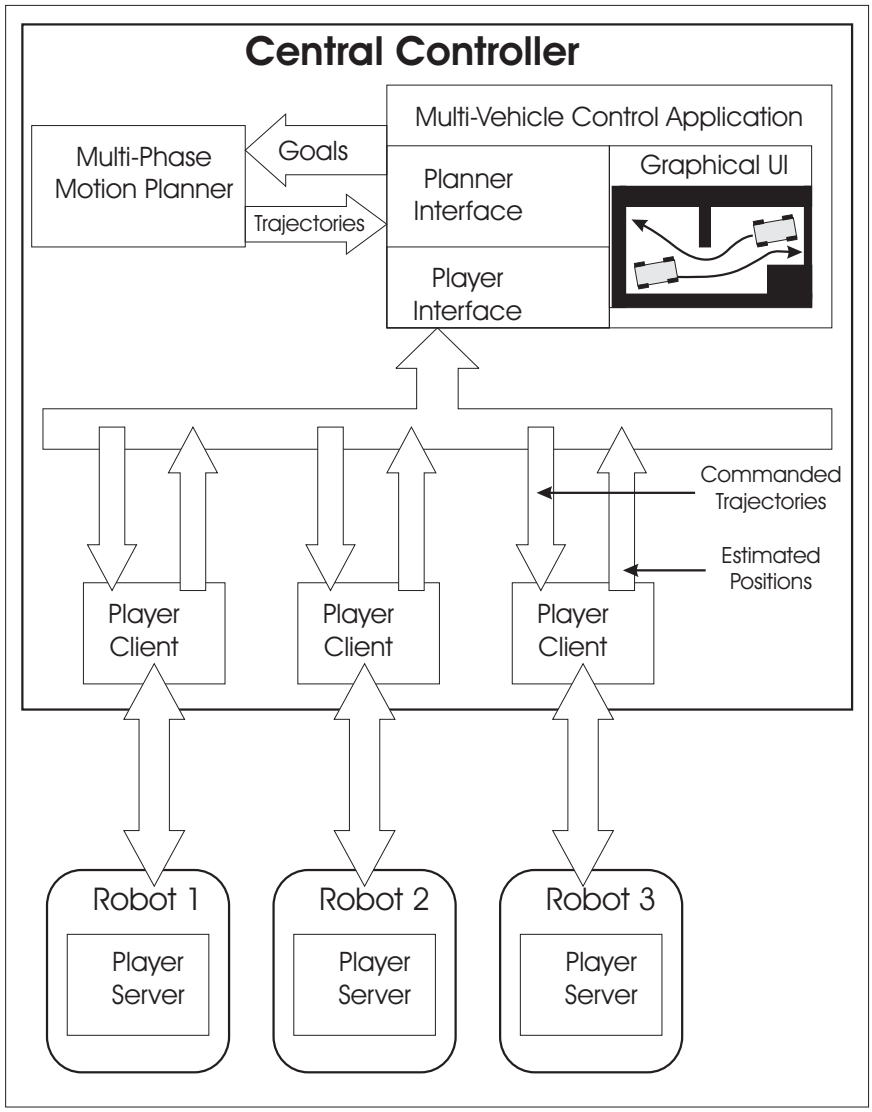


Figure 5.2: The multi-robot communication architecture

PCBot platform, and provides an interface to the drive the motors and read back odometry measurements based on rotations of the stepper motors.

URG laser: The URG driver provides an interface to read the range measurements from the scanning laser rangefinder.

Mapfile: This module provides an interface to a bitmap representation of the environment, stored as an image file.

Localization: The *Adaptive Monte-Carlo Localization* algorithm developed by Fox [17] is implemented in the AMCL module included with the Player software package. The module performs a particle filter localization based on the laser scan data from the URG laser module to estimate the position and orientation of the robot in the map provided by the Mapfile module.

Trajectory tracker: The trajectory tracking module drives the robot through a sequence of waypoints at specified times, by monitoring the current position estimate from the AMCL module, and commanding desired velocities through the WBR914 module. The trajectory tracker was developed specifically for this implementation, and is described in more detail in Section 5.3.3.

The Player server software runs under the Linux operating system on each of the mobile robots. To control the robots, a Player client program connects to the server through a network socket, through which it can receive state information and send control commands.

5.3.2 Control Application

For this project a centralized control architecture was used, where one PC takes the role of a central controller and interfaces directly with each of the

robots. The interface to each robot requires an instance of a Player client, as shown in the architecture diagram of Figure 5.2.

In this system, the Multi-Vehicle Control Application is a program written in Java, including:

- a graphical user interface, showing a map of the environment, the robots, and their trajectories,
- a configuration module, which defines the map and robot parameters,
- an interface to the multi-phase planner, which accepts a set of current and goal locations for the robots and returns a set of solution trajectories, and
- interfaces to the Player clients for each robot.

The output of the multi-phase planner is a set of collision-free trajectories for the robots to follow — that is, a list of waypoints for each robot. Each waypoint is specified by three coordinates, x_i, y_i, t_i, defining a point in the 2 dimensional map, and a time when the robot should arrive at that point.

5.3.3 Trajectory Tracker

To execute the planned trajectory, a *trajectory tracking* function was added to Player as a plug-in module. This tracker module runs on each robot, and receives trajectories from the central controller. The goal of the tracker is to follow the trajectory by arriving at each waypoint position x_i, y_i at the specified time t_i. It achieves this by planning a planning a sequence of straight-line segments between waypoints, and in-place rotations at waypoints when required to change direction.

For each waypoint in the trajectory, the tracker determines an entry angle at which it will arrive (based a straight line from the previous waypoint) and an exit angle at which it will leave (based on a straight line to following waypoint). If the entry and exit angles are the same, and the trajectory

does not require the robot to pause at the waypoint, the tracker plans a constant-velocity motion through the point, with a velocity determined by the specified arrival time and the distance from the previous waypoint. If the entry and exit angles differ, the tracker plans a decelerating approach to stop at the waypoint, followed by a fixed position rotation toward the next waypoint. Using these velocity profiles, the tracker drives the robot at the corresponding translational and rotational velocity using the standard Player interfaces. The tracker module continually monitors the estimated position of the robot determined by the localization module, and updates the velocities to correct for deviations from the specified trajectory using a PD control loop.

5.4 Multi-Phase Plan Execution

The coordinated planning task selected for the robots was to cooperatively navigate in a long, narrow corridor, repeatedly traveling between an elevator and randomly selected locations, as would be required for autonomous delivery robots. We assume that robots may not pass each other within the hallway due to size and safety constraints. As a practical application, note that the trajectory planning problem in this scenario is similar to that for several autonomous mining vehicles operating in a common area of an underground mine.

The floor-plan and graph representations of the environment are shown in Figure 5.3. By selecting a node near the center of the map as the root of the spanning tree (node 3 for example), the tree has $L = 4$ leafs, and the planner is guaranteed to find a solution for up to $r = 3$ robots.

As shown in Figure 5.2, the host PC monitors the positions of the robots, and assigns a new goal position for each robot as the goals are achieved. Whenever a new goal is assigned, a new multi-robot plan is generated for all of the robots, and the individual trajectories are transmitted to the trajectory followers of each robot.

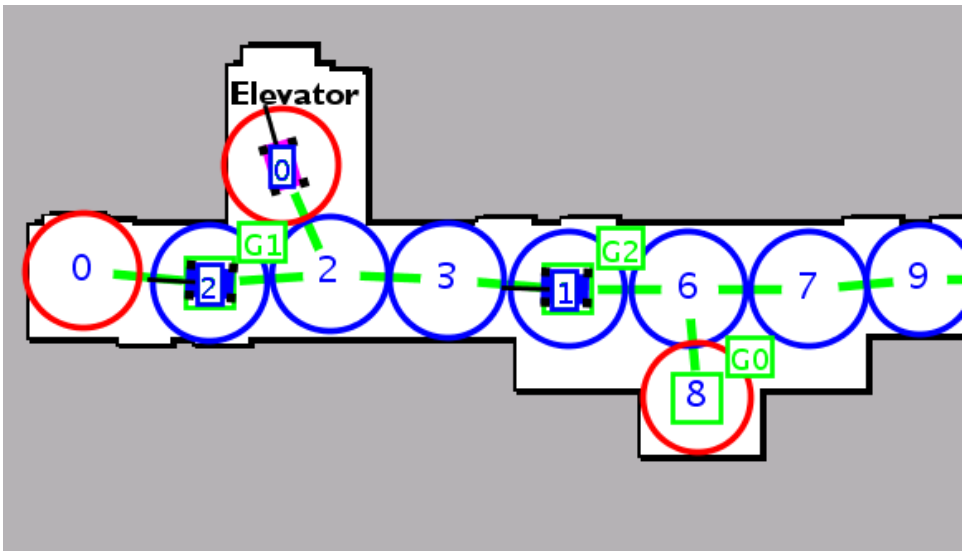


Figure 5.3: Graph representation of hallway environment, showing 9 nodes in the graph and the positions of three robots.

The addition of a time-dependent trajectory following module to the Player robot server allows for scalable simultaneous motion control of many robots from a centralized server. A new multi-robot plan is generated only when the goals change, and the communication to each robot involves only a list of waypoints with arrival and departure times. Current localization and status information is transmitted from each robot to the central server periodically to detect and resolve problems in plan execution (pausing in a corridor to avoid colliding with a person walking past, for example). In such cases, a new plan is generated from the current positions of all robots, and the trajectories are updated.

Figure 5.4 shows an example problem requiring a coordinated solution to demonstrate the implementation. The required transitions are indicated by dotted arrows, moving robots to the goal locations indicated by squares. Robots 1 and 2 are required to swap positions, and robot 0 is required to move from the top to the bottom of the map, crossing the direct paths of robots 1 and 2. Figure 5.5 shows the robots in the hallway environment.

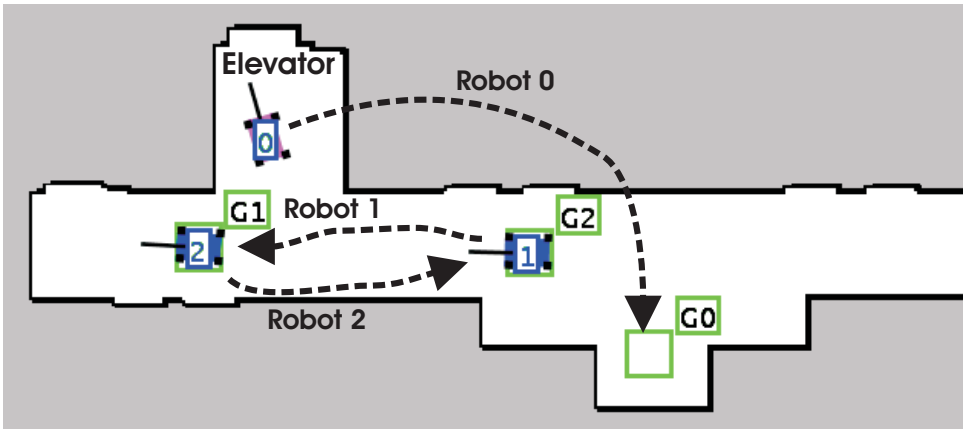


Figure 5.4: Multi-robot planner user interface view of the sample problem

For this problem, a typical decoupled planning approach will fail; if any robot takes the most direct path to its goal, it creates an unavoidable obstacle for another robot. However, the multi-phase algorithm finds a solution by first moving robot 1 to the leaf node 8, then robot 2 to node 7. From this arrangement, all robots reach their goal nodes: first robot 1 to G1; then robot 0 to G0; and finally robot 2 to G2.

The planning algorithm and the trajectory follower have been validated in the corridor environment by randomly allocating new goals as each robot completes its current trajectory. The system was run for continuously for 30 minutes in the experiment. Refer to Appendix B for a video of the robots executing the coordination plan in the hallway environment.

5.5 Task Allocation Implementation

The task allocation system presented in Chapter 4 was demonstrated in the same environment shown in Figure 5.3. Two classes of tasks were defined, a *pick-up* class and a *delivery* task, based on the autonomous mining problem considered in Chapter 4. In this scenario, robots must alternate between *pick-up* task nodes (where material is waiting to be moved out of the mine)

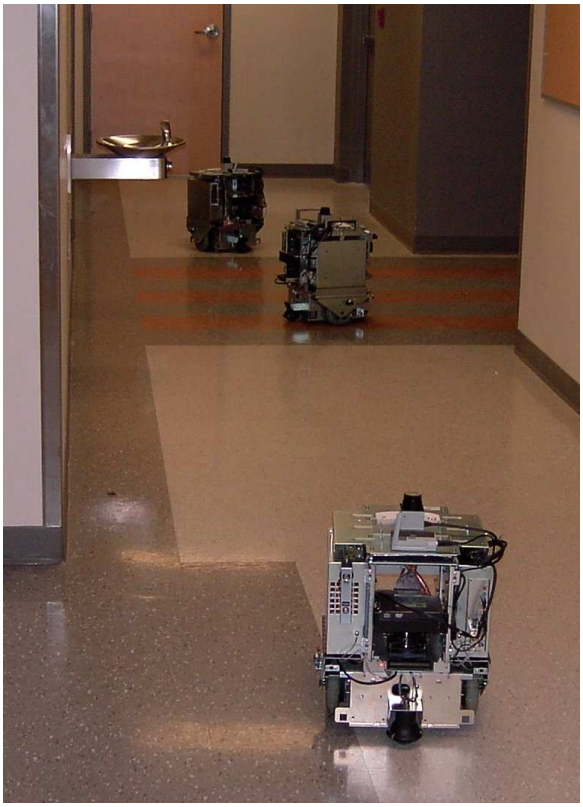


Figure 5.5: Robots executing the solution for the sample multi-robot planning problem

and *delivery* task nodes (where material can be dropped off). The position of the tasks in the map, and the task allocation at a particular instant are shown by a view of the user interface in Figure 5.6.

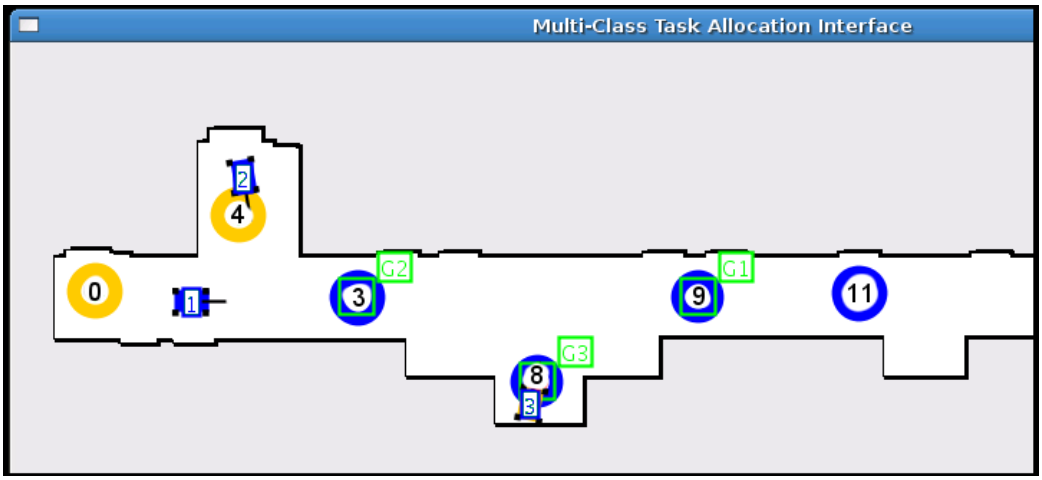


Figure 5.6: Multi-robot multi-class task allocation scenario. The user interface indicates the current position of the three robots (rectangles), two *deliver* task nodes (0 and 4), and 4 *pick-up* nodes (3, 8, 9, and 11). The currently assigned tasks (goals) for the robots are indicated by squares $G1$ at node 9, $G2$ at node 3, and $G3$ at node 8.

For each *pick-up* task, a value representing the number of loads of material available at that node is maintained. The number of loads available is decremented when a robot reaches that node, and is incremented periodically over time; when a task has no loads available to pick up, it is not included in the task allocation process. The multi-class task allocation is accomplished by alternating the class of each robot between *pick-up* and *delivery* as it completes each task, and repeatedly executing the process defined in Section 4.3. Both of the *delivery* tasks are always included in the allocation, on the assumption that there is always capacity for material to be dropped off.

The system architecture for this demonstration is the same as that shown in Figure 5.2 for the multi-phase planning demonstration, with the modifi-

cation that the control application includes a task allocation process. As described in Section 4.3, the task allocation optimization process generates potential allocations (goal assignments for all of the robots), and uses the multi-phase motion planner to generate a solution set of trajectories for each allocation considered. The total distance traveled by all of the robots following the trajectory is used as the objective function to select the best task allocation solution. The corresponding set of trajectories is then passed to the robots through the Player client/server interface.

As the robots are following the assigned trajectories, the task allocation application continues to evaluate different goal assignments based on the current positions of the robots and the state of the tasks. A more efficient task assignment may be found when:

- the optimization process generates a potential solution that was not considered in the previous allocation;
- a robot reaches its assigned goal, and needs to be assigned to a new task;
- a *pick-up* task becomes active (indicating a load of material available to be picked up), and is now included in the allocation process;
- one or more robots have deviated from their assigned trajectories due to obstacles in the environment.

The Player interface and trajectory tracking module allows trajectories to be dynamically updated as new task assignments are generated. The system runs in a continuous loop, updating the trajectories whenever a robot reaches its goal or when a more efficient task allocation is found. Refer to Appendix C for a video of the robots performing the task allocation in the hallway environment.

5.6 Results and Discussion

In the development of this demonstration, two aspects of the implementation that significantly affected the overall system performance were localization and waypoint tracking.

Localization was implemented using the AMCL particle filter module included with Player. Driven by the stepper motor odometry and range measurements from the URG scanning laser rangefinder, the localization system performed very well in tracking position, typically accurate to within 5cm of the actual robot position. However, the system was often unable to globally localize to a correct estimate on initialization, and would require a user to supply an approximate initial position. This was likely due to the lack of distinctive features in the hallway environment, and the limited range (approximately 4m) of the scanning laser. The inability to perform global localization was not a significant issue for this demonstration since a user-supplied position was only required on initialization, after which the tracking performance always maintained an accurate estimate. For systems of many robots, autonomous global localization would be a more significant concern, and a cooperative mechanism such as that proposed in Chapter 2 could be an effective solution.

Trajectory tracking, implemented as a module within the Player architecture, was a key element to the system performance, since the planner assumes that all robots will accurately follow the specified trajectories in order to maintain obstacle-free paths. In the trajectories generated by the multi-phase motion planner, if a plan requires one robot to wait for another to pass by, the first robot will be commanded to wait no longer than absolutely necessary. In the solution in Figure 3.1(a) of Chapter 3 for example, robot $R3$ will be commanded to move to node B as soon as $R2$ should have vacated the node. Any significant lag or position error would result in a depended on low-level obstacle avoidance and frequent re-planning to avoid collisions.

The tracking algorithm described in Section 5.3.3 typically maintained

the robots on their specified trajectories within a tolerance of approximately 10cm. Larger tracking errors, up to 80cm, occurred when a robot was required to stop at a waypoint and rotate to a different heading before continuing on to the next waypoint. The trajectory generated by the multi-phase planner does not include additional time for this maneuver, since the plans are based on topological maps and are independent of orientation. Despite the tracking lag when rotations were required, the tracker was sufficiently accurate that no additional obstacle avoidance was required between robots to avoid collisions, and re-planning was never necessary due to trajectory tracking errors.

5.7 Summary

The planning and task allocations presented in this thesis were demonstrated in a real-world implementation of three robots operating together in a hallway environment, with a central server coordinating their motion. The multi-phase motion planner developed in Chapter 3 was demonstrated by driving robots to assigned goals in a scenario requiring coordination that could not be achieved with a decoupled planner. The task allocation algorithm developed in Chapter 4 was demonstrated with a multi-class task scenario where task allocations are continually updated in real-time, and the robots follow the corresponding collision-free trajectories generated by the multi-phase planner.

Chapter 6

Conclusions

Teams of multiple mobile robots can be effectively applied to numerous applications, such as space exploration, underground mining, and building security. Compared to individual robots, cooperative teams allow for simultaneous presence in multiple locations, improved system performance, faster task execution, and greater system redundancy. However, effective use of multiple robots in a common environment requires coordination and distribution of the planning and control systems used for navigation.

This thesis presented new approaches for three aspects of navigation in the context of multi-robot teams: cooperative localization using inter-robot range measurements; scalable and complete motion planning for large teams of robots; and task allocation for multi-robot teams operating in corridor and tunnel environments.

6.1 Localization

A cooperative method of global localization for mobile robots was developed, based on an trilateration algorithm using range measurements among a triad of robots. A particle filter approach was used to estimate the centroid and orientation of a triangle formed by the triad of robots. Using the sensor data from all of the robots, the absolute pose of the centroid of the triangle

in the environment can be estimated, and the position of each robot can be determined relative to the centroid using the inter-robot range measurements. Simulation results demonstrated the performance of the algorithm with only simple range sensors on each robot, while the number of particles used is varied over a range of relatively small values. Three robots are able to localize themselves in an environment where isolated particle filters on individual robots failed to converge. Cooperative localization of a team of many robots was investigated by extending the triad-based algorithm to sub-teams of three robots. Dynamic selection of the cooperative triads, based on the visibility between robots at any time, makes the system robust to failure of individual robots or the communication network. Using a geometric selection criteria of maximizing the area enclosed by the robots in the triad leads to a decrease in the convergence time as the size of the team increases up to 18 robots in the simulation environment.

6.2 Motion Planning

A new approach was developed to address the problem of finding collision-free trajectories for many robots moving towards individual goals within a common environment. Most popular algorithms for multi-robot planning manage the complexity of the problem by planning trajectories for robots individually; such decoupled methods are not guaranteed to find a solution if one exists. In contrast, this thesis presented a multi-phase approach to the planning problem that uses a graph and spanning tree representation to create and maintain obstacle-free paths through the environment for each robot to reach its goal. The resulting algorithm guarantees a solution for a well-defined number of robots in a common environment.

A description of the algorithm was presented and illustrated using a simple example of a planning problem that cannot be directly solved using a common decoupled approach. Monte Carlo simulation results demonstrated that the multi-phase algorithm is particularly valuable for planning prob-

lems requiring complex coordination, and in scenarios with over 100 robots. The computational cost is shown to be scalable with complexity linear in the number of the robots, and demonstrated by solving the planning problem for 100 robots, simulated in an underground mine environment, in less than 1.5 seconds with a 1.5 GHz processor. The practicality of the algorithm was demonstrated in a real-world application requiring coordinated motion planning of multiple physical robots.

6.3 Task Allocation

Task allocation was considered in this thesis for a specific type of task, where robots are required to visit certain locations (tasks) in their environment, and the environment is composed primarily of narrow corridors or tunnels. This type of problem arises in a number of practical applications, such as building security and autonomous underground mining. The limited working space in these environments requires suitable task allocation combined with coordinated motion planning.

Two task allocation systems were developed and investigated. The first considered *single-class* problems, where the team was considered homogenous, and any robot could be assigned to any task. In this case, an auction-based allocation system was used, capable of finding a set of collision-free trajectories reaching all of the assigned tasks. The algorithm effectively distributes tasks for the patrol-type application among a team of R robots, requiring less than N/R steps between visits, where N is the number of steps required for a single robot to reach all tasks in a circuit.

The second system considered *multi-class* problems, where the team of robots is heterogenous, and some types of tasks may only be assigned to certain types of robots. For such problems, some scenarios required coordinated motion planning to find collision-free trajectories for all robots. To determine a suitable task allocation, an optimization loop was developed, where the task assignment is modified at each iteration, and the fitness of

each assignment is evaluated using the multi-phase motion planner described in Chapter 3.

6.4 Real-World Implementation

To validate the motion planning and task allocation algorithms in a real world application, a system of three WBR-914 PCBot robots was used. Each robot runs the *Player* robot server, and is equipped with a scanning laser rangefinder for localization. A *trajectory tracking* function was added to Player as a plug-in module, to drive the robot through a sequence of time-dependent waypoints as generated by the multi-phase planner. A centralized host PC coordinates the system, communicating with the Player server on each robot via wireless ethernet. Localization and status information is transmitted from each robot to the central server periodically, while the server performs task allocation and coordinated motion planning for the team. This implementation demonstrated the real-time performance and practicality of the algorithms for real-world applications.

6.5 Future Directions

The developments presented in this thesis may be extended by future research in several areas. First, the trilateration-based cooperative localization algorithm could be validated by a hardware implementation, based on a sonar system for generating inter-robot range measurements. An adaptation to the cooperative localization method may also be considered, using an analogous triangulation approach based on the measurement of the bearing, rather than the distance, between pairs of robots.

Future developments of the multi-phase motion planning algorithm, and the related task allocation system, could focus on the generalization of the system to handle additional constraints of different robots, such as maximum speeds and minimum turning radii. Kinematic constraints could be taken

into consideration by embedding geometric features of the environment into a graph representation, as additional properties of nodes and edges. Velocity constraints of different robots could be addressed by varying the size of time steps in the trajectory generation, and constraining the motion of individual robots between steps in the trajectory.

The motion planner system and implementation may be extended to use a distributed architecture. This development would eliminate the requirement of a centralized controller, and allow distribution of the computational effort of trajectory planning across all of the robots in the system.

6.6 Summary

As the number of applications for mobile robots increases, and larger teams of multiple robots are required to work in shared environments, the need for efficient, scalable, and practical algorithms to control their behaviour also grows. The developments presented in this thesis address this need, with novel approaches to three aspects of cooperative navigation for multi-robot teams. The proposed future research directions will build on these algorithms and make them applicable to a wide range of real-world applications.

Appendix A

Planner Animation

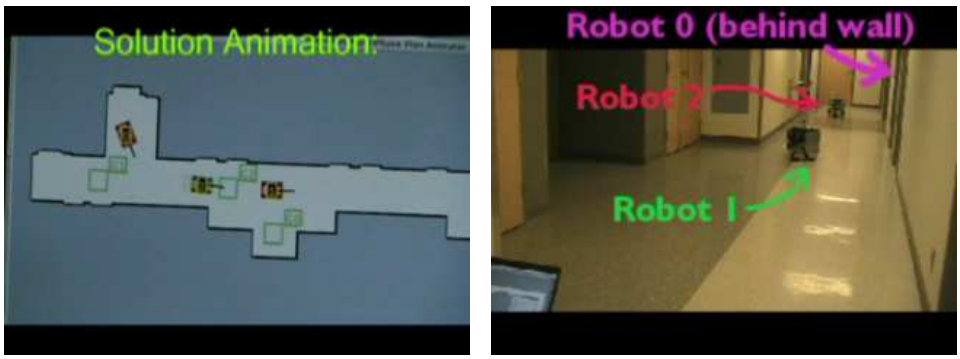
This appendix is an animation video file of the multi-phase planning algorithm presented in Chapter 3. The animation shows the behaviour of the planner in a simulation of 30 robots in a tunnel environment, as described in Section 3.4. The file name of this video is "planner_animation.mpg".

If you accessed this thesis from a source other than the University of Waterloo, you may not have access to this file. You may access it by searching for this thesis at http://uwspace.uwaterloo.ca.

Appendix B

Planner Video

This appendix is a video file of three WBR-914 robots executing plans generated by the multi-phase planning algorithm presented in Chapter 3. The file name of this video is "planner_video.mpg". Figure B.1 shows 2 screen captures from the video.



(a) User interface showing the planned trajectory.

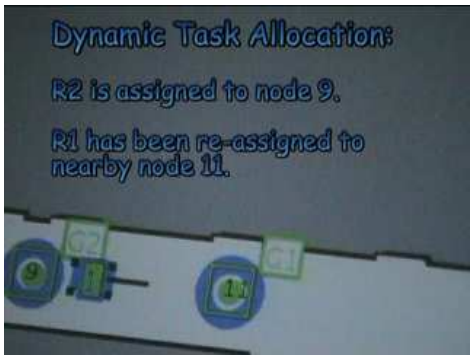
(b) Robots executing the trajectory plan.

Figure B.1: Motion planner video screen captures

Appendix C

Task Allocation Video

This appendix is a video file of the three-robot system implementation presented in Chapter 5, executing the task allocation algorithm presented in Chapter 4. The file name of this video is "task_allocation_video.mpg". Figure C.1 shows 2 screen captures from the video.



(a) The user interface showing dynamic task allocation.

(b) The WBR-914 robots driving to the specified tasks by following the coordinated motion plan trajectories.

Figure C.1: Task allocation video screen captures

If you accessed this thesis from a source other than the University of Waterloo, you may not have access to this file. You may access it by searching for this thesis at http://uwspace.uwaterloo.ca.

Bibliography

[1] R. Alami, F. Robert, F. Ingrand, and S. Suzuki. Multi-robot cooperation through incremental plan-merging. In *ICRA*, pages 2573–2579, 1995.

[2] K. Azarm and G. Schmidt. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3526–3533, 1997.

[3] J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, 1991.

[4] M. Bennewitz, W. Burgard, and S. Thrun. Optimizing schedules for prioritized path planning of multi-robot systems. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 271–276, 2001.

[5] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

[6] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

[7] J. Casey. *A Sequel to the First Six Books of the Elements of Euclid, Containing an Easy Introduction to Modern Geometry with Numerous Examples.* Hodges, Figgis, & Co., 5 edition, 1888.

[8] S. Chien, A. Barrett, T. Estlin, and G. Rabideau. A comparison of coordinated planning methods for cooperating rovers. In *AGENTS '00: Pro-*

ceedings of the fourth international conference on Autonomous agents, pages 100–101, New York, NY, USA, 2000. ACM Press.

[9] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized voronoi graph. In *Workshop on Algorithmic Foundations of Robotics*, 1996.

[10] C. Clark. *Dynamic Robot Networks: A Coordination Platform for Multi-Robot Systems*. PhD thesis, Stanford University, 2004.

[11] C. Clark and S. Rock. Randomized motion planning for groups of nonholonomic robots. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2001.

[12] M. A. Crespo, J. M. Caas, and V. Matellan. Comparing bayesian and montecarlo localization for a robot with local vision. In *Proceedings of the 2003 Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, pages 171–174, 2003.

[13] M. B. Dias and A. T. Stentz. Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. Technical Report CMU-RI -TR-03-19, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2003.

[14] M. B. Dias, R. M. Zlot, N. Kalra, and A. T. Stentz. Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, April 2005.

[15] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[16] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[17] D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[18] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI/IAAI*, 1999.

[19] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325 – 344, 2000.

[20] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, 2001.

[21] S. Ge and Y. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 1997.

[22] B. Gerkey and M. Mataric. *Experimental Robotics VII, LNCIS 271, edited by D. Rus and S. Singh*, chapter Principled Communication for Dynamic Multi-Robot Task Allocation, pages 353–362. Springer-Verlag Berlin Heidelberg, 2001.

[23] B. Gerkey and M. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[24] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *ICAR 2003*, pages 317–323, Coimbra, Portugal, June 2003.

[25] V. Gervasi and G. Prencipe. Coordination without communication: the case of the flocking problem. *Discrete Appl. Math.*, 144(3):324–344, 2004.

[26] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. T. Stentz. Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, volume 2, pages 27–38. Kluwer Academic Publishers, 2003.

[27] R. Grabowski and P. Khosla. Localization techniques for a team of small robots. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 2001.

[28] R. Grabowski, L. Navarro-Serment, C. Paredis, and P. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots - Special Issue on Heterogeneous Multirobot Systems*, 1999.

[29] Y. Guo and L. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2612–2619, 2002.

[30] J. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods continued. In *IEEE International Conference on Intelligent Robots and System (IROS)*, 2002.

[31] J.-S. Gutmann, T. Weigel, and B. Nebel. Fast, accurate, and robust self-localization in polygonal environments. In *IEEE International Conference on Intelligent Robots and System (IROS)*, 1999.

[32] E. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on SSC*, 4, 1968.

[33] Y. Hidaka, A. Mourikis, and S. Roumeliotis. Optimal formations for cooperative localization of mobile robots. In *IEEE International Conference on Robotics and Automation*, 2005.

[34] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

[35] A. Howard, M. J. Matarić, and G. S. Sukhatme. Putting the 'I' in 'Team': An Ego-Centric Approach to Cooperative Localization. In *IEEE International Conference on Robotics and Automation*, pages 868–892, Taipei, Taiwan, Sep 2003.

[36] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, 1997.

[37] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[38] K. Kant and S. W. Zucker. Toward efficient trajectory planning: the path-velocity decomposition. *Int. J. Rob. Res.*, 5(3):72–89, 1986.

[39] K. Kato, H. Ishiguro, and M. Barth. Identifying and localizing robots in a multi-robot system environment. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 1999.

[40] L. Kavraki and J. Latombe. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, chapter Probabilistic Roadmaps for Robot Path Planning, pages 33–53. John Wiley, 1998.

[41] K. Konolige, C. Ortiz, R. Vincent, B. Morisset, A. Agno, M. Eriksen, D. Fox, B. Limketkai, J. Ko, B. Stewart, and D. Schulz. Centibots: Very large scale distributed robotic teams. In *IFIP Congress Topical Sessions*, page 761, 2004.

[42] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[43] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[44] E. L. Lawler, J. K. Lenstra, A. H. G. R. Khan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.

[45] M. A. Lewis and G. A. Bekey. The behavioral self-organization of nanorobots using local rules. In *IEEE/RSJ IROS*, page 13331338, 1992.

[46] H.-I. Lu and R. Ravi. A fast approximation algorithm for maximum-leaf spanning tree. In *ISPAN '97: Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '97)*, page 351, Washington, DC, USA, 1997. IEEE Computer Society.

[47] V. Lumelsky and K. Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1):121–135, 1997.

[48] R. Madhavan, K. Fregene, and L. Parker. Distributed heterogeneous outdoor multi-robot localization. In *IEEE Int. Conf. on Robotics and Automation*, pages 374–381, May 2002.

[49] R. Madhavan, K. Fregene, and L. E. Parker. Distributed cooperative outdoor multirobot localization and mapping. *Autonomous Robots*, 17:23 – 39, July 2004.

[50] M. Mataric, G. S. Sukhatme, and E. H. stergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14:255–263, 2003.

[51] D. McFarland. Towards robot cooperation. In *Simulation of Adaptive Behaviour*, 1994.

[52] J. McLurkin. Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots. Master's thesis, M.I.T., 2004.

[53] J. McLurkin. Distributed algorithms for multi-robot systems. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 545–546, New York, NY, USA, 2007. ACM Press.

[54] H. Moravec and A. Elfes. High-resolution maps from wide-angle sonar. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 1985.

[55] A. Mourikis and S. Roumeliotis. Optimal sensing strategies for mobile robot formations: Resource constrained localization. In *Robotics: Science and Systems Conference*, 2005.

[56] L. Navarro-Serment, C. Paredis, and P. Khosla. A beacon system for the localization of distributed robotic teams. In *Proceedings of the International Conference on Field and Service Robotics (FSR '99)*, 1999.

[57] L. E. Navarro-Serment, R. Grabowski, C. Paredis, and P. Khosla. Millibots. *IEEE Robotics and Automation*, pages 31 – 40, December 2002.

[58] R. R. Negenborn. Kalman filters and robot localization. Master's thesis, Institute of Information and Computer Science, Utrecht University, Utrecht, Netherlands, 2003.

[59] F. R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: application to indoor environment. *Int. J. Rob. Res.*, 12(1):79–98, 1993.

[60] P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 484–489, 1989.

[61] C. H. Papadimitriou and K. Steiglitz. Some complexity results for the traveling salesman problem. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1976. ACM Press.

[62] L. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[63] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *Int. J. Rob. Res.*, 24(4):295–310, 2005.

[64] N. B. Priyantha. *The Cricket Indoor Location System.* PhD thesis, Massachusetts Institute of Technology, June 2005.

[65] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *6th ACM MOBICOM*, 2000.

[66] I. M. Rekleitis, G. Dudek, and E. Milios. Multi-robot collaboration for robust exploration. In *IEEE Int. Conf. in Robotics and Automation*, pages 3164–3169, San Francisco, USA, April 2000.

[67] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM Press.

[68] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, and II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[69] S. I. Roumeliotis and G. A. Bekey. Distributed multi-robot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, 2002.

[70] M. R. K. Ryan. Graph decomposition for efficient multi-robot path planning. In *IJCAI*, pages 2003–2008, 2007.

[71] A. Rynn, W. A. Malik, and S. Lee. Sensor based localization for multiple mobile robots using virtual links. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 2003.

[72] S. Sariel and T. Balch. Efficient bids on task allocation for multi robot exploration. In *The 19th International Florida Artificial Intelligence Research Society (FLAIRS) Conference*, 2006.

[73] T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics & Automation*, 18(1), Feb. 2002.

[74] R. G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *AAAI/IAAI*, pages 852–858, 2000.

[75] R. Smith, M. Self, and P. Cheeseman. *Estimating uncertain spatial relationships in robotics*, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[76] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12):1104–1113, 1980.

[77] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 4, pages 3310–3317, May 1994.

[78] P. Svestka and M. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998.

[79] J. D. Sweeney, H. Li, R. A. Grupen, and K. Ramamritham. Scalability and schedulability in large, coordinated, distributed robot systems. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2003.

[80] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[81] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[82] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[83] UNECE. 2004 world robotics survey. Technical report, United Nations Economic Commission for Europe, 2004.