Mixed Modes of Autonomy for Scalable Communication and Control of Multi-Robot Systems

By

John Paul Bird

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Mechanical Engineering

Alfred L. Wicks, Chair

Charles F. Reinholtz

Andrew J. Kurdila

Daniel J. Stilwell

Craig A. Woolsey

September 26, 2011

Blacksburg, VA

Keywords: Multi-robot systems, Coordinated Control, Autonomous Transportation

Mixed Modes of Autonomy for Scalable Communication and Control of Multi-Robot Systems

John Paul Bird

Abstract

Multi-robot systems (MRS) offer many performance benefits over single robots for tasks that can be completed by one robot. They offer potential redundancies to the system to improve robustness and allow tasks to be completed in parallel. These benefits, however, can be quickly offset by losses in productivity from diminishing returns caused by interference between robots and communication problems. This dissertation developed and evaluated MRS control architectures to solve the dynamic multi-robot autonomous routing problem. Dynamic multi-robot autonomous routing requires robots to complete a trip from their initial location at the time of task allocation to an assigned destination. The primary concern for the control architectures was how well the communication requirements and overall system performance scaled as the number of robots in the MRS got larger. The primary metrics for evaluation of the controller were the effective robot usage rate and the bandwidth usage.

This dissertation evaluated several different approaches to solving dynamic multi-robot autonomous routing. The first three methods were based off of common MRS coordination approaches from previous research. These three control architectures with distributed control without communication (a swarm-like method), distributed control with communication, and centralized control. An additional architecture was developed to solve the problem in a way that scales better as the number of robots increase. This architecture, mixed mode autonomy, combines the strengths of distributed control with communication and centralized control. Like distributed control with communication, mixed mode autonomy's performance degrades gracefully with communication failures and is not dependent on a single controller. Like centralized control, there is oversight from a central controller to ensure repeatable high performance of the system. Each of the controllers other than distributed control without communication is based on building world models to facilitate coordination of the routes. A second variant of mixed mode autonomy was developed to allow robots to share parts of their world models with their peers when their models were incomplete or outdated.

The system performance was evaluated for three example applications that represent different cases of dynamic multi-robot autonomous routing. These example applications were the automation of open pit mines, container terminals, and warehouses. The effective robot usage rates for mixed mode autonomy were generally significantly higher than the other controllers with a higher numbers of robots. The bandwidth usage was also much lower. These performance trends were also observed across a wide range of operating conditions for dynamic multi-robot autonomous routing.

The original contributions from this work were the development of a new MRS control architecture, development of system model for the dynamic multi-robot autonomous routing problem, and identification of the tradeoffs for MRS design for the dynamic multi-robot autonomous routing problem.

Acknowledgements

I would like to thank all of the people that have made this possible. With as many years as this has taken, I am sure that I am going to leave some people out so I apologize in advance. All of this starts with the support my family has given my throughout my education. They've supported me each step of the way. No words that I can write here can really express how grateful I am.

To Dr. Wicks, thank you for all the opportunities I have had to continue to develop personally and professionally. My PhD has been about so much more than the dissertation research. I've enjoyed all the assorted projects and teaching and I wouldn't be who I am without that experience. The challenges along the way are what have made this worth doing.

I would like to thank the many wonderful people I have shared lab and office space with over the years. You have all made coming in to work each day much more fun and rewarding than you probably realized. I especially want to thank Patrick Currier and Jonathan Gaines (even if you guys did beat me to the PhD by a little bit).

Table of Contents

List of Figures

List of Tables

Chapter 1 – Introduction

Autonomous robots are increasingly used as multi-robot systems (MRS). MRS tasks can be grouped into two types – parallel tasks that can be completed by single robot and coupled tasks that require multiple robots to complete [**1**]. Common parallel tasks for multi-robot systems include transportation, search, and mapping. Each robot can usually complete their task independently of the rest of the robot team, but coordination between the robots is often required. The MRS must allocate tasks to individual robots and handle resource conflicts to ensure additional robots add to the total tasks performed by the system. Coupled tasks typically involve cooperative manipulation (like the box moving problem) or formation control (flocking, etc.)[**2**]. These tasks cannot be completed by single robots. The work in this dissertation focuses on parallel tasks, specifically autonomous coordinated routing for transportation.

The parallel operation of the robots protects the system against the failure of single robot and increases the speed at which a set of tasks can be accomplished. For MRS, the best case scenario is that N robots complete the tasks in 1/N of the time taken by a single robot. The advantages of multiple robots are often not fully realized due to issues with coordination and control. Most of the existing work in this area is focused on optimal task allocation. If the plans developed in optimal task allocation can be followed, the system should maximize the efficiency of the system. This task allocation work is a good starting point for efficient multi-robot systems, but two issues arise in many practical systems that can significantly reduce the system efficiency – highly dynamic or uncertain operating conditions and dynamic task generation. Since optimal allocations are made based on current and projected system states, highly dynamic operating conditions can prevent the tasks from being executed as planned. In these cases, two sets of options are available – replanning the task allocation or accepting possible suboptimal task execution. In many practical systems replanning is not an option due to communication bandwidth limitations or other constraints. Dynamic task generation also limits the effectiveness of optimal allocation algorithms. Kalyanasundaram and Pruhs have shown that online assignment at the lower bound has a performance three times as inefficient as optimal allocation [**3**]. To address these situations when optimal task allocation does not sufficiently address the multi-robot system control problem, a method for improving task execution is needed. This work focused on the development of a control framework for the execution and coordination of tasks in an uncertain environment with limited communication.

This chapter defines the key issues involved with coordination of robot teams for autonomous transportation, states the multi-robot coordination case study problem – dynamic multi-robot autonomous routing; presents the example applications – automation of open pit mines, container terminals, and warehouse; establishes the original contributions of this work; and presents an overview of the rest of the document.

1.1 Challenges for Multi-Robot Coordination in Large Scale Teams

Scalability of control algorithms and communication must be addressed to successfully implement a large scale autonomous multi-robot system. As the total number of robots grows the system will eventually reach a point of diminishing return due to congestion related conflicts, insufficient bandwidth, or message latency. One of the primary assumptions that much of the

previous work in this area makes is that all communication is ideal. There are no limitations placed on the communication range or data rate and all transmitted data is successfully received. This is highly unrealistic in most real environments. As the number of robots increase the limitations on communication can become significant. With ideal communications the bandwidth requirements scale linearly with the number of robots for most control types. In real networks, message retransmission can increase the bandwidth needed even more. This bandwidth requirement can quickly grow even larger when a mesh network topology is used, since many of the messages will need to be rebroadcast. Latency also becomes an issue as the number of robots and thus messages increase. This work addresses this issue by with a new method of multi-robot control – Mixed Mode Autonomy (MMA).

1.2 Problem Statement

A wide variety of multi-robot control problems can be used to study the scalability of MRS [**2**]. These control problems typically involve three parts – allocation, planning, and execution. Allocation is the assignment of tasks to each robot. Robots can either be committed to a task until completion or the tasks can be dynamically reassigned as the system state changes. Much of the work done in multi-robot control is on optimal task allocation. Planning is the development of a sequence of operations to complete the assigned tasks. Execution is the implementation of the plan. The emphasis of this work is to improve the planning and execution of multi-robot control tasks in uncertain environments. Dynamic Autonomous Multi-Robot Routing is used as the case study problem for this work.

Dynamic Multi-Robot Autonomous Routing

The multi-robot task is to complete a trip from a robot's location at the time of task allocation to an assigned destination while following specified limits on speed and separation distance between robots. Each of the robots is assigned a destination by online assignment. The system must plan and execute a route to get the robot to its destination. A route is found for each robot when a destination is allocated to a robot. The route is reevaluated at each intersection where a potential turn could be made. The uncertainties involved in each robot's execution of their route causes changes in the system state that may make it more efficient to change the route. After a robot reaches its destination, it is assigned another destination. Figure 1-1 shows a typical task for a single robot in a simplified environment. "R1" is robot 1's current location in the environment. "D1" is robot 1's destination. The task is to travel from R1 to D1. The route is highlighted. At each intersection (marked a, b, c, d, and e), the route is reevaluated. The route will not change unless there is a clear reduction in the potential congestion. Congestion is when too many robots are in one area to maintain the separation distances without pausing any of the robots.

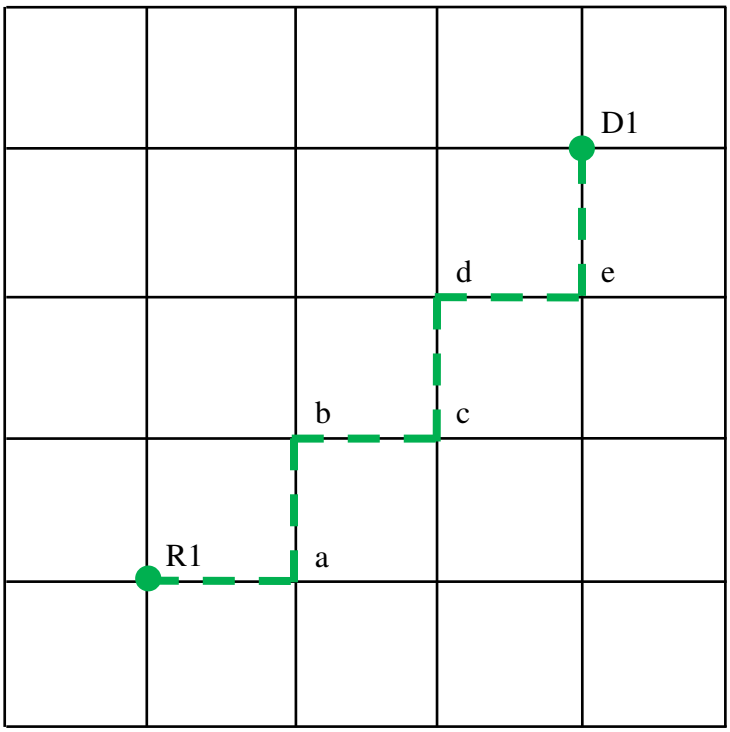


Figure 1-1. Example Dynamic Multi-Robot Autonomous Routing Task

Dynamic multi-robot autonomous routing seeks to solve the routing problem to attempt to maximize the system throughput/ minimize travel times with a dynamic system and environment with communication constraints.

Multi-Robot Environment

The multi-robot environment can be represented as a network of roads. The network of roads is a set of vertices connected by road segments. All robot motion is constrained to be along road segments. When robots are on the same road segment they must maintain a safe minimum separation distance. The road segments are directional (a two-lane road would have two road segments in opposite directions connecting the two vertices). Robots capabilities are defined by their maximum velocity, sensor characteristics, and communication settings.

For this work, the environments are a uniform two dimensional grid. Two parameters define the grid – grid shape and bottleneck width. Grid shape is the ratio of the lengths of road segments. Bottleneck width is the minimum number of vertices at the narrowest point in the environment. A narrow bottleneck is a possible congestion point in the environment. Figure 1-2 shows an example environment. The simulations of the dynamic multi-robot autonomous routing problem are run for different numbers of robots for each environment and communication settings.

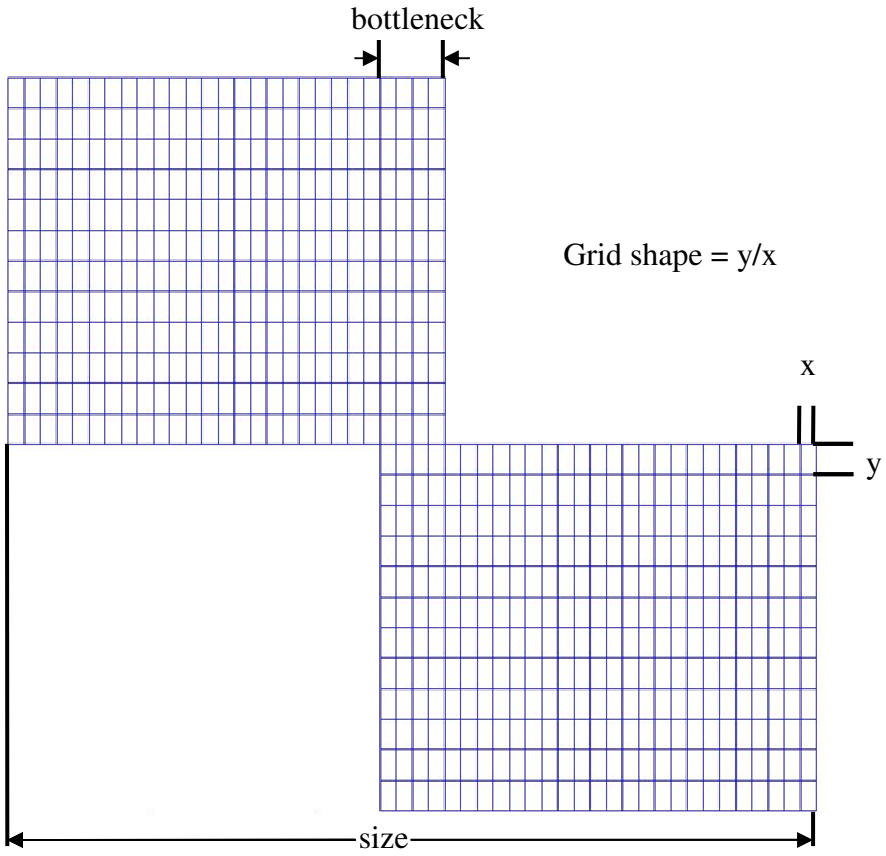


Figure 1-2. Example Environment

Communication Considerations

The simulations to evaluate controllers to solve the dynamic multi-robot autonomous routing problem do not assume ideal communications. Both packet errors and communication latency can have a significant impact on controller performance. A stochastic model of the communication network is used to calculate packet error rates (PER) as a function of distance, network parameters, and environment properties. Channel availability is also modeled to include the latency effects. The communication model is presented in detail in Chapter 5.

Formal Definition of the Dynamic Multi-Robot Autonomous Routing Problem

This section presents the formal mathematical definition of the control problem and the related metrics of evaluation. Dynamic multi-robot autonomous routing is a general formulation of many of the routing problems in previous research[**4**][**5**][**6**][**7**].

Multi-Robot System

The multi-robot system (MRS) is made up of N robots. Robot i has a maximum velocity V_i. Each robot has a set of sensors to detect objects or other robots in its path. These sensors are

modeled to have a false positive error rate of E_i. Robots are required to maintain a separation distance F when they are on the same road network. A robot's current position is $P_i(x,y)$.

Road Network

The road network is made up of segments connecting a set of vertices. The road network can be represented as a weighted adjacency matrix W. $W(v_1, v_2)$ is the distance from vertex 1 to vertex 2. If $W(v_1, v_2)$ is infinity it means that there is no connection between the two vertices. For this work, the vertices are distributed over a uniform square grid. In the bottlenecked environment not all of the vertices are connected. This generalization simplifies the calculations for the simulations, which allows the simulations to be completed more quickly. The road networks are described by three parameters – network size, grid shape, and bottleneck width. The network size, M, is the number of vertices along one side of the square. The grid shape, g, is the ratio of the segment lengths in the x and y directions. The bottleneck width, b, is the number of segments wide the narrowest portion of the network. These parameters allow the road network used in simulations to represent the environments of the example applications.

Task Assignment

The robots' task is to complete trips to their assigned destinations. An unassigned task is T(v), a trip to vertex v. An assigned task is T(v, i), a trip to vertex v assigned to robot i. The tasks are allocated by online assignment. The spatial distribution of the tasks is set by the coupling, C_t. The coupling is the standard distribution of the distances of the tasks from the task center. The two task centers are located in the center of the opposite quadrants of the road network. Consecutive tasks are distributed around opposite task centers. A low coupling value represents tasks that are tightly grouped. Tasks with coupling values approaching infinity are effectively uniformly distributed throughout the road network. The time to complete a task is t(T(v, i)).

Routes

When a robot is assigned a task, it needs a route from its current location, its previous task $T_{-1}(v)$, and its current task, T(v). Within the road network many possible routes to the assigned destination can be found. The j^{th} possible route for robot i is $r_{i,j}(S, v1, v2, \ldots, T(v, i))$. S is the start of the route – the current location of the robot. The travel cost of the route j is $D(r_{i,j})$. For the nominal case, the travel cost is the length of the route. The selected route for robot i is $R_i(S, v1, v2, \ldots, T(v, i))$.

Communication

Most of the control systems used in this work rely on sharing information between robots and a central controller and/or their peers. A model of a realistic wireless communication system is used for the simulations. The primary two effects included in the model are the Packet Error Rate (PER) and the wireless communication channel availability. This allows the controller simulations to include the effects of dropping packets of information and the latency in communication. The network parameters are data rate (B_{total}), transmit power (P_t), and packet overhead rate (O). The message length, L is specific to the controller. The full communication model is presented in Chapter 4.

Metrics

A set of metrics are used to measure the performance of the system to compare the different controllers used for dynamic multi-robot autonomous routing. Effective Usage Rate (U) is a measure of how efficiently a robot completes a task. It is the ratio of the ideal time to complete a task to the actual time the task took to complete. The controller goal is to push this number for the system as close to one as possible.

$$U_i = \frac{D(shortest\ path\ for\ T(v,i)\ ) * V_i}{t(\ T(v,i)\ )}$$

The primary controller goal is for the MRS to complete as many tasks as possible during a given interval in time. The measure of controller performance is the MRS effective usage rate U_{total}.

$$U_{total} = \sum_{all\ tasks} \frac{\sum_{i=1}^{N} D(shortest\ path\ for\ T(v,i)\ ) * V_i}{\sum_{i-1}^{N} t(\ T(v,i)\ )}$$

Safe operation is also a critical aspect of MRS performance. The system tracks the number of near misses (NM). These are instances when both the planning and sensing failed to prevent two (or more) robots from coming within the minimum separation distance.

$$NM = \sum_{all\ tasks} \frac{Near\ Misses\ for\ task}{N}$$

Bandwidth Used (B_{used}) is a measure of how much of the capacity of the communication network is used.

$$B_{used} = \sum_{t=0} \sum_{i=1}^{N} (L_i + O_i)$$

Many other variables and metrics are tracked or calculated with the simulation to illustrate why there are differences between different controllers and parameters. These metrics are presented in Chapters 4 and 5.

1.3 Example Applications

By varying the environment and robot parameters many possible applications can be represented as dynamic multi-robot autonomous routing problems. Some example applications include automated warehouses, container terminal automation, and open pit mining.

Automated Warehouses

Warehouse automation is a potential application for multi-robot systems. Robots can be used to move goods throughout the warehouse. Using robots could improve the rate at which orders are processed and shipments are unloaded while reducing the number of people required to run the

warehouse. Several companies have already started to develop robot systems to automate warehouse [**8**][**9**]. Figure 1-3 shows an example of warehouse tasks. Warehouse automation can be represented as a dynamic multi-robot autonomous routing problem. The warehouse environment can be characterized as a uniform grid with no bottleneck. The tasks are usually tightly coupled at one end (loading dock) and loosely coupled at the other (shelves/storage area). The number of robots using in a MRS for warehouse automation could be quite large – potentially reaching or exceeding 500 robots.

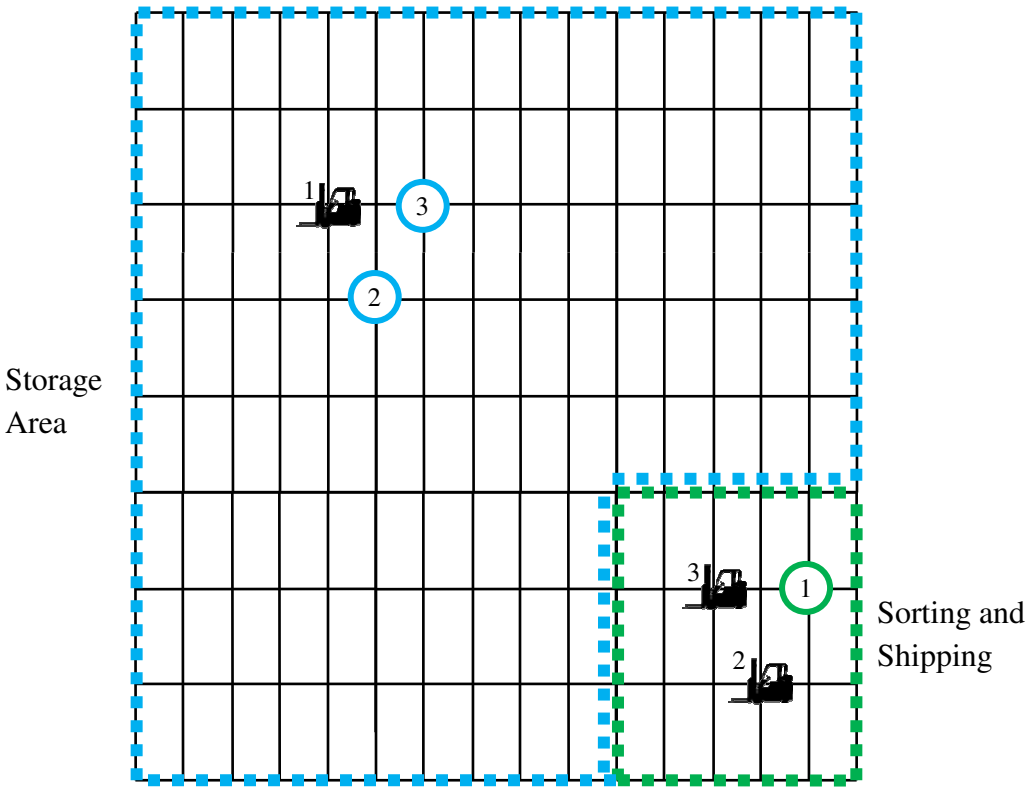


Figure 1-3. Warehouse tasks for MRS automation

Container Terminals

Loading and unloading cargo containers onto trains and ships can be automated by making the container terminal transport vehicles autonomous. Container terminal systems already have well developed dispatchers systems that are optimal or near optimal for task allocation with the transport vehicles. Solving the dynamic multi-robot autonomous routing problem is a way to handle the uncertainties and improve the execution of these tasks with an autonomous MRS. The typical container terminal tasks are shown in Figure 1-4. The container terminal can be represented as a uniform grid with a narrow bottleneck. The tasks are tightly coupled on the ship or train end and loosely coupled in the container storage yard.

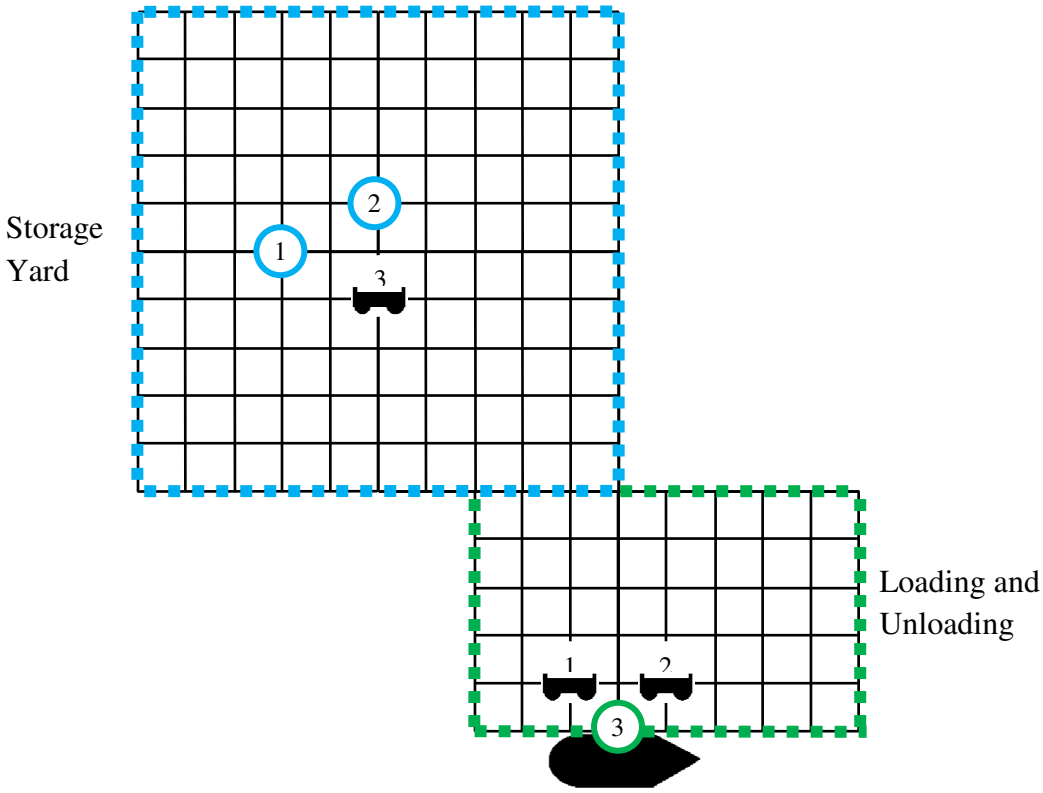


Figure 1-4. Container terminal environment and tasks

Open Pit Mines

Driving haul trucks in open pit mines is both dangerous and dull, making it a potential application for robots. The tasks for an open pit mine are shown in Figure 1-5. Automation of open pit mines can be seen as an extension of the dynamic multi-robot autonomous routing problem. The environment is a uniform grid with a narrow bottleneck. Unlike the other environments the narrow section is much longer. The tasks are usually tightly coupled on both ends. The simulations used in this work do not directly model an environment like this; however, the results from this work can be scaled to approximate the open pit mine style environments. This scaling is presented in Chapter 6.

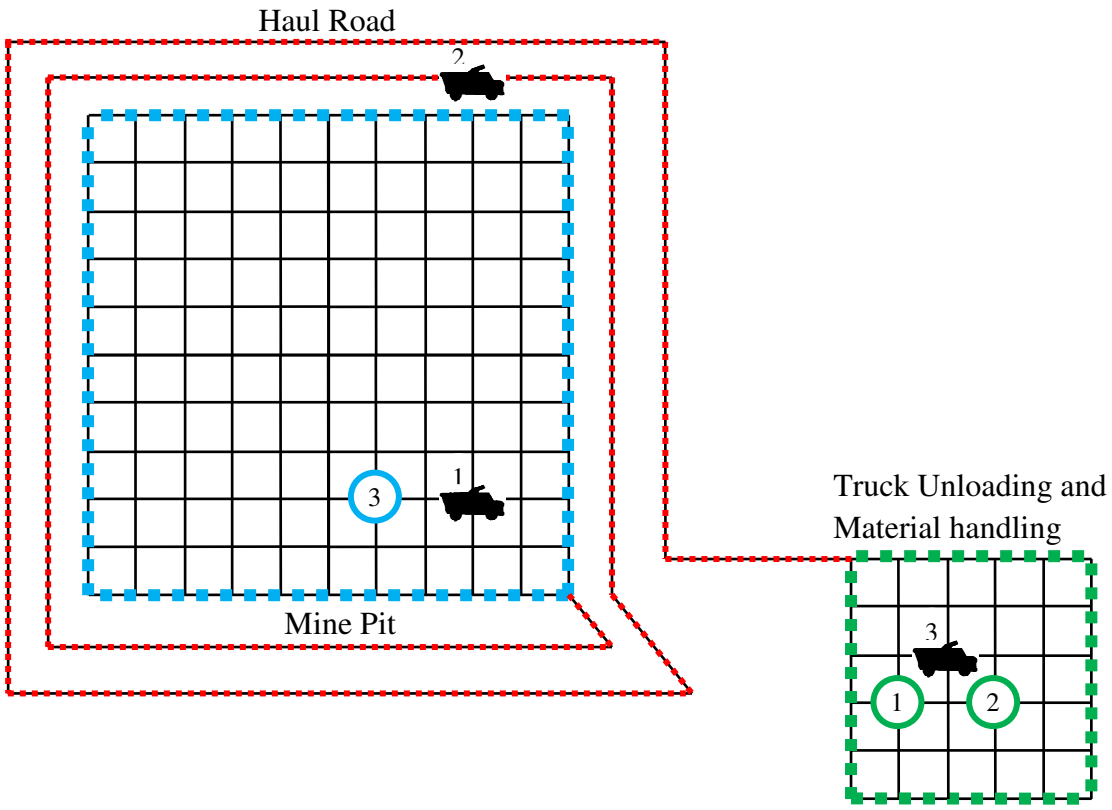


Figure 1-5. Open pit mine environment and tasks

1.4 Control Approaches for Dynamic Multi-Robot Autonomous Routing

There were five control architectures studied for the dynamic multi-robot autonomous routing problem. They were centralized control, distributed control without communication, distributed control with communication, mixed mode autonomy, and mixed mode autonomy with model updates. The first three represent the typical approaches from previous MRS research. The two forms of mixed mode autonomy are new architectures for coordination of MRS. These controllers are summarized briefly here and are reviewed in detail in Chapter 3.

Distributed Control without Communication

This is the simplest controller used. It is based on the emergent behaviors seen in swarm robotics. All of the control is based on sensor inputs. Distributed control without communication is the lower baseline of performance for the system.

Distributed Control with Communication

Distributed control with communication is a more structure distributed control architecture. It is based on the robots each maintaining a world model of the states of all of the robots in the MRS. Each robot shares its information so the other robots can update their world models.

Centralized Control

Centralized control uses a single controller to coordinate the actions of the entire MRS. All of the robots send status updates to the central controller. The centralized controller maintains a world model and sends commands to the robots.

Mixed Mode Autonomy

Mixed mode autonomy is the new controller developed for this dissertation. It combines a distributed controller with oversight from a centralized controller. Each of the robots shares information with each other and the centralized controller to maintain world models. The robots each make their own control decisions and share these as part of their updates. The centralized controller then determines if it can find a better solution. If it can it sends a command to override the robot. Mixed mode autonomy also includes a communication management system to prioritize communication to reduce bandwidth usage. Mixed mode autonomy with model updates allows robots to share parts of their world models other than their own information to help keep the other robots in the MRS current.

1.5 Contribution to the Field from this Dissertation

This dissertation presents a new multi-robot control architecture that was developed to scale well for large scale multi-robot systems. This novel architecture, mixed mode autonomy, combines elements of centralized and distributed control to achieve high performance and reduce the communication requirement per robot. The reduced communication requirements are key to the scalability of the control architecture for larger multi-robot systems.

The performance of the mixed mode autonomy architecture was evaluated by solving the dynamic multi-robot autonomous routing problem. The performance was compared to implementations of more traditional centralized and distributed multi-robot system control architectures. To assess the performance of Mixed Mode Autonomy and the other controllers, dynamic multi-robot autonomous routing was simulated with large scale multi-robot systems. To ensure a meaningful evaluation, the simulations needed to include enough of the real effects to approximate the real world performance. The high level effects needed to be incorporated while abstracting less significant details to keep the simulations computationally manageable. The robot mobility and motion control is represented as incremental movements on fixed paths (the roads). The sensors for collision avoidance are simplified to false positive and false negative rates. These simplifications keep the simulation simple enough to run many iterations while preserving the effects that influence multi-robot control at a high level.

For multi-robot control, the reliability of the information shared through wireless communication can have a significant impact on the performance. Unfortunately, many multi-robot system control simulation assume ideal (or near ideal) communications (no packet losses or latency). One of the key differences between this work and much of the preceding work is that the simulation models include a realistic model of both packet loss and latency of messages.

The results of the comparison of the multi-robot control architectures for dynamic multi-robot autonomous routing can be used to make recommendations about the appropriate controllers for

different applications or scenarios. The applications can be represented by a set of environment, task distribution, and communication parameters. The simulation parameters used are close to several warehouse automation and container terminal automation problems. With some extensions the simulation parameters can also possibly represent open pit mining problems.

1.5 Outline of the Rest of this Document

The remainder of this dissertation is structured as follows.

Chapter 2 – Literature Review summarizes the background on previous work in this area and the underlying theories used as a foundation to this work.
Chapter 3 – Coordination and Control Strategies presents the control architectures used for solution of dynamic multi-robot autonomous routing.
Chapter 4 – Multi-Robot Simulation Model presents the mathematical equations used to represent the problem domain, robot behaviors, and controllers for the Monte Carlo simulations.
Chapter 5 – Simulation Results and Discussion shows the simulation results for the three example applications.
Chapter 6 – Conclusions and Recommendations for Future Work discusses the key findings of the work and recommendations for how to best continue it.

Chapter 2 – Literature Review

The literature review focuses on methods of coordination, work done to solve problems similar to dynamic multi-robot autonomous routing, communication for multi-robot systems, and an overview of the example application problems.

Methods of coordination presents the ways multi-robot systems can be made to work together. The multi-robot coordination section discusses the two aspects of the problem – task allocation and task execution. The methods of coordination are generally divided between centralized and distributed control. These control approaches are the basis of the traditional methods that the novel control architecture developed for this work, to which mixed mode autonomy is compared.

The work done to solve problems like the dynamic multi-robot autonomous routing is summarized. Much of this work focuses on allocation of the robots within the MRS. The work for this dissertation is to improve the task execution in dynamic multi-robot autonomous routing.

The communication section presents the underlying concepts of RF propagation and modulation needed to model the wireless communication needed to support coordination. The justification for the assumptions used for the communication model is presented.

The three example problems are automation of open pit mines, automation of container terminals, and automation of warehouses. The applications are explained and past work towards automation of these problems are summarized.

2.1 Motivation for this Work

Multi-robot systems (MRS) are becoming increasingly common. To effectively achieve the desired tasks proper coordination of the MRS is necessary. The reasons for coordination include resource allocation, conflict elimination or resolution, achieving goals that require more than one robot, and improving efficiency [**10**]. For the dynamic multi-robot autonomous routing problem eliminating or resolving conflicts and improving efficiency are most relevant. Conflicts can be reduced by routing the robots away from potential congestion. Less congestion and improved use of the communication bandwidth can improve the overall efficiency of the MRS. The primary problems that need to be address in design of a controller for an MRS are development of scalable control laws with limited communication bandwidth, evaluation of effect of time delays for the controller, and to generalize the controller for complex systems and environments [**11**]. The mixed mode autonomy MRS control architecture introduced in this work improves the scalability of the bandwidth requirements over many of the traditional methods of coordination discussed in the next section. The communication model used for this work makes it possible to simulate and evaluate the effect of the communication delays in real systems. Networked robotics is one of the major emerging areas in cooperative robotics [**12**]. This area of research builds off of the earlier work on swarms (multi-robot systems many without communication) and sensor networks[**12**]. The previous research in multi-robot systems indicated the need for more scalable control architectures and more realistic simulation for evaluation. The rest of this literature review presents the methods/architectures used for coordination of MRS, the algorithms for routing multiple robots (the specific MRS coordination problem for this work), an overview of the communication modeling, and an explanation of the example applications.

2.2 Methods of Coordination

This section describes the ways robots can be made to work with each other for a common goal. It presents the key dimensions of classification for the approaches to control of multiple robots. The tasks for a MRS are one of two types – single robot tasks or multiple robot tasks [**1**]. Single robot tasks can be completed by one robot, independent of other robots. Multiple robot tasks require cooperation between robots to complete the task. The tasks for dynamic multi-robot autonomous routing are single robot tasks. The rest of this literature review concentrates on methods of coordination that are relevant for single robot tasks.

Parker does a good job classifying the different approaches to control of MRS. They are broadly grouped into two categories swarm systems and intentionally cooperative systems [**2**]. Swarm systems are generally homogenous robots with simple rules. The cooperative behavior emerges from these simple rules. For this work distributed control without communication is an example of a swarm based control approach. The intentionally cooperative systems are further broken down into four categories – centralized, hierarchical, decentralized, and hybrid [**2**]. Centralized control has a single node that is responsible for all of the decisions. The centralized control is one of the five architectures studied in this work. Hierarchical control divides the system into sub-groups. Control decisions are passed down through the ranks. This method was not included in this work because the problem did not require that level of complexity. Decentralized control gives each of the individual robots the decision making responsibility. Distributed control with communication is the decentralized controller in this work. Hybrid controllers are mixes of the other different approaches. The mixed mode autonomy architectures are a centralized-decentralized hybrid. Communication is either implicit or explicit. More information does not necessarily add to performance [**2**]. This is key to the design of mixed mode autonomy – use communication when it is likely to have more benefit.

An alternative taxonomy of cooperative multi-robot systems is presented by Nardi et al. [**13**]. MRS are grouped as aware or unaware. Aware cooperative MRS would be intentionally coordinated. Coordination of unaware MRS is an emergent behavior of the system since each robot lacks any knowledge of the other robots. Aware cooperative MRS can be strongly, weakly, or not coordinated. The strongly coordinated MRS can be strongly centralized, weakly centralized, or distributed. The systems are also described based on their communication – direct or indirect, team composition, system architecture, and team size. Indirect communication is based on sensor readings and interaction in a common environment (stigmergy). The team composition is heterogeneous or homogenous. The system architecture in this taxonomy refers to whether the system is reactive or deliberative. The team size is the number of robots. For this work the MRS are homogenous with a total number of robots up to 500. The distributed control without communication architecture has indirect communication and is an unaware MRS. All of the other control architectures in this work have direct communication. Distributed control with communication is a distributed MRS with strong coordination. Centralized control is a strongly centralized, strongly coordinated MRS. The mixed mode autonomy architectures are weakly centralized strongly coordinated MRS.

In addition to classifying the MRS, the tasks can be classified. Gerkey and Mataric presented one of the most used classifications [**1**]. They define three axes describing task allocation for robots within the MRS. Robots are either single-task (ST) or multi-task (MT). This part of the

taxonomy describes how many tasks a robot is assigned at one time. Each of the tasks is either single robot (SR) or multi-robot (MR). This describes whether one robot can complete one task or if more than one robot is necessary to complete a single task. Assignment of the tasks is either instantaneous assignment (IA) or time-extended assignment. Instantaneous assignment means the task is allocated once. Time-extended assignment can reallocate tasks to optimize the assignments. Dynamic multi-robot autonomous routing is a ST-SR-IA problem. This task type is equivalent to the optimal assignment problem (OAP) [**1**]. Their review of the research suggests that a centralized approach is better when the number of robots is 200 or less for OAP, but centralized methods do not scale well as the number of robots gets large. Much of the existing research for this type of problem focuses primarily on the task allocation methods to solve OAP. The computational and communication costs for the task allocation have been well studied and the general trends for the common architecture types are known [**14**]. For architectures to solve problems that use instantaneous assignment the task allocation computation and communication requirements scale with the number of robots. The work in this dissertation focuses on the execution of the ST-SR-IA style problems instead of task allocation. Many of the methods used for allocation of tasks, however, can be used for the task execution problem as well.

Another taxonomy of cooperative robotic systems that fits the task execution problem well focuses on the knowledge available to the robots and communication. Gustafson and Matson break MRS down along three axes – robot knowledge, robot communication, and goal knowledge [**15**]. Robot knowledge refers to the information that a controller has about its state and its peers' states. The three levels are local state information (self knowledge only), neighborhood state information, and global state information. Robot communication can be implicit (stigmergy), broadcast, or directed. Goal knowledge can also be described as intention and is organized into three levels similar to robot knowledge. Distributed control without communication has only local state information, knows only its own goal, and relies on implicit communication. Distributed control with communication has neighborhood state and goal knowledge with broadcast communication. Centralized control has a single controller with global state and goal information. All communication is directed to the central controller or from the central controller to the robot receiving a command. Mixed mode autonomy is a hybrid – the robots operate at levels like distributed control with communication, but there is a central controller like with centralized control.

MRS can also be classified by the level of autonomy of the robots. Control of a MRS is divided into four parts – supervision and execution, coordination, planning, and task allocation [**16**]. Each of these parts can assigned one of three levels of autonomy – none (human controlled), operational autonomy (centralized control), and decisional autonomy (each robot makes owns plan). For some of the parts the decision making can be blended between central control and the robot [**17**]. For autonomous MRS, the four parts of control can be assigned to five levels of autonomy [**16**]. Level 1 is fully centralized control for all parts. Level 5 is fully distributed control of the MRS for all parts. The three intermediate levels divide the control of the MRS between a centralized controller and the individual robots. For this work, the distributed controllers (with or without communication) would be a level 5 MRS. The centralized controller would be a level 2 (the robots have some decision making ability as part of the execution or the plan). The levels of autonomy were one of the ideas behind the creation of mixed mode

autonomy. One of the goals of mixed mode autonomy was to create a control architecture that could adjust the level of autonomy for the MRS to improve the efficiency of the control. Nominally, the mixed mode autonomy controllers would be a level 3 since most of the decisions can be made by the robots, but a centralized controller can still be part of the overall MRS control.

One of the primary issues with control and coordination of MRS is the scalability as the number of robots gets larger [**18**]. Most any of the control approaches in literature for single robot tasks will work for small MRS, but as the numbers get larger the computation and communication requirements can become very difficult to reach. For most MRS, the Law of Marginal Return from economics will apply – adding additional workers (robots) results in less and less productivity of each worker [**19**]. This means that there is a point at which adding an additional robot will actually drop the total productivity. The goal of the controller design and evaluation in this work is to push this point of diminishing return to as high a number of robots as possible. For all of the methods of coordination, there are two critical points for the system performance – the point at which the productivity of individual robots starts to drop and the point at which the total system productivity starts to decrease [**19**]. There are two primary factors that influence the productivity of a robot in a MRS – the amount of interference between the robots and the availability of communication. The interference is usually directly related to the number of robots in the same area. The communication availability is related to the amount of information that needs to be shared, the overhead of the communication protocol, and the available bandwidth. The simulations for this work used different environments and task parameters to create different levels of interference and different communication overhead to change the communication availability.

Many of the coordination methods for task allocation and task execution are based on auctions. For task allocation, robots place bids on their cost to complete the task. The lowest bidder is assigned the task. For task execution, robots bid on resources (roads, entrance to an intersection, etc.). The highest bidder is awarded access to the resource. The bids are usually based on how important the resource is to the robot to complete their task. MURDOCH is one of the better known auction-based task allocation methods [**20**]. One agent announces a task, the robots evaluate how well they could do the task, submit their bid, the auction closes, and then the auctioneer awards the task. If tasks are not being completed, they can be re-auctioned. When the execution is highly uncertain (noisy sensing) the robots are better off committing to a task instead switching tasks [**21**]. Similar architectures have been applied for resource access for execution of tasks such as multi-robot routing [**5**]. The challenge for auction-based methods is the communication requirements grow large as the number of robots that can bid gets large. When there are as many tasks or resources as robots, the number of messages is proportional to the square of the number of robots. Some bid aggregation methods can reduce the bandwidth requirements some, but they remain high. Unreliable communication can also significantly reduce the optimality of the system since the auctioneer might miss the bidder that is most beneficial to the system performance.

Negotiation is a coordination method for MRS control that is similar to auctions. Robots propose task assignments or resource usage and the other robots in the MRS respond to their proposals [**22**]. Neighboring robots can accept or reject the task if it is the only robot for the

task/resource or if it is the agent with the highest benefit for the task/resource. Only one robot can accept the task or resource. The robot proceeds when all of its neighbors accept the task or resource. Like auctions, negotiation has communication bandwidth challenges when the number of robots or tasks gets large.

Another form of negotiation for coordination of MRS is plan merging [**23**]. Each robot develops a detailed plan of which tasks they will attempt to complete and how they would complete them. The robots share these plans with the rest of the MRS. The rest of the MRS analyzes the plsnf and then has a chance to address the potential problems caused by the plan – conflict in task allocation or resource usage and inefficient plans. The feedback from the plans is for the robot to change the plan by inserting a waiting period, inserting additional steps or tasks, deleting part of the plan, or completely replanning. Like many other auction or negotiation methods of coordinated MRS the communication requirements do not scale well as the number of robots or complexity of the plans grows large.

Coordination communication requirements can be improved by robots modeling the state of the other robots in the MRS. By anticipating the control decisions other robots might, a robot can plan accordingly. This kind of coordination has been used successfully in environments like RoboCup (robot soccer) [**24**]. The coordination can be implicit or explicit. In the implicit case the robot uses its sensors to predict the state of the other robots. The explicit coordination involves communication of state information between robots. Combining the implicit and explicit estimation of the other robots' states can significantly improve the reliability of the coordination [**25**]. For this work, the distributed control with communication and mixed mode autonomy incorporated the sharing of information to build world models to keep the communication requirements lower than the auction or negotiation methods.

Coordination can be handled by breaking the MRS down into subgroups. These subgroups can be static or dynamic. Scerri compared three hierarchical MRS control architectures for large scale systems with 100 or more robots [**26**]. The methods (Machinetta and Teamwork, Centibots Dispatching, and Cooperative Mediation) compared were all highly structured with top-down guidance (no swarms) and all divided the MRS up into smaller sections. Machinetta and Teamwork is hierarchical control scheme based on dividing the MRS into subgroups and modeling the state of each of the robots to reduce some of the communication. Centibots Dispatching selects a managers (dispatchers) out of the team. MRS goals are assigned to the managers. Each robot reports to one or more manager to get assignments. Cooperative mediation relies on robots mediating a solution when resource conflicts arise. The mediators are effectively a localized central controller. One of the common observations across the three methods was that communication was far more of a limiting factor than computation. The problems with these methods were with communication for the hierarchical methods (Machinetta and Teamwork and Centibots Dispatching) to successfully get information to the robots in subgroups and with unpredictable emergent behaviors in cooperative mediation. Mixed mode autonomy was designed to reduce these kinds of problems. By avoiding subgroups the communication latency problems were reduced since messages are direct. Oversight by a single central controller prevents the problems with unintended emergent behaviors.

2.3 Dynamic Multi-Robot Autonomous Routing

This section details the methods for solving problems similar to dynamic multi-robot autonomous routing. Attention is paid to the information requirements of each method and whether the method can be done centrally and/or distributed. Multi-robot routing is the process of planning and executing paths to get a team of robots from their current locations to their destinations. Each of the methods typically establishes a set of traffic safety rules to protect the robots. The normal rules include things like safe following distances, only one robot in an intersection at a time, no passing, and speed limits [**6**]. These rules are the constraints on the planned paths and execution of the routes. The routing methods will attempt to maximize the system performance. The system performance is usually defined in terms of the time required for the robots to travel to their destinations. The time to travel the paths is function of the distance traveled, time required to handle the safety rules, and any latency due to the controller [**7**]. The ideal case is for the robots to drive the shortest distance possible at the maximum speed, however, as the number of robots increase this gets more difficult to achieve. The routing problem can be divided into three levels – strategic, tactical, and operational [**27**]. The strategic is the planning of routes. The tactical is following the rules of the road. Operational is the lowest level where the actual commands are given to the robot. Most of these methods focus on the strategic and tactical levels. Dynamic multi-robot autonomous routing is similar to the problem posed by the MAGIC 2010 competition, but at a significantly larger scale of MRS[**28**]. The methods are divided into centralized and distributed approaches.

Centralized control relies on a single controller to make all of the routing decisions. Central control does not have as many variations as the distributed controllers since there are only so many ways to structure a top-down controller. Many of the early attempts at routing robots in a MRS were centralized [**6**]. Since the work was all largely in simulation, the practical considerations of communication scaling were not considered in many of these works. Many of the centralized approaches build off of the traditional single robot routing algorithms like Dijkstra's Algorithm or A* [**29**]. They add costs associated with highly used roads or roads that are likely to be needed by other robots. For the purest forms of centralized control, the robots are essentially distributed sensors and actuators and all of the intelligence is in the centralized controller. The communication for these methods requires status updates from all of the robots and commands from the centralized controller. The more complex the environment, the more frequently messages will be required to be sent. The bandwidth requirements increase linearly with the number of robots.

Many of the centralized approaches combine task allocation with the routing problem to come up with optimally efficient solutions. This is analogous to the Multiple Travel Salesperson problem. In most real world application of this combined approach, the optimality constraints are relaxed to reflect the errors in modeling and sensing, communication failures, and the time constraints involved with finding the optimal solution [**30**]. These methods have similar communication requirements to other centralized routing algorithms, but are more computationally intensive. Lim and Wang compared the difference in system efficiency between integrated assignment and routing (one-stage) separate assignment and routing (two-stage) for the Multi-Depot Vehicle Routing Problem (MDVRP) [**31**]. They found that the one-stage approach did well, especially with 200 plus destinations (sinks).

SiPaMoP (Simultaneous Path and Motion Planning) is another one of the combined task allocation and routing algorithms [**32**]. The issues SiPaMoP addresses are that shortest path algorithms do not account for the number of robots, congestion and bottlenecks are not managed, and collision avoidance is inefficient. The goal is to find the shortest time path for each of the robots to reach their destination. The time is increased if there is another robot is likely to reach the same spot within a certain time difference ΔT. The paths are adjusted to find a shorter time path or adjust the speed of the robot to keep robots from reaching an intersection with ΔT of each other. The authors implemented SiPaMoP with a centralized controller and four robots. The core routing algorithms for this work used by all of the control architectures except the distributed control without communication are inspired by the framework of this routing algorithm. For larger scale systems, the authors used Ant Colony Optimization (ACO) with similar routing concepts to find route solutions [**33**]. This was near optimal, but rather computationally intensive.

Many of the newer variants of centralized control are actually hierarchical control with decision making locally centralized. The MRS is divided into subgroups with a leader that acts as the centralized controller. This can reduce the communication requirements and computational complexity by dividing the routing problem up. Problems can arise however if tasks cannot be contained within a sub-region of the environment. One example of a locally centralized method uses sharing of world models to plan paths in subgroups [**34**]. Each robot generates a path (a set of trajectories) for itself and all of the other robots in its subgroup and shares that with the group. The robot with the best scoring plan is chosen as the leader and directs the implementation of the plan by the group. This method can work well as long as the subgroup size is relatively small since the information requirements will group with the square of the number of robots in a subgroup.

Most of the existing centralized control approaches to MRS coordination either neglect communication costs and constraints or assume a complete graph [**35**]. These considerations can have a strongly negative impact on the performance of the centralized routing algorithms. Distributed control gives the route decision making to the individual robots. Distributed routing has the benefit that it does not have a single point of failure that will cause the system to stop functioning. Failure of one of the individual robots will degrade the overall performance, but will not completely stop the system. Distributed routing can be highly structured like centralized control or rely on behavior emerging from simple rules. The emergent behaviors are typical of swarm type systems.

The swarm-based routing methods rely on simple rules executed by each of the robots. Most of the communication and coordination is implicit. At the lowest level control of the swarms are based solely on the sensor inputs. These low-level methods rely on sensing other robots and tracking common landmarks [**36**]. Other methods use simple communication like beacons or beacon emulation [**37**]. These methods have been shown to have low computation and communication costs and are robust to disturbances in the system, but are not efficient systems compared to the potential performance of deliberative systems [**38**]. The lower efficiency means that the MRS needs more robots to accomplish the same tasks. For some applications, the robustness is considered to be enough of benefit to be worth this cost. Because of the less generally lower efficiency of swarm systems they are not necessarily appropriate for the kind of

applications targeted by dynamic multi-robot autonomous routing (automation of open pit mines, automation of container terminals, and automation of warehouses).

The more structured distributed routing methods are based on communication and closely mirror the techniques used for distributed coordination of MRS. The distributed routing methods include auctions, plan merging, and shared world models.

Auction based routing algorithms reduces the communication requirements by reducing much of the information to be shared to solve the routing problem to bids. The auction methods are suboptimal, but can achieve results that are near optimal in many instances. Bids are for access to a resource (road segment, entrance to an intersection, etc.). Unlike with task allocation where the bids are based on the cost, the bids are based on the reward the robot gets if it has access to the resource [**39**]. Thus the more important the resource is to completion of the robot's task the higher the bid. The communication requirements scale linearly with the number of robots for auction-based routing. The bigger communication challenge can be latency when multiple resources require a bid at nearly the same time.

Plan merging for multi-robot routing allows each robot to plan their route. They then share their routes and then negotiate on the conflicts between the routes. In most cases the number of steps included in the plan is limited to keep the communication bandwidth requirements low [**40**]. The plan merging algorithms are a way to arbitrate the conflicts between robot plans. The communication requirements are low unless except when the plans get more detailed or the environments becomes more congested which leads to more conflicts in the plans.

Another distributed routing method is to share world models between the robots in the MRS. The concept is that if all of the robots have largely the same information they can make decisions that maximize the group performance (utility) instead of just their performance. Instead of auctioning or negotiating for access to resources, robots understand when resources they would like to use are more important for overall MRS performance for another robot in the system. These methods can build models based on communication information, sensing, and prediction of the next step. The routing methods used for this work with the exception of distributed control without communication use world model sharing as a basis of the coordination of their routes.

One of these methods makes a set of rules for the routes and the robots try to plan routes that follow the rules and prevent the other robots from having to break the rules to complete their routes [**41**]. The rules include items like one robot per intersection, maintain safe following distance, keep to the left, and communicate when breaking a rule to avoid an obstacle. The information sharing does not have to be done at the same rate for all of the robots in the MRS. Asynchronous information sharing is used to keep the system from waiting for information from computationally slower robots [**42**]. The robots share their information when it is ready instead of at fixed time intervals. This allows the robots to all have the most current information possible. Mixed mode autonomy uses asynchronous communication, but for reduction of communication requirements instead of differences in computation time.

2.4 Example Applications

This dissertation looks at dynamic multi-robot autonomous routing for three example application – automation of open pit mines, container terminals, and warehouses. These application domains each represent a different set of conditions for dynamic multi-robot autonomous routing.

Open Pit Mines

Automation of open pit mines is a significant goal of the mining industry. Labor costs and availability are high in many of the remote mine sites [**43**]. It is also a dull and dangerous job. The work is high repetitive and, due to the scale of the haul trucks, accidents involving other mine site vehicles can be deadly. Automation of the haul trucks can potentially reduce the labor costs and improve the safety and reliability of the system. Many of the technologies for autonomous vehicles are already incorporated into mine operation for tracking a safety. Many of the trucks and shovels already include GPS to track individual vehicles [**44**]. Work has already been done to implement some collision avoidance behaviors on mine equipment [**45**]. Modern dispatching systems for open pit mines resemble the centralized control systems typical of autonomous MRS with humans providing the high-level decision making [**46**]. Part of the reason for the work in this dissertation is to determine is the dispatcher style systems used in the application domains can be used as the basis of an autonomous centralized controller for automation of the applications. For the open pit mine, the primary interest is how well the systems would scale for large MRS.

Container Terminals

Container terminal automation has been studied for quite a while [**47**]. Many of the early MRS control domains simulated where container terminals or similar environments. The MARTHA project was one of the early works in this area [**40**]. A single site can have several hundred Automated Guided Vehicles (AGV). Like with the open pit mine, one of the primary concerns is with the scaling of the system. One of the approaches taken in simulation work so far has been to divide the group of AGV into platoons [**48**]. The simulation results were good for the size of systems in the simulation, but the upper limit on size might prevent the method from working for larger container terminals. Another recent method incorporates maintenance scheduling into the problem to improve the reliability [**49**]. There overall method was centralized and worked very well for the 30 or less AGV in simulation, but again may not scale very well.

Warehouses

Warehouse automation could potentially reduce the cost of handling goods and improve the overall efficiency [**50**]. Kiva is an example of a commercially available system for automation of pick, pack, and ship [**8**]. The Kiva System solution is for new designs of warehouses – they have shown good results and have been adopted by a number of major companies. However, it would also be useful to be able to automate existing warehouses. In these cases the problem can be expressed as dynamic multi-robot autonomous routing. Based on the number of different items or orders processed daily in a warehouse the number of robots required could be large. Centralized control approaches that mirror a traditional dispatcher have been studied [**51**]. They showed that the dispatcher policies improved performance over less structured approaches.

Chapter 3 – Coordination and Control Strategies

Control of multi-robot systems (MRS) can be divided between a centralized controller and the individual robots. This section presents several different control strategies for dynamic multi-robot autonomous routing. The strategies represent common approaches to multi-robot control and a novel architecture developed as part of this work. The approaches described in this chapter are centralized control, distributed control (with and without communication), and Mixed Mode Autonomy. The goal of Mixed Mode Autonomy is to take advantage of the strengths of traditional centralized and distributed methods while minimizing the difficulties. Each of the strategies was evaluated by simulation.

The control of the MRS is divided into four parts for dynamic multi-robot autonomous routing – route planning, route execution, communication management, and world modeling. The route planning and execution should allow each robot to reach their destinations in a minimum travel time while ensuring sufficient separation between robots and avoiding collisions. Ideally the collisions are prevented primarily by routing to manage congestion, but a basic control scheme is needed to handle avoiding obstacles and failures in congestion management. The control in this model does not include motion control to follow the path planned. This level of detail is abstracted to focus on the performance of high level control and system coordination. Communication management determines which messages the robots send to the centralized controller or their neighboring robots and how often to send them. The world models are the robots' and central controller's representation of the environment and other robots. It is used to plan routes and communication.

This chapter describes how each of the controllers handles these parts of MRS control. The details of the controller implementations are described in Chapter 4 as part of the simulation models.

3.1 Control Requirements

Typical MRS control applications need to address task allocation, planning, and execution. For the dynamic multi-robot autonomous routing problem studied in this work, the task allocation is handled by online assignment for all of the controllers. The next task is assigned to the next available robot. The task planning selects a route for the robots to reach their destinations. These routes should minimize the travel time by reducing the congestion (number of robots on the same road section). The plans are updated at each intersection to reflect the current MRS operating conditions. The task execution completes the planned route while maintaining the safe separation distance between the robots. The plans are not implemented exactly due to velocity variations and uncertainties or errors in the model.

Multi-Robot System Capabilities

The multi-robot system (MRS) is made up of a set of robots and for some of the control architectures a centralized controller. For all of the controllers, the robots have a common set of capabilities. The robots are represented as occupying a single point – abstracting much of the lower level effects. The robots in the MRS for this work are homogeneous to simplify

simulations. Nothing in any of the control architectures, however, precludes heterogeneous robots. Each robot can sense its position in the environment. For real robots, this would be a localization sensor like GPS or an inertial navigation system. Each robot is also equipped with sensors to detect objects in its path. With real systems, this would be LIDAR, RADAR, or computer vision sensors to detect obstacles or other robots. To account for the effects of real sensors, the model of the sensors includes a false positive rate (percent of the time a robot detects an object in the path of the sensor whether one is present or not). All of the robots travel at the same speed. Robots can be equipped with a wireless communication system to share information with other robots or a central controller. A summary of the parameters describing the robot capabilities is shown in Table 3-1.

Table 3-1. Robot Capabilities

Capability	**Parameter**	**Description**
Mobility	Speed	
Sensing for Collision Avoidance	Sensor Range	Maximum distance at which obstacle can be detected
	False Positive Rate	Percent of measurements that incorrectly indicate an obstacle is present within threshold
	False Negative Rate	Percent of measurements that fail to detect an obstacle that is present within threshold
Sensing for Localization	Localization Accuracy	Error in position measurement
Communication System	Frequency	Carrier frequency of wireless system
	Communication Range	Set by transmission power – distance a signal can be sent
	Communication Data Rate	Available bandwidth to send messages

Multi-Robot Task Environment

The robots in the MRS operate on a network of roads consisting of road segments and intersections. All robots must remain on the road and follow basic driving rules – yield to traffic in an intersection, obey speed limits, and maintain a safe following distance from a robot on the same section of road. The road environments are uniform grids for this work (other environments are less connected, but more computationally complex to simulate). Two primary parameters shape the environment – the grid pattern and the bottleneck size. The grid shape is the ratio of the length of the segments in the y direction to the length of the segments in the x direction. The bottleneck is the width (number of road segments) of the narrowest section of the grid. An example environment and the environmental parameters are shown in Figure 3-1.

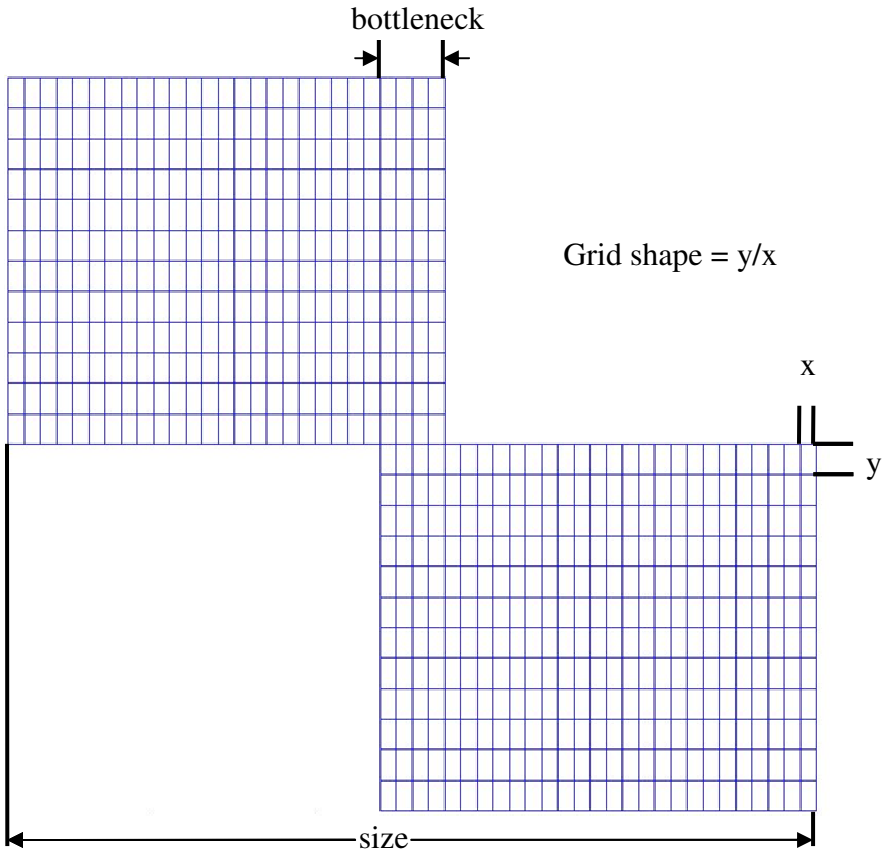


Figure 3-1. Multi-Robot Task Environment

Task Allocation

The robot tasks are to complete trips to destinations (intersections) in the road network. New destinations are assigned to the robots after they reach their current destination. All allocation is online assignment – the next destination is assigned to the next available robot. This is not an optimal allocation of robots, but can be shown to be three-competitive (worse by less than a factor of three) [**1**]. This allocation scheme is common for many applications, especially ones where the tasks are not known in advance. Using online assignment also allows this work to focus on improving the performance of the robots while they plan and execute the tasks.

Controller Structure

Each controller includes a world model, a route planner, a route execution and collision avoidance controller, and a communication management system. The world model is a representation of the task environment that contains the information the robots or central controller use such as the map/graph of the road network, their own location, the location of other robots, and the current task information. The route planner selects a path to complete the tasks for dynamic multi-robot autonomous routing. The route execution and collision avoidance controller calculates the velocity of the robot to follow the path and handles the sensing to detect and avoid obstacles and other robots. The communication management system determines which messages to send and how frequently to send them. These parts are divided between a

centralized controller and the individual robots. Each of the control methods represents a different division of these responsibilities. The control methods used for this work are distributed control without communication, centralized control, distributed control with communication, mixed mode autonomy, and mixed mode autonomy with model updates.

3.2 Distributed Control with No Communication

Distributed control with no communication is a stigmergic controller. All coordination is handled by interactions with a common environment. Sensors on the robot are the basis on all control. The coordination is an emergent behavior that is a result of the simple behaviors of each robot. Distributed control with no communication is the lower bound for task performance. All of the other control architectures should perform better with the additional information available to the controllers through the communication system.

World Model

Each robot has a priori information on the network of roads. They know their current location and the destination they are trying to reach. The robots have no knowledge of their peers except for what they can sense. The robot world model (kept by each robot) is shown in Table 3-2.

Table 3-2. Robot i World Model for Distributed Control without Communication

Position robot i	Closest object to i	Destination	Next Intersection

Route Planning

Distributed control with no communication has the simplest route planning since the robots have no knowledge of any other robots and all of the information used to plan the route is static. Dijkstra's algorithm is used to find the set of shortest paths between each pair of nodes in the network of roads. Since the network is static, the algorithm is only run once and the result is stored on each robot. The robot selects from the set of shortest paths from its current location and its destination. When more than one path is available the selection is made randomly from the set of paths. Dijkstra's Algorithm in summarized below.

For each node in the road network, k = 1 – m

1. Set the value for distance to the node $d_k = 0$, all other $d = \infty$
2. Mark all nodes as unvisited
3. Set k, the initial node, as the current node
4. Calculate tentative distances to the unvisited adjacent nodes to the current node
5. If the tentative distance for a node is less than the current value, replace it with the tentative distance. Store the current node as the antecedent for that node.
6. Once the tentative distance to all the adjacent nodes to the current node have been calculated mark the current node as visited
7. If all nodes have been visited the algorithm is done. Otherwise, visit the unvisited node with the shortest distance.

8. The antecedents are used to generate the path between the initial node k and all of the other nodes.

Route Execution and Collision Avoidance

At each time interval of the control algorithm, the robot velocity and heading is set. First, the robot checks its collision avoidance sensors. If it detects a robot within the separation distance, it will pause (set velocity to zero) for that time interval. If nothing is detected, the robot sets its velocity to its maximum velocity V_i and moves towards the next vertex on its route R. Integrated over time this algorithm to set velocity is functionally equivalent to slowing down to maintain the separation distance. In real systems, this is the approach that would be taken; however, it is much simpler to use the binary velocity (zero or max) for simulations. The velocity for robot i is

If $d_{obj} > separation\ distance$

$$v_i = \frac{(x_{nv,i} - x_i)\hat{\imath} + (y_{nv,i} - y_i)\hat{\jmath}}{\sqrt{(x_{nv,i} - x_i)^2 + (y_{nv,i} - y_i)^2}} \times v_{avg}$$

If $d_{obj} \leq separation\ distance$

$$v_i = 0$$

where $(x_{nv,i}, y_{nv,i})$ is location of the next vertex of robot i, (x_i, y_i) is the current location of robot I, and v_{avg} is the standard robot speed.

Communication Management

Since the robots do no communicate, they do not have any communication management system. A real system would likely still include the ability to communicate to the whole system to pause the system in an emergency.

Summary

Distributed control without communication is the simplest of the four controllers simulated for this work. Due to the mostly open loop nature of the control, distributed control without communication would not likely be used for any of the applications suggested with this work. It, however, establishes the lower baseline of performance for control of a MRS with the dynamic multi-robot autonomous routing problem, since any information added to a well designed controller should improve the performance. A graphic summary of how distributed control without communication would work is shown in Figure 3-2. The shortest paths are precalculated with Dijkstra's Algorithm and stored in memory on each robot.

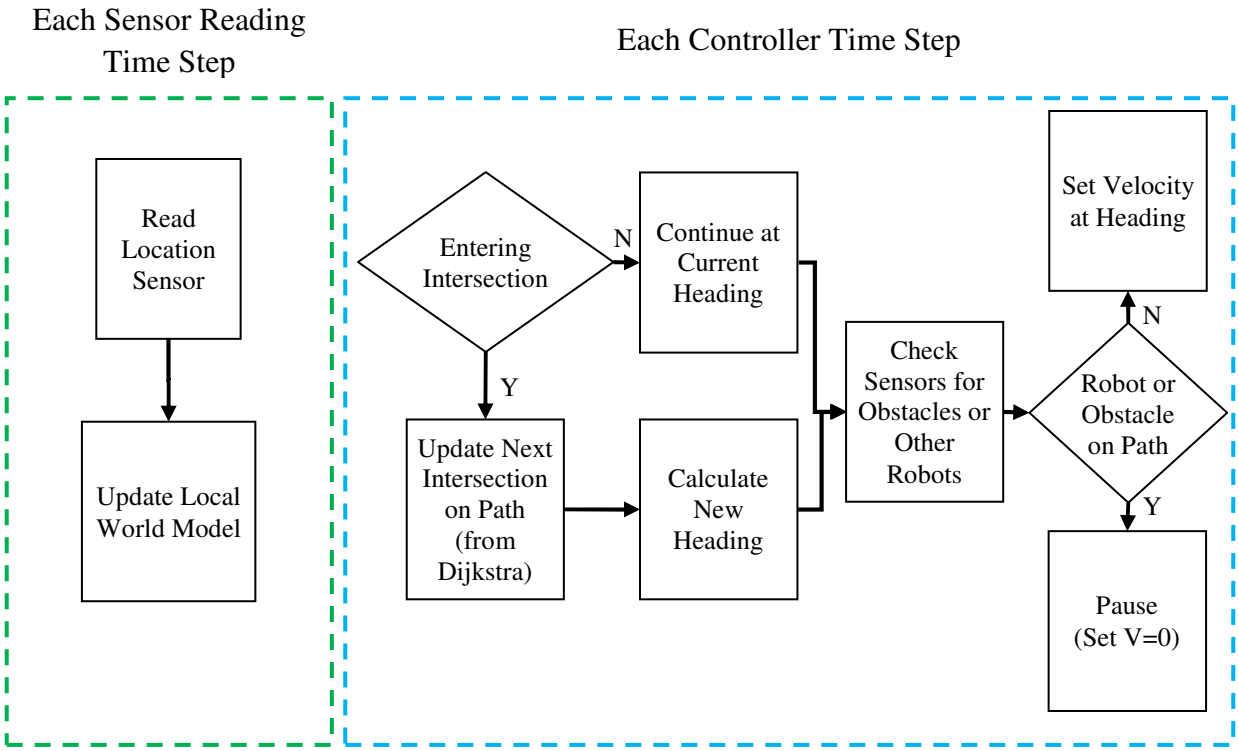


Figure 3-2. Distributed Control without Communication

3.3 Centralized Control

Centralized control is one of the more common approaches to multi-robot control. The central controller makes all of the decisions about route planning and communication. Effectively the multi-robot system becomes a distributed set of sensors with one controller. This can be efficient for optimal control of the system, but the MRS has a single point of failure with the centralized controller.

World Model

The central controller maintains a model of the complete multi-robot system that includes the location or every robot, the assigned tasks for each robot, and all the planned routes. The model is updated through communication with each of the robots. Discrepancies between the model and the real state of the system happen when communication failures occur. The world model for the centralized controller is summarized in Table 3-3.

Table 3-3. World Model for Centralized Controller

Robot 1	Position 1	Next Vertex 1	Destination 1	Needs Command 1	Command Queue 1	Last Command Received 1
⋮						
Robot i	Position i	Next Vertex i	Destination i	Needs Command i	Command Queue i	Last Command Received i
⋮						
Robot N	Position N	Next Vertex N	Destination N	Needs Command N	Command Queue N	Last Command Received N

Route Planning

The route for each robot is planned by the central controller based on the information in the world model. The route planner is based on Dijkstra's Algorithm in a similar method to the distributed controller without communication. The raw distances in the weighted adjacency matrix are replaced with travel times. For road segments with no other robots present, the travel time is just the maximum velocity multiplied by the distance. When more than one robot is on a segment or the proposed route would cause two robots to occupy the same segment, the travel time is the nominal time plus the pause time required to maintain the safe separation distance. Thus the robots are assigned paths that are likely to be less congested so they are more likely to be able to maintain their maximum velocity. The routing algorithm for robot i is summarized below.

1. Dijkstra's Algorithm is performed at the initialization of the system as is described in section 3.2.
2. Find the paths that start h steps from the initial (current) node. These are the candidate paths.
3. Assign the shortest path cost from Dijkstra's Algorithm as the base cost for the candidate paths.
4. Calculate the cost of the path from the current node to the start of each of the candidate paths
    a. The nominal cost of these paths is found from Dijkstra's Algorithm
    b. The cost is increased by the estimated possible delays to robots on or possibly on segments during the time the robot would complete the path
5. Add the costs from steps 3 and 4 to get the cost of each candidate path
6. Select the lowest cost path

A new route is calculated whenever a new task is assigned to a robot. The centralized controller checks the robots' routes when the robot is close to completing a road segment (nearing an intersection) to verify that the route is one of the lowest cost paths and a command is added to the message queue. The central controller sends the route information to the robots in the order the routes are added to the queue.

The robots wait at the intersection until they receive confirmation of their route, which is normally sent to the robot before it reaches the intersection. Normally, there will not be any

delays, but message failures or message latency when the MRS has a large numbers of robots may cause the robot to be idle while it waits for a message.

Route Execution and Collision Avoidance

The route execution and collision avoidance is handled in much the same way as with distributed control with no communication. At each controller time interval the velocity is set to V_i in the direction of the next vertex or to zero if the robot is too close to a robot ahead of it on the road segment. The robots should receive their next vertex of their route before they reach an intersection. The exceptions are when the command message from the central controller fails (dropped packet) or when the network is saturated and the central controller is not able to send the message. Robots wait at an intersection until it receives a command. The velocity of robot i is

If $d_{obj} \leq separation\ distance\ OR\ (x_{nv,i}, y_{nv,i}) = (x_i, y_i)$

$$v_i = 0$$

If $d_{obj} > separation\ distance$

$$v_i = \frac{(x_{nv,i} - x_i)\hat{\imath} + (y_{nv,i} - y_i)\hat{\jmath}}{\sqrt{(x_{nv,i} - x_i)^2 + (y_{nv,i} - y_i)^2}} \times v_{avg}$$

where $(x_{nv,i}, y_{nv,i})$ is location of the next vertex of robot i, (x_i, y_i) is the current location of robot I, and v_{avg} is the standard robot speed. $(x_{nv,i}, y_{nv,i})$ is updated when the vertex is reached and a new command has been received.

Communication Management

Each robot sends the central controller regular updates of its position on the network of roads. These updates are scheduled to occur at each controller interval (ten times per segment of road for uninterrupted travel). When a robot's estimated time of arrival at an intersection is within three controller time intervals, the central controller checks that robot's route and adds a corresponding command message to the central controller's message queue. Messages from the central controller are sent FIFO. The central controller messages are a higher priority on the communication network than the robot messages. The robot status updates also include the next vertex on the robots route. This information is used by the central controller to confirm the commands it has sent to the robots. If a robot's next vertex does not match the one sent by the central controller, the central controller knows that the previous command message failed and adds the command to the message queue to be retransmitted. Table 3-4 shows a list of the messages sent by the central controller and individual robots for centralized control.

Table 3-4. Messages for Centralized Control

Central Message			
Robot address	Retransmit flag	Time Stamp	Next Vertex
Robot i Message			
Robot i address	Location i	Receive time of last command	Next Vertex i

Figure 3-3 shows the realistic bounds on the information requirements as a function of the number of robots for centralized control. The information requirement is based on the number of messages that the robots and centralized controller send. During the nominal time a robot travels a segment of road, they send ten status updates. The centralized controller must send one command for each segment. The upper bound assumes a 5% failure rate for commands from the central controller. This means that 5% of these messages will need to be retransmitted. Both limits assume that the robots are completing tasks without pausing due to congestion. Pauses would increase the time required to complete a road segment and thus reduce the frequency that commands would need to be sent. The status update messages are sent at a constant rate and are unaffected by how the tasks are completed. The effective bandwidth is the rate data can be sent after packet overhead is subtracted from the wireless communication channel's total bandwidth.

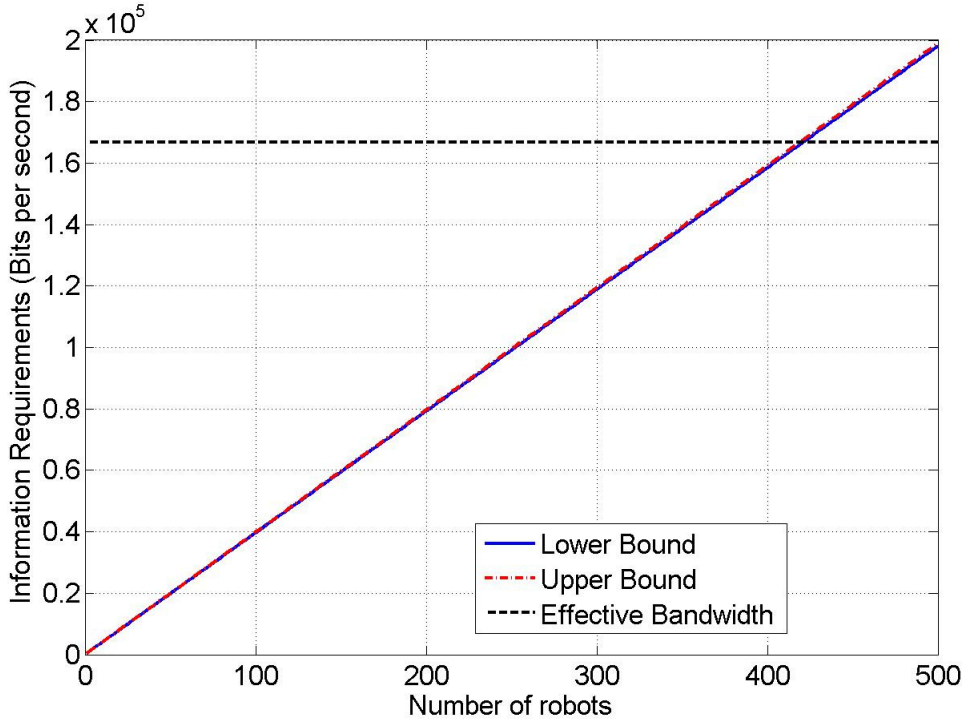


Figure 3-3. Realistic Bounds for Information Requirements for Centralized Control

Summary

Centralized control relies on a single controller to make all of the high level (planning) decision for the MRS. Communication from the robots allows the central controller to keep track of the

state of the MRS. This information is used to plan the routes of the robots to minimize congestion while keeping the route lengths near their minimum. A graphic summary of how centralized control works is shown in Figure 3-4. The shaded commands are done by the centralized controller. All of the others are completed by the robots.

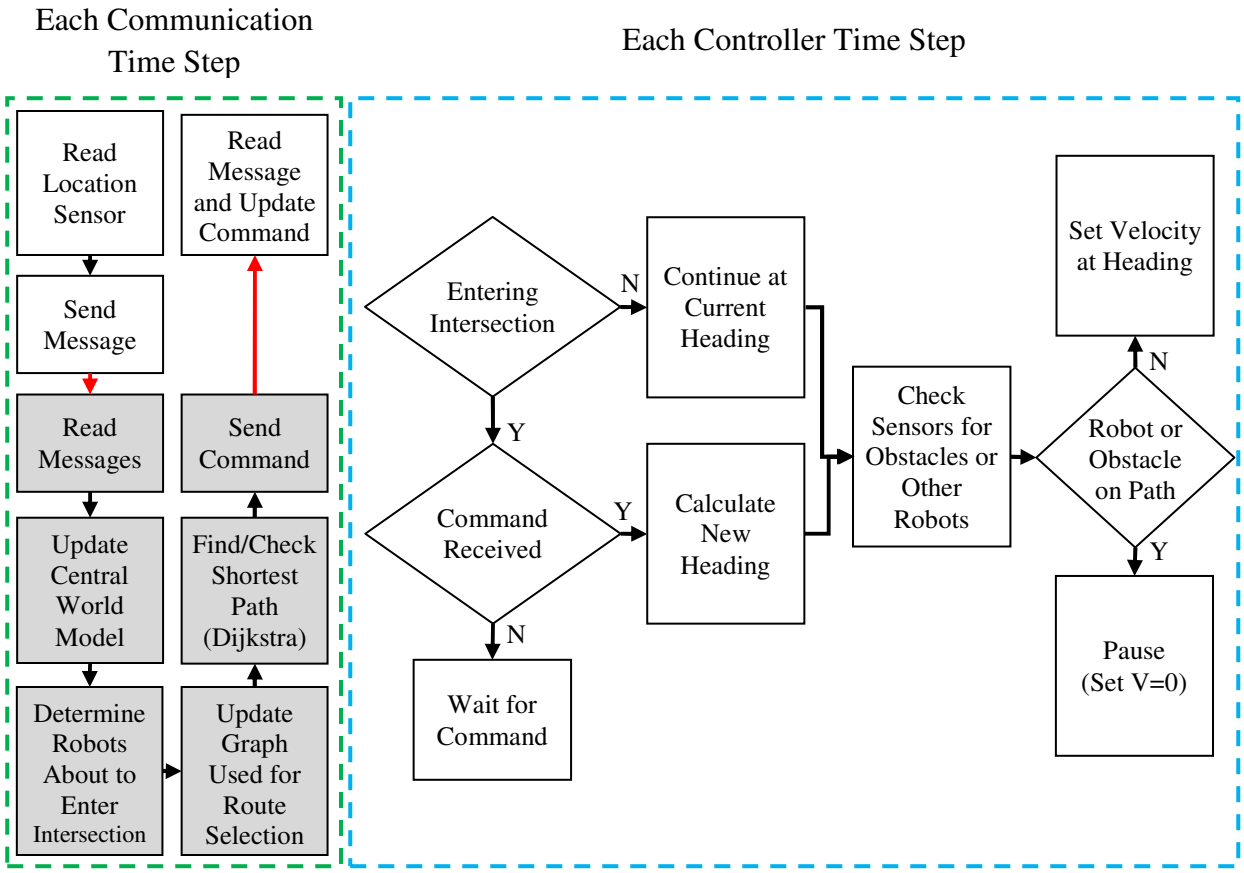


Figure 3-4. Centralized Control

3.4 Distributed Control with Communication

Distributed control with communication makes each individual robot responsible for its own route planning. Unlike the distributed control described in section 3.2, each robot has knowledge of its peers within the MRS. The system no longer has a single point of failure like with centralized control, but it loses some of the predictability of having one controller handle all the decision making.

World Model

Each robot keeps a model of the MRS with information about the location and goals of each robot. This model is updated through peer-to-peer communication within the MRS. With the distributed world models, the MRS now maintains N different world models (one kept by each robot), instead of one central model. Dropped packets cause the models to differ. For robots that

are far apart (the ones more likely to not receive a message due to a dropped packet), this is less of a problem since distant robots will not have much effect on their planning. In MRS with a high number of robots, latency of the messages could have a small effect on the model. Since the robots are constrained to remain on a road segment, each robot can model approximately where other robots are without information updates until the time a robot would reach an intersection. At this time, with the information about the final vertex in the robot's path, probable paths can be calculated. This allows the robot to take into account the possible locations of the robot that it has not received messages from recently, but over estimates the likelihood that that robot will cause congestion. The world model of robot i is summarized in Table 3-5. Each of the N robots keeps a model in this form.

Table 3-5. World Model kept by Robot i for Distributed Control

Robot 1	Position 1	Next Vertex 1	Destination 1	Last Update Received 1
⋮				
Robot i (own information)	Position i	Next Vertex i	Destination i	Distance to closest obstacle
⋮				
Robot N	Position N	Next Vertex N	Destination N	Last Update Received N

Route Planning

Distributed control with communication uses the same route planning algorithm as the centralized controller. This algorithm is described in section 3.3. The difference is that each robot plans its own path and uses the information in its own world model. When communication is reliable or few robots are near each other the routing with the modified Dijkra's Algorithm will end up being nearly identical to with the central controller. Since each robot plan their own route, they do not need to ever wait for a command before entering an intersection.

Route Execution and Collision Avoidance

The route execution and collision avoidance is largely the same as with the other controllers. Each robot sets its velocity to V_i in the direction of the next vertex or zero if a robot is detected within the safe separation distance each time the control loop runs. The velocity for robot i is

If $d_{obj} > separation\ distance$

$$v_i = \frac{(x_{nv,i} - x_i)\hat{\imath} + (y_{nv,i} - y_i)\hat{\jmath}}{\sqrt{(x_{nv,i} - x_i)^2 + (y_{nv,i} - y_i)^2}} \times v_{avg}$$

If $d_{obj} \leq separation\ distance$

$$v_i = 0$$

where $(x_{nv,i}, y_{nv,i})$ is location of the next vertex of robot i, (x_i, y_i) is the current location of robot I, and v_{avg} is the standard robot speed. $(x_{nv,i}, y_{nv,i})$ is updated by each robot when it reaches an intersection.

Communication Management

Each robot sends its status updates each time the controller runs. Nominally this is ten times during the time to travel a road segment. The robots may not have actually travelled that far due to pauses to maintain the safe separation distance. This update rate is a tradeoff between keeping up accurate models and the bandwidth required. Table 3-6 is a listing of the messages sent by the robots. Each of the robots sends the same message to its peers.

Table 3-6. Messages for Distributed Control

Robot i Message			
Robot i address	Location i	Destination i	Next Vertex i

Figure 3-5 shows the information requirements as a function of the number of robots for distributed control with communication. During the nominal time a robot travels a segment of road, they send ten status updates to the other robots. Since the update time is fixed and no failed messages are retransmitted the information requirement is constant for a given number of robots.

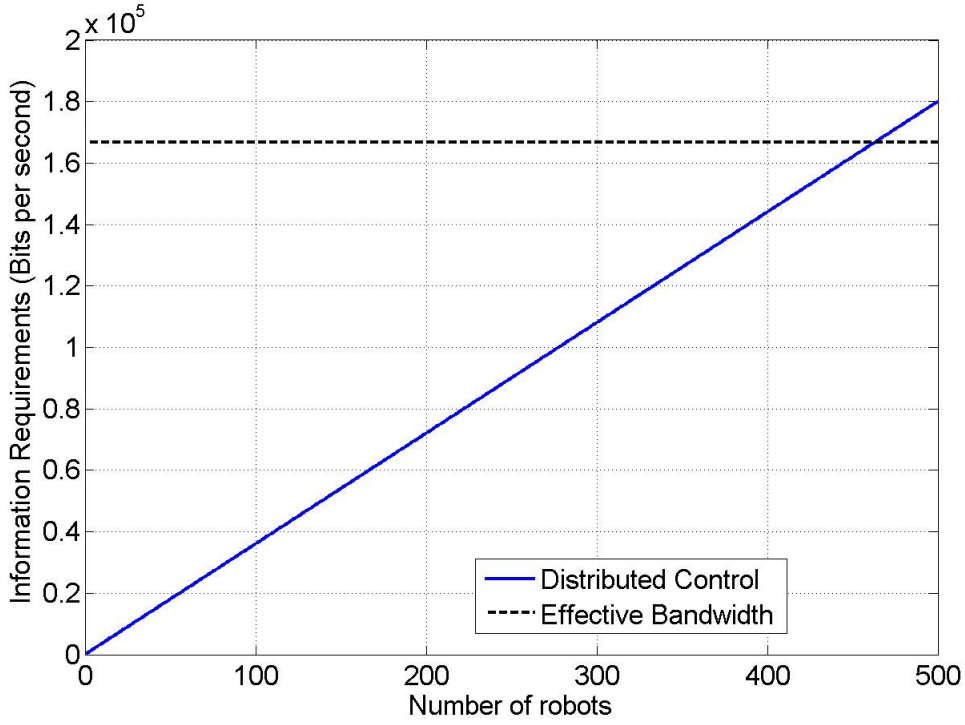


Figure 3-5. Information Requirements for Distributed Control with Communication

Summary

With distributed control with communication, each robot makes all of its own control decisions. Peer-to-peer communication allows each robot to maintain a model of the current state of the MRS. The robots use these models to plan routes to avoid congestion to solve the dynamic multi-robot autonomous routing problem. A graphic summary of how distributed control with communication would work is shown in Figure 3-6.

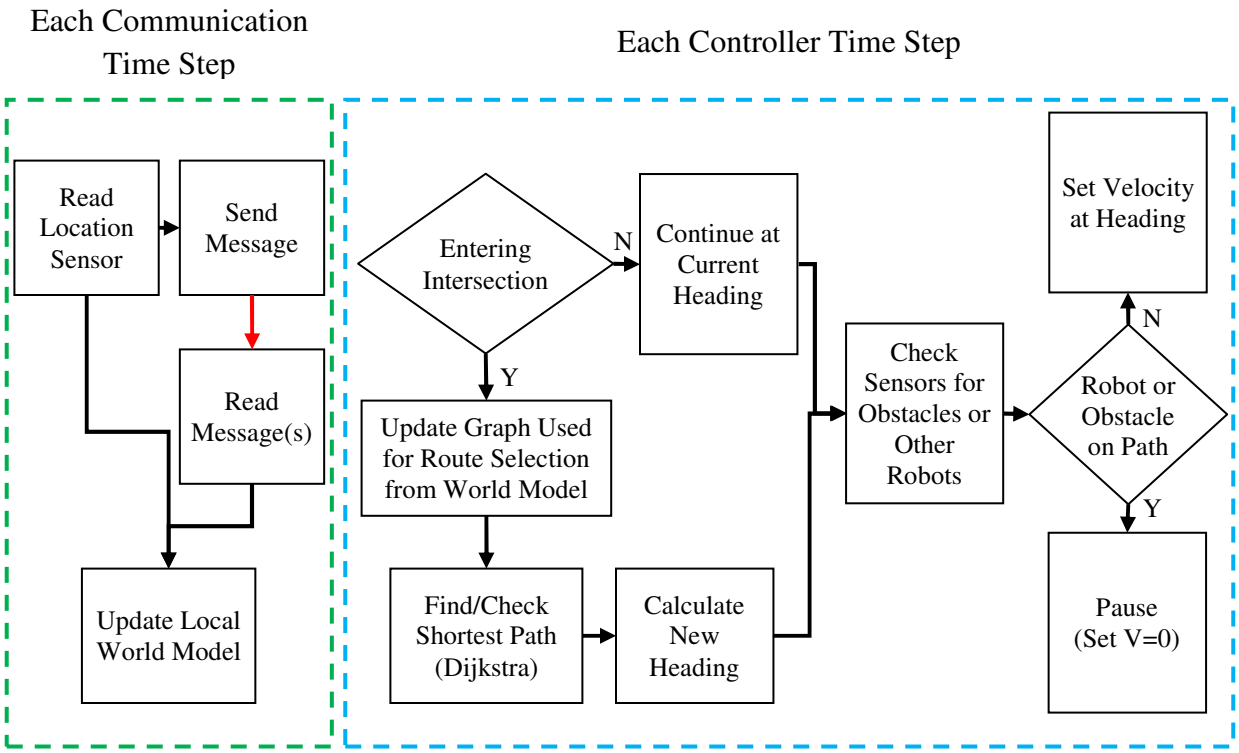


Figure 3-6. Distributed Control with Communication

3.5 Mixed Mode Autonomy

Mixed mode autonomy was developed to take advantage of the benefits of centralized control and distributed control with communication. Mixed mode autonomy gives each robot the same responsibilities as they have with distributed control with a central presence. Route planning is handled locally by individual robots, but is supervised by a central controller. This eliminates the dependency of the system on the central controller while still having most of the benefits of centralized control.

World Model

Each robot maintains a world model based on peer-to-peer communication as they did with distributed control with communication. A central controller uses these same messages to also maintain a world model. These models track the current locations, next vertices, and final vertices for each robot in the MRS. The central world model also includes how likely the information in its model is as current (robot still on the same road segment) and whether the next

vertex is a new command. The model data is considered current if the robot i would still be on the same road segment if it traveled at the average speed since the time the message was received. The change in next vertex value is used to determine whether the central controller should check routes. The models are summarized in Tables 3-7 and 3-8.

Table 3-7. World Model kept by Robot i for Mixed Mode Autonomy

Robot 1	Position 1	Next Vertex 1	Destination 1	Last Update Received 1
⋮				
Robot i (own information)	Position i	Next Vertex i	Destination i	Distance to closest obstacle
⋮				
Robot N	Position N	Next Vertex N	Destination N	Last Update Received N

Table 3-8. World Model for Centralized Controller for Mixed Mode Autonomy

Robot 1	Model Update Time 1	Position 1	Next Vertex 1	Destination 1	Unchecked Route 1	Needs Command 1	Command Queue 1	Last Command Received 1
⋮								
Robot i	Model Update Time i	Position i	Next Vertex i	Destination i	Unchecked Route i	Needs Command i	Command Queue i	Last Command Received i
⋮								
Robot N	Model Update Time N	Position N	Next Vertex N	Destination N	Unchecked Route N	Needs Command N	Command Queue N	Last Command Received N

Route Planning

As with distributed control with communication, mixed mode autonomy allows each robot to plan its own route using a modified Dijkstra's Algorithm that is described in section 3.3 with the information in their world models. The difference between mixed mode autonomy and distributed control with communication is that a central controller also calculates a route for the robot based on the information shared. 52

The centralized controller uses the same algorithm as the individual robots, but with more information. If the central controller calculates a better route, it will send a command message to the robot. If the routes are the same or equivalent, the central controller will do nothing. This gives the robots the independence of distributed control with the oversight of a central controller.

Route Execution and Collision Avoidance

The route execution and collision avoidance is largely the same as with the other controllers. Each robot sets its velocity to V_i in the direction of their next vertex or zero if another robot is detected within the safe separation distance. The velocity for robot i is

If $d_{obj} > separation\ distance$

$$v_i = \frac{(x_{nv,i} - x_i)\hat{\imath} + (y_{nv,i} - y_i)\hat{\jmath}}{\sqrt{(x_{nv,i} - x_i)^2 + (y_{nv,i} - y_i)^2}} \times v_{avg}$$

If $d_{obj} \leq separation\ distance$

$$v_i = 0$$

where $(x_{nv,i}, y_{nv,i})$ is location of the next vertex of robot i, (x_i, y_i) is the current location of robot I, and v_{avg} is the standard robot speed. $(x_{nv,i}, y_{nv,i})$ is updated by each robot when it reaches an intersection. The next vertex is calculated by the individual robots, but can be overridden by the central controller when it finds a better solution. Because the robots use routes they calculate themselves as the default, there is no dependence on the central controller command message like there is with centralized control.

Communication Management

Each of the robots sends status updates over the communication network. Both the central controller and the other robots can receive these messages. The robot status updates include their position on the road network, their next planned vertex, and their task assignment. The messages for mixed mode autonomy are listed in Table 3-9.

Table 3-9. Messages for Mixed Mode Autonomy

Robot i Message			
Robot i address	Location i	Destination i	Next Vertex i
Central Message			
Robot address	Retransmit flag	Time Stamp	Next Vertex

The robot status messages are not sent at a constant interval – the update rate is based on how likely their information will affect any other robot's route planning or that the message would fail to be received by the central controller. The nominal lower update rate is one third of what is used with the other controllers. As the number of neighboring robots increase or when the distance to the central controller gets large (causes a higher packet error rate), the update rate is increased. A robot is considered to be close if it is within six road segments. Robots are considered far from the central controller when the nominal packet error rate increases beyond 5%. The update rate used by the robot is the greater of the two. The equation for the status message update rate is

$$Update\ Factor\ for\ Robot\ i = min\left\langle 2 \middle| \sum_{j=1}^{N} d_{ij} \leq 6\ segments \middle| d_{i,AP} > 20\ segments \right\rangle$$

$$Update\ Rate\ for\ Robot\ i = (Update\ Factor\ for\ Robot\ i + 1) \times Nominal\ Update\ Rate$$

where d_{ij} world model estimate of the distance between robot i and j, $d_{i,AP}$ is the world model estimate of the distance between robot i and the access point for the central controller, and the nominal update rate is the minimum rate status messages are sent.

The realistic bounds of information requirements for mixed mode autonomy are shown in Figure 3-7. The lower limit assumes robots send messages at the nominal update rate and the central controller does not send any route commands. The upper bound assumes that all of the robots send status updates at the maximum rate and that 5% of the routes need to be corrected. Typical operation assumes that each of the robots is either close to another robot or is far from the access point for the central controller and that 5% of the routes need to be corrected.

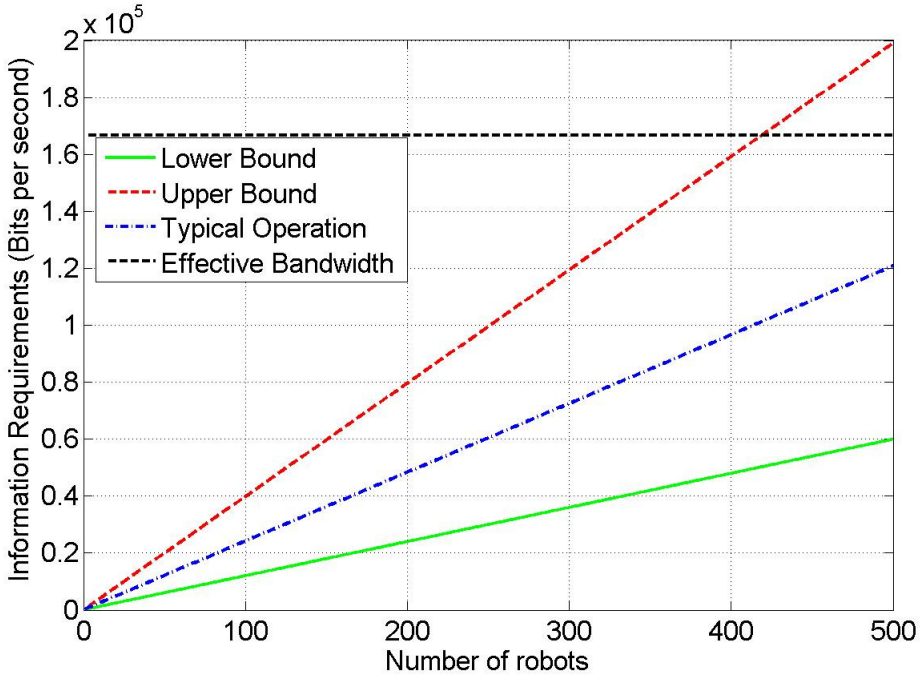


Figure 3-7. Information Requirements for Mixed Mode Autonomy

Summary

Mixed mode autonomy combines centralized and distributed control methods into a single multi-robot control architecture. Mixed mode autonomy distributes the decision making across the robots, while maintaining a central control presence in the system. A graphic summary of how mixed mode autonomy would work is shown in Figure 3-8. The fully shaded tasks are done by

the central controller. The partially shaded tasks are done by both the central controller and the robots. The rest of the tasks are done by the robots.

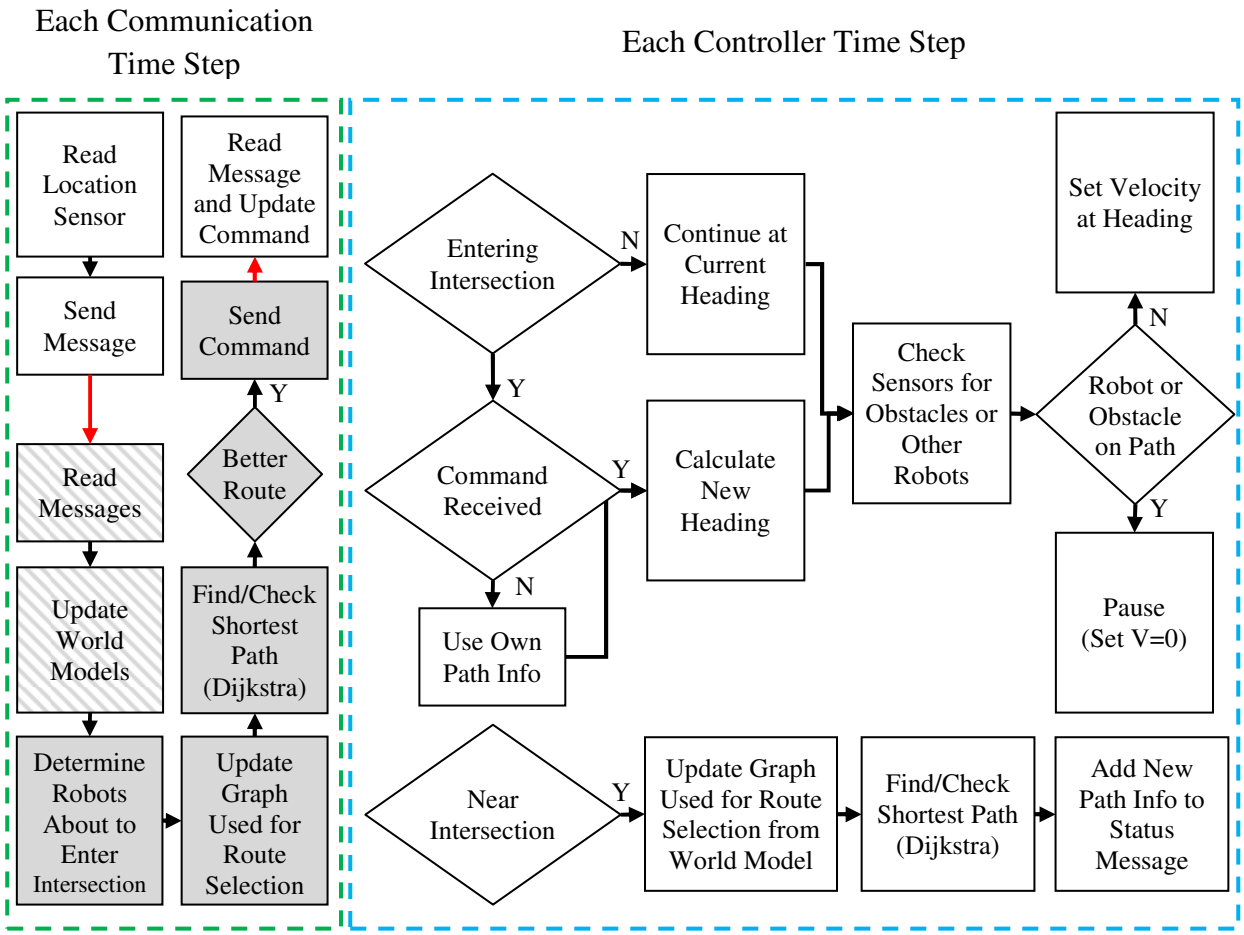


Figure 3-8. Mixed Mode Autonomy

3.6 Mixed Mode Autonomy with Model Updates

Mixed mode autonomy with model updates is an extension of mixed mode autonomy. In addition to the normal status updates, the central controller and robots can also send messages to update their world models if they are out of date.

World Model

The world models are the same as with the base mixed mode autonomy. The difference is that each robot and the central controller all keep track of when their information is out of date. The goal of the updates is to improve the route planning while keeping the communication bandwidth generally lower than the distributed and centralized controllers. Information in the model is old when the robot would no longer be on the same road segment. The world models are summarized in Table 3-10 and 3-11.

Table 3-10. World Model kept by Robot i for Mixed Mode Autonomy with Model Updates

Robot 1	Position 1	Next Vertex 1	Destination 1	Last Update Received 1	Data Current Flag 1
⋮					
Robot i (own information)	Position i	Next Vertex i	Destination i	Distance to closest obstacle	
⋮					
Robot N	Position N	Next Vertex N	Destination N	Last Update Received N	Data Current Flag N

Table 3-11. World Model for Centralized Controller for Mixed Mode Autonomy with Model Updates

Robot 1	Model Update Time 1	Position 1	Next Vertex 1	Destination 1	Unchecked Route 1	Needs Command 1	Command Queue 1	Data Current Flag 1
⋮								
Robot i	Model Update Time i	Position i	Next Vertex i	Destination i	Unchecked Route i	Needs Command i	Command Queue i	Data Current Flag i
⋮								
Robot N	Model Update Time N	Position N	Next Vertex N	Destination N	Unchecked Route N	Needs Command N	Command Queue N	Data Current Flag N

Route Planning

The route planning algorithm is the same for both variations of mixed mode autonomy. The difference is that the robots and central controller can share additional updates to keep the world models up to date.

Route Execution and Collision Avoidance

Both variations of mixed mode autonomy use the same route execution and collision avoidance behaviors.

Communication Management

In addition to the status updates, the robots and central controller can send information about other robots. These messages include the same information as the status updates (position, task assignment, and next vertex) as well as the time the robot or central controller received the message. Updates are requested when information about a robot is out of date and the robot is possibly close enough to influence routing. For the robots, these messages allow the world models to be updated when neighboring robots have communication obstructions between them.

For the central controller, the messages allow it to update its world model when robots move out of reliable communication range by getting information from its neighbors who are closer to the central controller. The messages sent with mixed mode autonomy are shown in Table 3-12.

Table 3-12. Messages for Mixed Mode Autonomy with Model Updates

Robot i Message				
Robot i address		Location i	Destination i	Next Vertex i
Central Command Message				
Robot address		Retransmit flag	Time Stamp	Next Vertex
Update Message – sent by robot or central controller				
Update from ID	Robot info ID	Location i	Destination i	Next Vertex i

The realistic bounds of information requirements for mixed mode autonomy are shown in Figure 3-9. The lower limit assumes robots send messages at the nominal update rate and the central controller does not send any route commands. The upper bound assumes that all of the robots send status updates at the maximum rate, 5% of the routes need to be corrected, and 5% of the robot models require an update. Typical operation assumes that each of the robots is either close to another robot or is far from the access point for the central controller, 5% of the routes need to be corrected, and 5% of the robot models need to be updated. The upper bound for this case is the highest of all of the controllers, but is unlikely to ever occur.

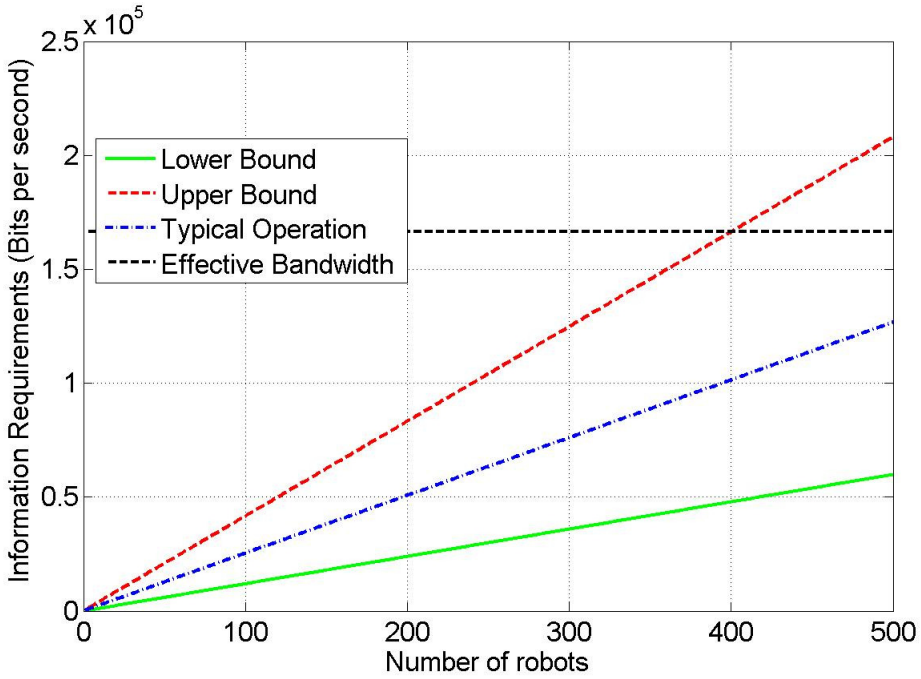


Figure 3-9. Information Requirements for Mixed Mode Autonomy with Model Updates

Summary

Mixed mode autonomy with model updates builds off of the mixed mode autonomy architecture. The model update requests and neighbor status updates help keep the robot and central controller world models current when communication is not as reliable. A graphic summary of how mixed mode autonomy with model updates would work is shown in Figure 3-10.

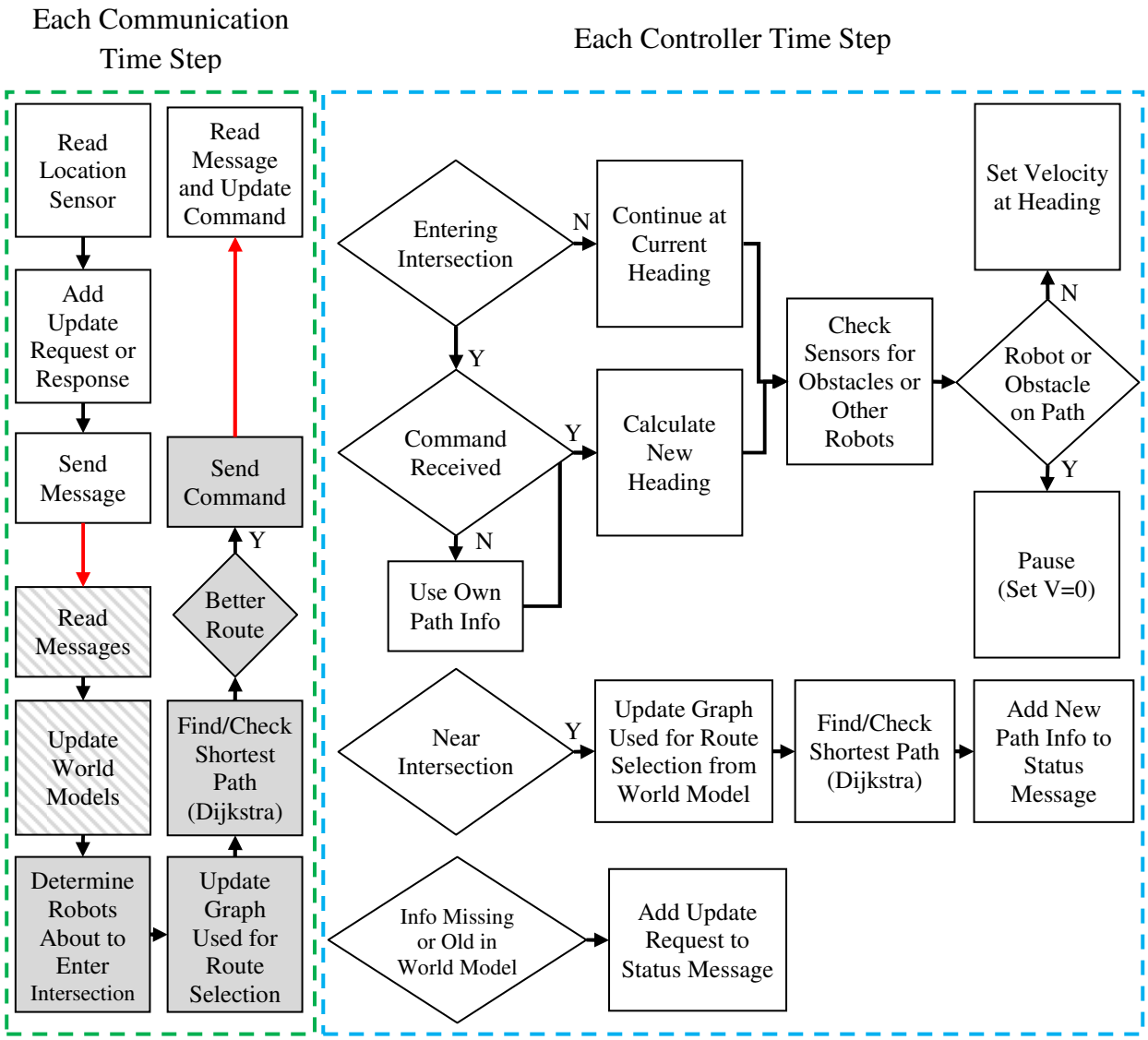


Figure 3-10. Mixed Mode Autonomy with Model Updates

3.7 Comparison of the Multi-Robot Controllers

This section summarizes the key similarities and differences of the four controllers used to solve the dynamic multi-robot autonomous routing problem. Each of the controllers uses largely the

same methods to execute their routes and avoid collisions. The primary differences are with the way route decisions are made and how communication to handle world model updates is done.

Table 3-13 lists how each of the major components of multi-robot control is divided between a centralized controller and the individual robots for all of the controllers.

Table 3-13. Summary of the Multi-Robot Control Methods

Method	**Central Controller**	**Robot**	**Communication**
Distributed Control without Communication	None	Plan route based on road network, Avoid collisions based on sensor data	None
Centralized Control	• Plan routes • Track location of all robots • Confirm route execution	Avoid collisions based on sensor data	Central – Commands for segments of route Robots – Update of position, commands received
Distributed Control with Communication	None	• Plan routes • Track location of all robots • Avoid collisions based on sensor data	Robots – Update of position, commands received
Mixed Mode Autonomy	• Track location of all robots • Check routes planned by robots • Send and confirm routes when it finds better one	• Plan routes • Track location of all robots • Avoid collisions based on sensor data	Robots – Update of position, commands received Central – Commands for segments of route (when better)
Mixed Mode Autonomy with Model Updates	• Track location of all robots • Check routes planned by robots • Send and confirm routes when it finds better one • Update information on robots	• Plan routes • Track location of all robots • Avoid collisions based on sensor data • Update information on their peers	Robots – Update of position, commands received Central – Commands for segments of route (when better) Both/Either – Status updates

Figure 3-11 compares the communication bounds for each of the control methods. The effective bandwidth requirement is the amount of data that can be transmitted on the wireless channel after the packet overhead is subtracted. The effective bandwidth on these plots is for the high overhead case. The information requirements are all well below the effective bandwidth for the

low overhead case. All of the methods assume 5% retransmission of commands. The lower bounds are the minimum for the minimum number of messages that would be sent if the tasks were all completed in their minimum time. The real tasks would likely take longer due to the time required to handle congestion and false positive sensor readings. This would reduce the number of commands that need to be sent for the central and mixed mode autonomy controllers. The upper bounds for central control and mixed mode autonomy are the same. Mixed mode autonomy, however, will very rarely require all of the commands to be sent like centralized control. The lower bounds for the both kinds of mixed mode autonomy are the same.

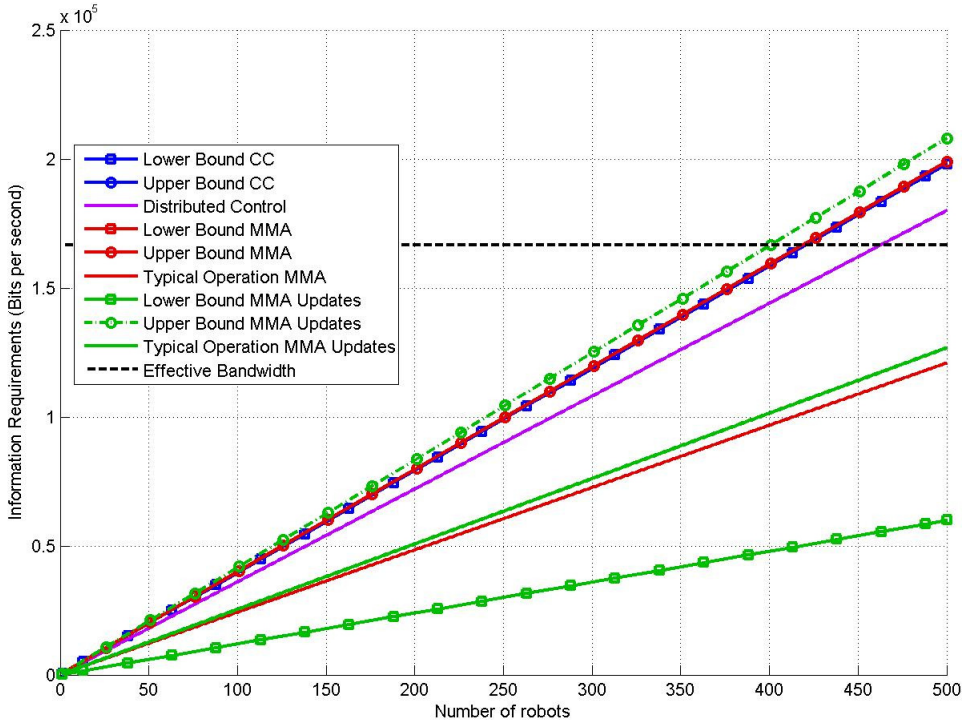


Figure 3-11. Information Requirements for Dynamic Multi-Robot Autonomous Routing

Figure 3-12 compares the typical operation of each of the controllers. Both centralized control and distributed control with communication exceed the effective bandwidth in the high overhead case. The difference is that latency in the central command messages will result in idle robots instead of incomplete information for route planning with the distributed control. The mixed mode autonomy methods both have plenty of available bandwidth for operation.

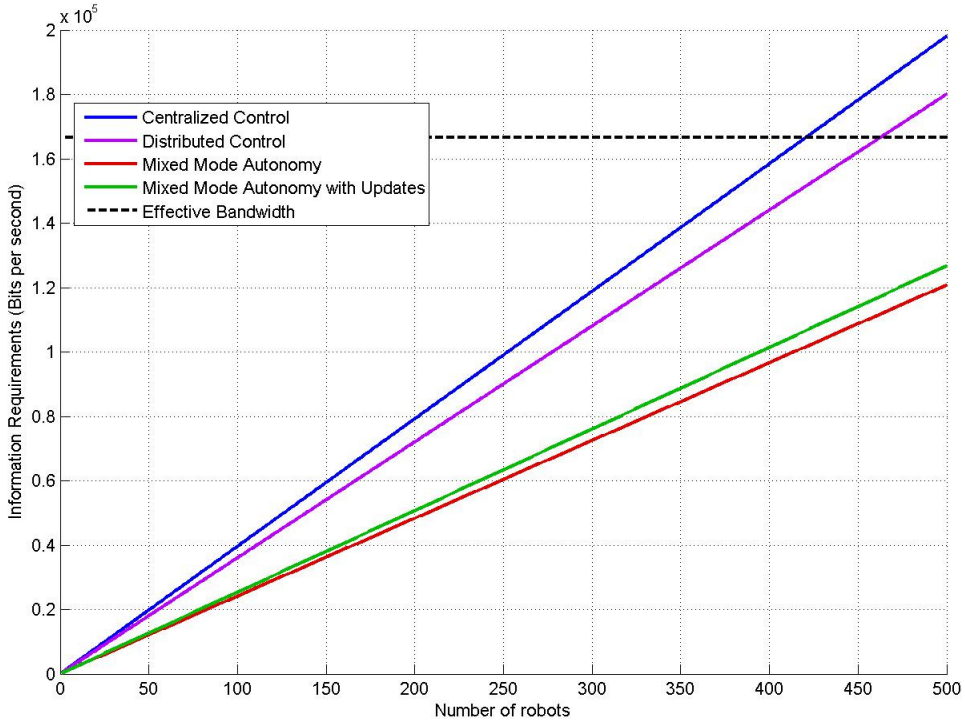


Figure 3-12. Typical Information Requirements for Dynamic Multi-Robot Autonomous Routing

Figure 3-13 shows the realistic worst case information requirements for the different controllers as a function of the number of robots. This graph is for over one controller time interval. The worst case is when all of the robots that require a command to update their route need it at the same time. The mixed mode autonomy controllers assume that 5% of the routes found by the robot will need correction by the central controller. This is an indicator of how likely the controller will have latency problems.

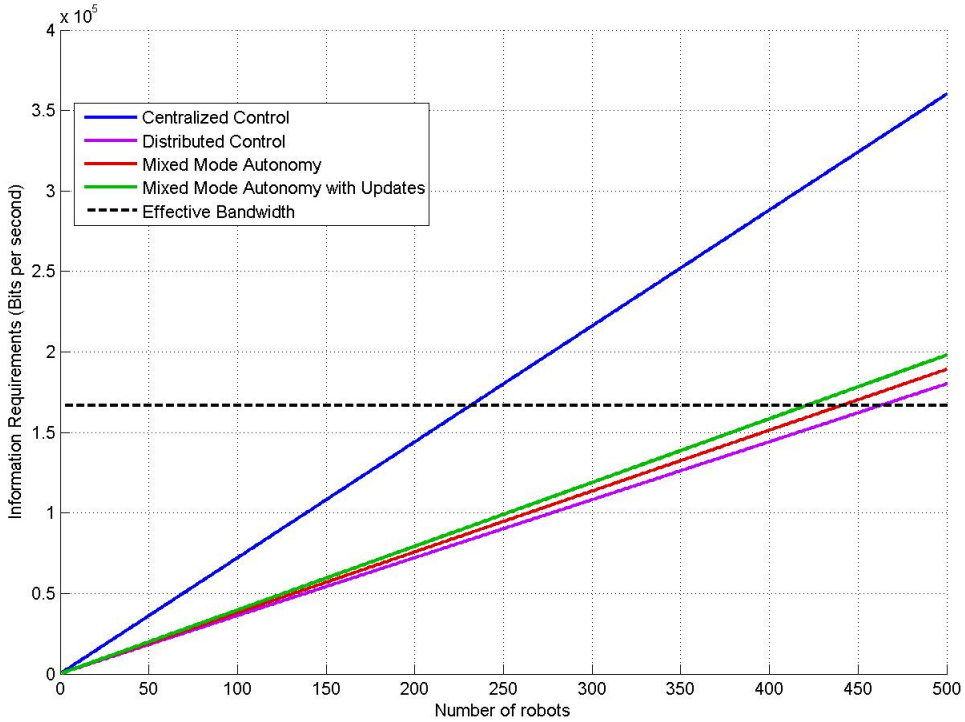


Figure 3-13. Realistic Worst Case Information Requirements for the Different Controllers

Chapter 4 – Multi-Robot Simulation Model

To evaluate the different multi-robot controllers presented in Chapter 3, several sets of simulations of the dynamic multi-robot autonomous routing problem were run. For these simulations, a detailed model of the problem domain/environment was needed. The simulation model includes the multi-robot environment, a wireless communication system, the robot capabilities, and the multi-robot controllers. The simulations log data to evaluate the performance of the controllers. Monte Carlo simulation is used to find the average performance for a set of operating conditions. Due to the large number of simulations required for Monte Carlo simulation and the size of the models for large numbers of robots, the simulations were simplified to reduce computation time. The goal was to preserve the behavior and effects related to each controller, while reducing the time to run each simulation.

The multi-robot environment for dynamic multi-robot autonomous routing is a network of roads. Robots are constrained to travel on roads. For the general formulation of the multi-robot controllers, the road networks can be any arbitrary graph with vertices located at any 3-D point. To reduce the complexity of simulating the environments for this work, they are two-dimensional uniform grids. Changes in elevation increase the difficulty of modeling the wireless communication system and would require significantly more different environments to be simulated to ensure that the performance measured was not specific to the environments tested. The uniform grid simplifies all of the distance calculations in the model, which reduces the simulation run time by almost two thirds. These generalizations of the environment are strictly for ease of simulation and not limitations of the controller.

The communication model estimates the likelihood that transmitted messages will be received and models the message latency (wireless communication channel usage). The model is based off of values from empirical models of wireless communication systems that operate in typical environments for the example applications (warehouse automation, container terminal automation, and open pit mine automation). To model the likelihood that a message is received the packet error rate (PER) is calculated. The PER is a function of the signal to noise ratio (SNR) and the encoding method used by the transmitter. The received signal strength model includes the path loss, shadowing, and fading effects. Several different communication configurations can be modeled to determine the limitations and costs associated with each method and determine the appropriate method for a control scheme. The model uses the mean signal strength and the standard deviations of the slow (shadowing) and fast (fading) effects to calculate the value of the received signal strength (RSS). These statistics describe the typical signal and are used to calculate a signal strength value at the receiver. The signal strength is used to get SNR and then packet error rate (PER). PER models the probability that a transmitted packet is received. In addition to determining whether messages are successfully received, the communication model simulates the timing of the messages. Latency in the messages can have significant impact on the multi-robot controller performance – particularly the command messages. Only one message can be sent at a time on the wireless communication channel. The communication time period for the simulator is the time required to send one message. A simulation message queue is used to mimic the timing of a real wireless network. Messages are sent one at a time from the queue and the PER is calculate for each message to determine which receivers actually got the information.

The robot capability model is limited to the robot mobility and sensing. The specifics of things like motion control and planning are abstracted to focus the work on the high level control. The robots are treated as a point with a maximum speed and a sensor range. The robot heading is limited by the requirement that the robots remain on the roads.

Each of the different controllers discussed in Chapter 3 is modeled for the simulations. These controllers are distributed control without communication, centralized control, distributed control with communication, mixed mode autonomy, and mixed mode autonomy with model updates. The controller functions include world modeling, route planning, route execution, collision avoidance, and communication management. Each of the controllers is a different split of these functions between a centralized controller and control distributed across the multi-robot system.

The performance data for each individual simulation is logged so that the trends can be found. For most of the variables of interest only the totals or averages and standard deviations are logged because the data storage requirements would become quite large for the simulations with a large number of robots. The evaluation criteria are used to assess the effectiveness of the controllers for dynamic multi-robot autonomous routing and the communication costs over a range of operating conditions. The simulation results are also used to predict performance for the example applications and recommend an appropriate controller for each.

The full code for implementation of the simulation of the five controllers (distributed control with no communication, centralized control, distributed control with communication, mixed mode autonomy, and mixed mode autonomy with model updates) is included in the Appendix.

4.1 Simulation Structure

The simulations are to evaluate the different approaches to control for dynamic multi-robot autonomous routing across a variety of system and environment parameters. To diminish the effects of specific initial conditions or task lists Monte Carlo simulations are used. Simulation of the controllers is divided into two parts – simulation of the robot motion and control and simulation of the wireless communication system.

Simulation Loop

Each of the five controllers implemented uses the same loop structure for simulation. The initialization and motion parts of the loop are the same for all of the controllers. Each of the controllers has different control and communication algorithms. The simulation loop structure is shown in Figure 4-1. Initialization sets up the multi-robot environment and parameters for a set of Monte Carlo simulations. Control plans and checks routes and calculates the motion of the robot for the simulation time step. Motion updates the position of the robots in the simulation based on the robot motion calculated in control and the current collision avoidance sensor readings. Communication selects the messages to be sent by the robots and any central controller, simulates the wireless transmission, and updates the world models for the next iteration of the simulator.

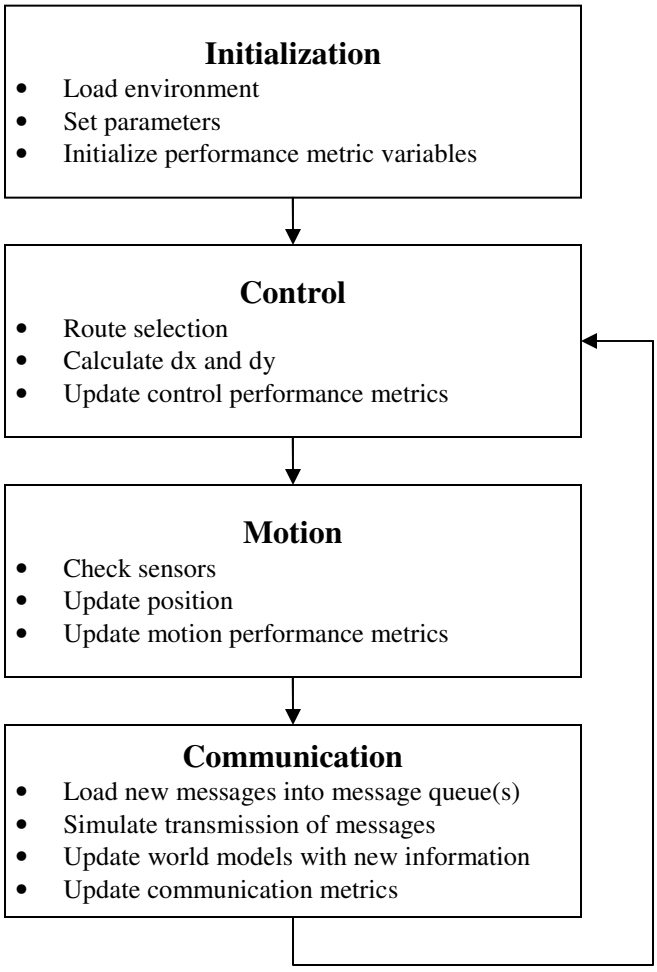


Figure 4-1. Simulation Loop

Simulation Timing

The simulations are each 2000 seconds long. Based on the environment size and robot speed, the average robot should complete 25 tasks (trips between two intersections) under ideal conditions in this time. The simulations operate at a fixed time step between updates. The time step is set so that the simulation will update the motion and communication of the robots at least ten times per road segment. The simulation update time is thus the distance between two orthogonal vertices divided by the robot velocity divided by ten. This was found to be a good balance between simulation resolution and simulation runtime. The simulation results were not significantly different (task completion times, robot usage rates, bandwidth usage, etc. converged to the same value) with a higher rate of updates and the runtime was linearly related with the number of updates.

4.2 Simulation Parameters

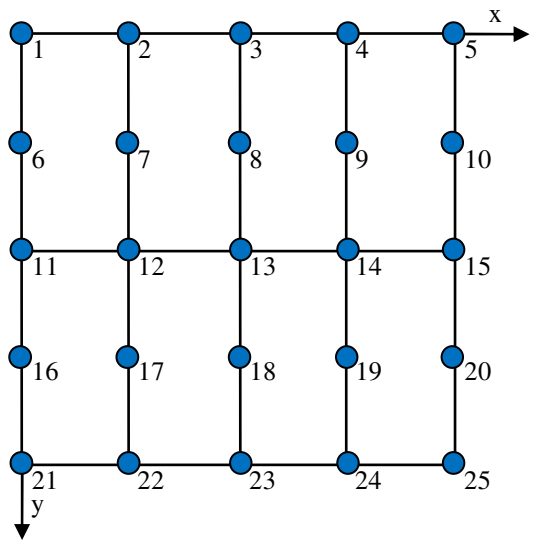
To assess the controllers for the typical operating conditions of the three example applications, the simulation were run over a range of environmental, task, and communication parameters. All of the parameters were scaled relative to the environment size. A set of simulations is 20 trials for the same parameter values. Twenty was the minimum number of simulations required to see statistically significant differences in the Monte Carlo simulation results. The parameters changed are the multi-robot environment, task coupling, communication range, message overhead, and the number of robots. The environments are characterized based on the grid shape and the bottleneck width. The task coupling describes how tightly grouped the tasks are about the two task centers in the environment. The communication range is set based on the transmit power of the wireless communication system. The message overhead is the part of a packet other than the actual content of the message (headers, addresses, checksums, etc.). The number of robots is varied from one to 500. The one robot case was a baseline to verify that all of the controllers were using the right set of values. The parameters for the Monte Carlo simulations are summarized in Table 4-1.

Table 4-1. Simulation Parameters

Parameter	Class	Values	Description
Control Type	Control	Distributed without Comm., Centralized, Distributed with Comm., Mixed Mode Autonomy, Mixed Mode Autonomy with Model Updates	These are the different multi-robot controllers implemented. Mixed mode autonomy is the new control architecture developed as part of this work.
Number of Robots	Robot	1, 10, 20, 50, 100, 200, 300, 400, 500	The performance was evaluated to determine how well the MRS scaled with increasing size of the group.
Grid Shape	Env.	1, 2, 4	This is connection pattern for the environment – higher numbers are sparser environments.
Bottleneck	Env.	4, full	The width of the restriction (narrowest part) of the environment. Creates an area of congestion when it is small.
Task Coupling (pair)	Task	Uniform-Uniform, Uniform-5, Uniform-2.5, 5-5, 5-2.5, 2.5-2.5	Standard deviation (in environment segment lengths) of the distribution of tasks about the two task centers in the environment. The lower the number the tighter the coupling of the tasks.
Overhead	Comm.	48, 240	Bits per message required other than the message. The higher the overhead the lower the effective bandwidth.
Communication Range	Comm.	15, 45 (distance from task center to access point plus 10)	The range at which 90% of the transmitted signals will have a PER of 0.

4.3 Multi-Robot Environment

The multi-robot environment is the task domain for dynamic multi-robot autonomous routing and can be modeled as a network of roads. The task for dynamic multi-robot autonomous routing is for a robot to travel to an assigned intersection. The robots are constrained to travel only on the roads. The network is made up of a set of intersections and lanes of roads. The intersections are represented as vertices and the lanes are segments connecting the vertices. A vertex list describes the location of each intersection of the road segments. A weighted adjacency matrix stores the information about the connection of the vertices. The environments are divided into four quadrants. A task center is located in the center of the second and fourth quadrants (one and three would be equivalent). The vertex assignments for the robots are normally distributed about the task centers. The assigned intersections for the robots (tasks) alternate between the two centers. Three parameters are used to generate the environments – number of vertices, grid shape, and the bottleneck width. The number of vertices is fixed for all of the environments that were used for the simulations for this work. The grid shape is the ratio of the horizontal and vertical segment lengths. The bottleneck is the number of vertices wide at the narrowest part of the environment. The higher the grid shape value the more sparsely connected the network. Figure 4-2a shows a small example of an environment and the equivalent vertex list. Figure 4-2b shows the adjacency matrix for this example. The environments simulated include many more vertices and segments. For the more general case, when the vertices are not placed on a uniform grid, the adjacency matrix could be weighted to account for the different costs (distance, travel time, etc.) of each connection between the vertices.



Intersection	x	y
1	0	0
2	1	0
3	2	0
4	3	0
5	4	0
6	0	1
7	1	1
8	2	1
9	3	1
10	4	1
11	0	2
12	1	2
13	2	2
14	3	2
15	4	2
16	0	3
17	1	3
18	2	3
19	3	3
20	4	3
21	0	4
22	1	4
23	2	4
24	3	4
25	4	4

Figure 4-2a. Example Environment and the Corresponding Vertex List

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0	0	**1**	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	**1**
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0

Figure 4-2b. Corresponding Adjacency Matrix for the Example Environment

4.4 Robot Capabilities

For the dynamic multi-robot autonomous routing problem, the robot model includes two parts – mobility and sensing. For this work, the robots are homogeneous. This is for simplicity of running lots of simulations, not a limitation of the controllers implemented. Each of the controllers can account for robot differences as part of their world models and route planning.

The x and y locations and the velocity in the x and y directions of the robots are stored in arrays. The robots have a maximum velocity V and are constrained to remain on the roads. The velocity was set so that each robot could complete 25 average tasks (the trip between one task center and the other) during the simulation. This was enough tasks during each simulation to get meaningful averages for the metrics of performance for each robot and the MRS as a whole. For simplicity of simulation, the robots travel at their maximum velocity or pause rather than modulate their velocity. Since the environment is a uniform orthogonal grid, the robots move a fixed distance in the x or y direction at each update of the simulation. The step size is

$$\Delta = V \times \Delta t$$

where Δt is the simulation update rate. The steps are always in the x or y direction since the roads align with the x and y axes.

The robots have a simulated sensor suite to detect the presence of objects or other robots in their path. These sensors have a range half of a road segment long. The multi-robot controllers all attempt to maintain a separation distance between robots. The sensor suite is used to verify this distance. Each of the sensors has a false positive rate. False positives for objects at the separation distance or less are the most likely error with the sensors and effectively add the variability in the robot velocities to the simulation (the robots have a 100% - FP% chance of moving Δ during a given update of the simulation). If the robot senses an object in its path within the separation distance, it will not move during that time step. A uniform random number generator is used to determine if a false positive is registered by the sensors. If the random number is less than the false positive rate, the sensor reading is a false positive. The path obstruction state is

$$if\ r > FP\ AND\ d > s$$
$$PO = 0$$

$$if\ r < FP\ OR\ d < s$$
$$PO = 1$$

where a PO of one indicates that an obstruction is present, r is a uniform random number, d is the distance to the closest robot on the same segment, and s is the minimum separation distance. The robot position array is used to find d. If no other robot is on the segment d is set to infinity.

4.5 Wireless Communication

To accurately assess the performance of the controllers a realistic model of the wireless communication systems is required. Dropped and delayed messages reduce the information available to the controllers. The central controller and robots update their world models based on the messages sent through wireless communication. Many of the controllers simulated in literature make overly optimistic assumptions about the communication performance. The wireless communication model must reflect the packet errors and latency of real networks since the controllers will function differently if they do not receive some of the messages or receive the messages late. Two different communication scenarios are used for the controllers – communication between transmitter-receiver pairs and broadcast communication.

Packet Error Rate

Packet error rate (PER) is the percent chance that a receiver would fail to get a transmitted message correctly. The receivers are the other robots and/or the centralized controller. PER is a function of the signal to noise ratio (SNR) and the packet encoding method. The received signal strength is made up of the line-of-sight component plus reflected and scattered terms. The distance between the transmitter and receiver(s) predicts the average signal strength at a location, but the value for a specific signal varies from the average based on the standard deviation terms due to reflection and scattering.

The line-of-sight path loss is inversely related to the distance raised to an exponent. In the ideal model the exponent is 2. In most realistic models the exponent (empirically determined) ranges from 2.4 to 4. The equation for line of sight path loss is

$$PL(d) = PL(d_0) + 10 \times p * \log \frac{d}{d_0}$$

where PL(d_0) is the path at d_0, d is the distance between the transmitter and receiver, and p is the path lost exponent.

The mean received signal strength is the transmitted power plus the transmitter antenna gain minus the line of sight path loss. The received signal will also include components from the scattered and reflected signals. These signal contributions are characterized statistically based on a standard deviation with zero mean.

Scattering effects are often modeled as many reflections. Reflections are product of geometry and material properties of the reflective surface. For this work these effects are represented as mean-zero normally-distributed variations in the distance dependent mean signal strength. The values of the standard deviation terms for scattering and reflections and the path loss exponent are based on averages from several empirical models of networks that operate in environments similar to the example applications. The reflection and scattering components are

$$R = \sigma_r \times N(\sigma = 1, \mu = 0)$$
$$S = \sigma_s \times N(\sigma = 1, \mu = 0)$$

where N(σ=1, μ=0) is a number from a normal distribution with zero mean and standard deviation of one, σ_r is the coefficient for reflection standard deviation, and σ_s is the coefficient for scattering standard deviation.

Ultimately, the received signal strength (RSS) is

$$RSS = P_t - PL - S - R$$

where P_t is the power transmitted in dB, PL is the path loss, S is the scattering term, and R is the reflection term. The signal strength is a calculated at a specific time by substituting numbers from the statistical distributions for the scattering and reflection terms. The signal to noise ratio (SNR) is

$$SNR = RSS - P_n$$

where the thermal noise Pn is

$$P_n = 10 \times \log[(F + 1) \times k \times T_0 \times B_N \times 1000]$$

where F is the noise factor, k is the Boltzmann's constant, T_0 is the temperature, and B_N is the noise bandwidth.

From the calculation of signal strength at one time and place the packet error rate (PER) is estimated. The PER is a function of signal strength and the modulation method used. From the PER the model determines if individual packets arrive or fail using a uniform random number compared to PER. Binary Phase Shift Key (BPSK) encoding was assumed. This is a common method for many of the spread spectrum networks that are often used in the kind of environments of the example applications. The equation for bit error rate (BER) is

$$BER = Q\left(\sqrt{2 \times 10^{\frac{SNR}{10}} \times \frac{B_N}{DR}}\right)$$

where the Q-function is the tail probability of the normal distribution and DR is the data rate. SNR is in dB for this form of the equation.

The equation for PER is

$$PER = 1 - (1 - BER)^{FL}$$

where FL is the frame length in bits.

Wireless Communication Range/Coverage

The average PER is a function of the distance, transmit power, message length, and packet encoding. Figures 4.3 and 4.4 are maps of the typical PER values for an example environment with the transmitter at the center of the environment (25, 25). Figure 4.3 is the longer range communication distance. Figure 4.4 is the shorter range communication distance. The 90% worst case scenarios are for signals that are at the worst 10% once reflection and scattering are added. The areas in blue are the lowest PER (message unlikely to fail). The areas in red are lower PER (messages more likely to fail). The lowest values are still above 10%, meaning most of the messages are likely to be received at any point in the environment. Realistically, a wireless communication system would be unlikely to be used if the PER values were much above 10% at the desired range.

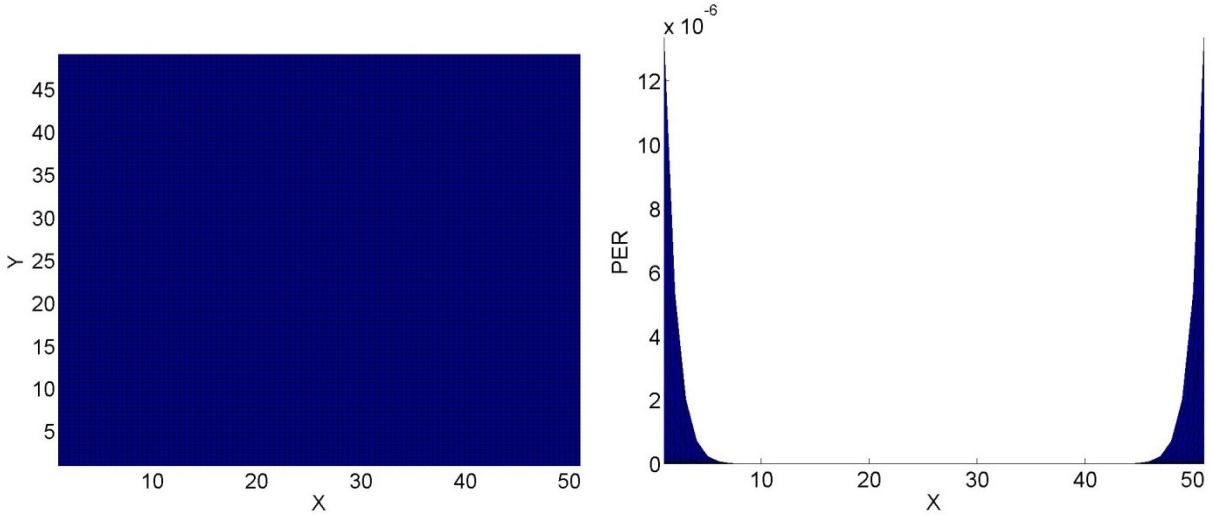


Figure 4-3a. Communication Coverage – Long Range, Nominal Signal

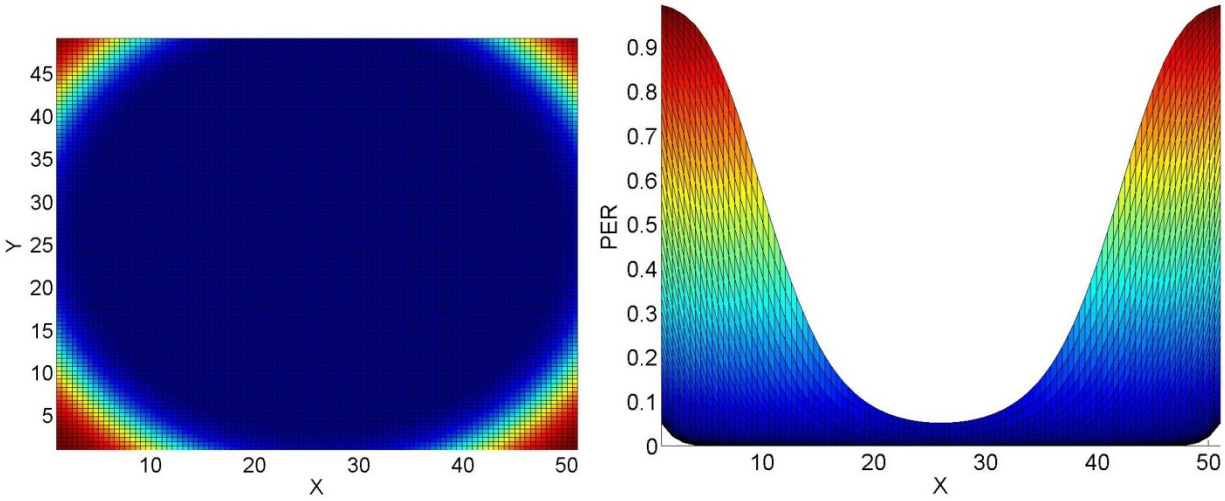


Figure 4-3b. Communication Coverage – Long Range, 90% Worst Case

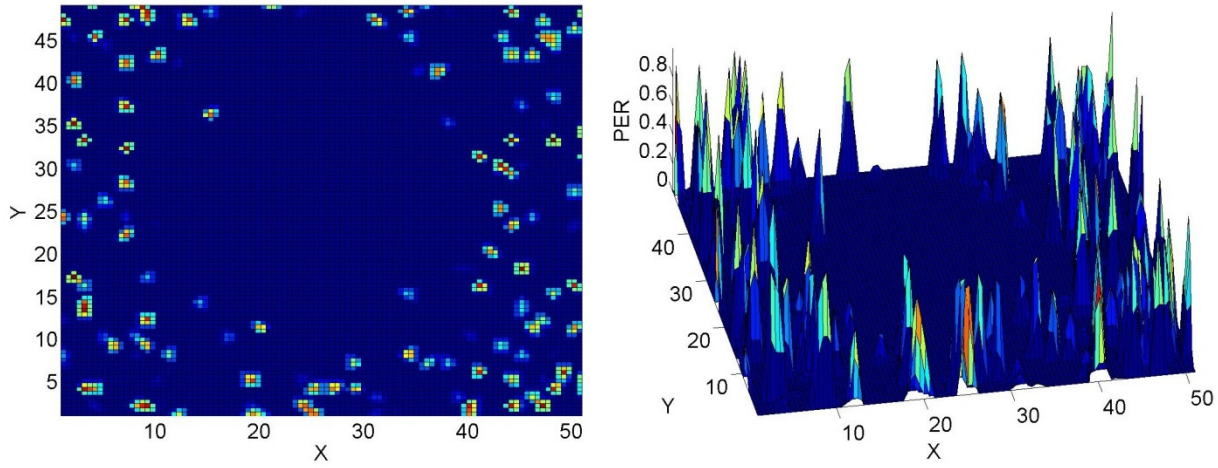


Figure 4-3c. Communication Coverage – Long Range, Values for one iteration of simulation

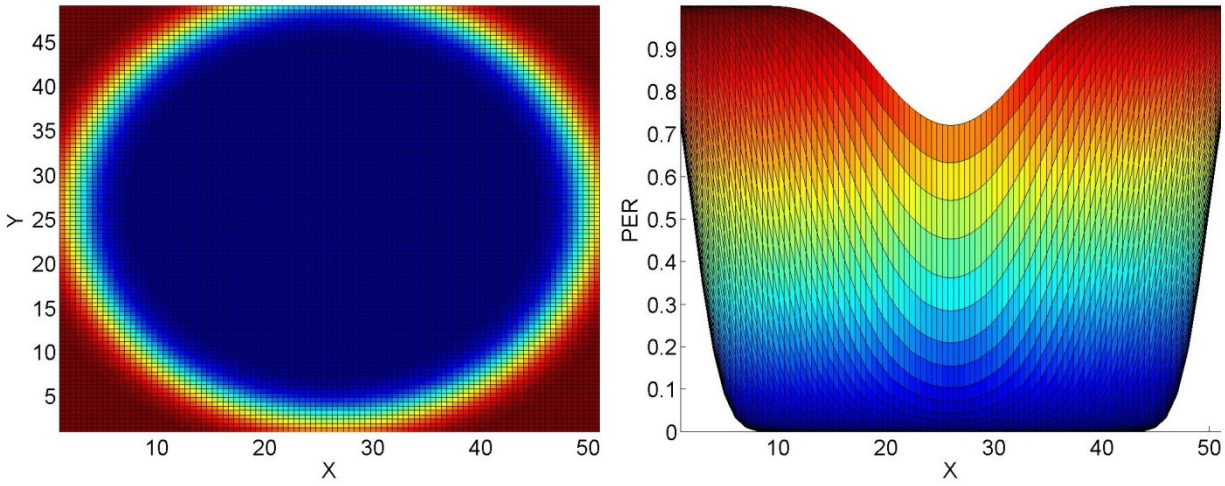


Figure 4-4a. Communication Coverage – Short Range, Nominal Signal

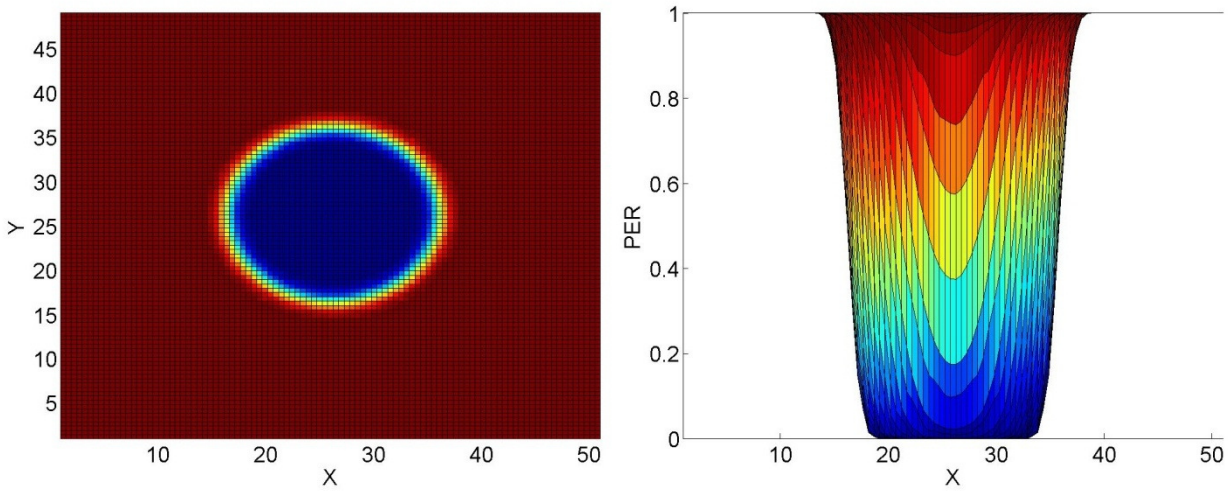


Figure 4-4b. Communication Coverage – Short Range, 90% Worst Case

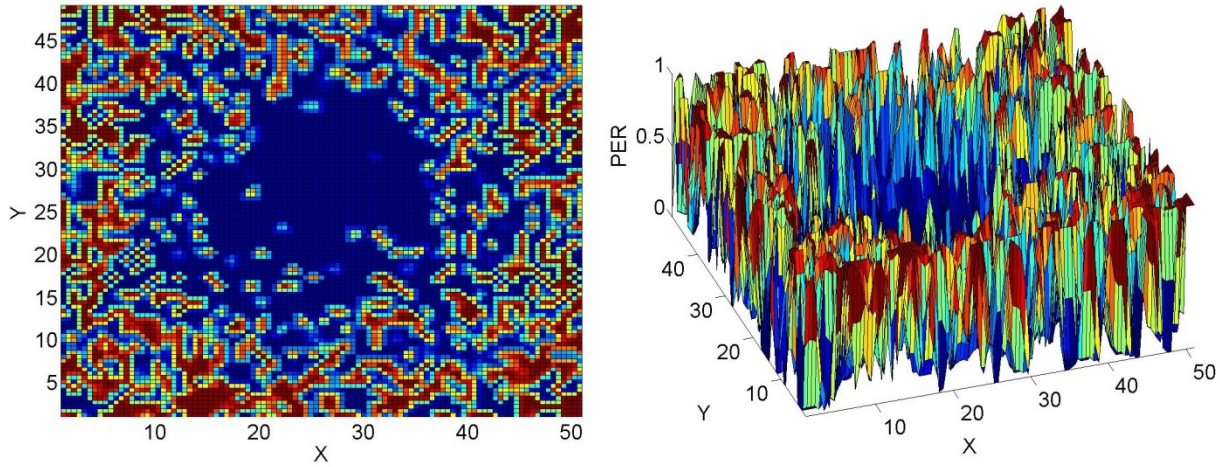


Figure 4-4c. Communication Coverage – Short Range, Values for one iteration of simulation

Message Latency

Message latency is represented with a model of the wireless communication channel usage. Only one message can be sent on the wireless network at a time. To simulate the wireless communication channel usage, two message queues are used. The central controller and the robots each load messages into a queue. Whenever the channel is idle, a message is selected to be transmitted from these queues. The central controller queue is given priority over the robot queue. These queues allow the simulations to account for the timing of message transmissions.

When a message is transmitted, the simulator evaluates if the message was successfully received. The PER value for the relevant transmitter-receiver pair is compared to a uniform random number. If the random number is greater than the PER, the message is successfully transmitted and the information in the message is available to the controller to use. When the random

number is lower than PER, the message fails. The message will not be retransmitted unless it is reloaded into the message queue. The message queue structure is not the way the wireless communication network actually sends messages. It is a way to simulate the channel usage. The queues are used to simulate both peer-to-peer and broadcast communications. The central controller messages are evaluated to determine if the target robot received the messages. The robot messages can be evaluated to see if the central controller received the message and/or if the other robots got the message.

4.6 Multi-Robot Controllers

This portion of the model implements the multi-robot controllers for both the robots and a central controller (if present in the system). The parts of the controllers include world models, route planning, route execution and collision avoidance, and communication management. The controllers are simulated as part of the motion loop. The general structure for the controllers is in Figure 4-5. The rest of this section describes the implementation for the simulations of the controllers (distributed control without communication, centralized control, distributed control with communication, mixed mode autonomy, and mixed mode autonomy with model updates.

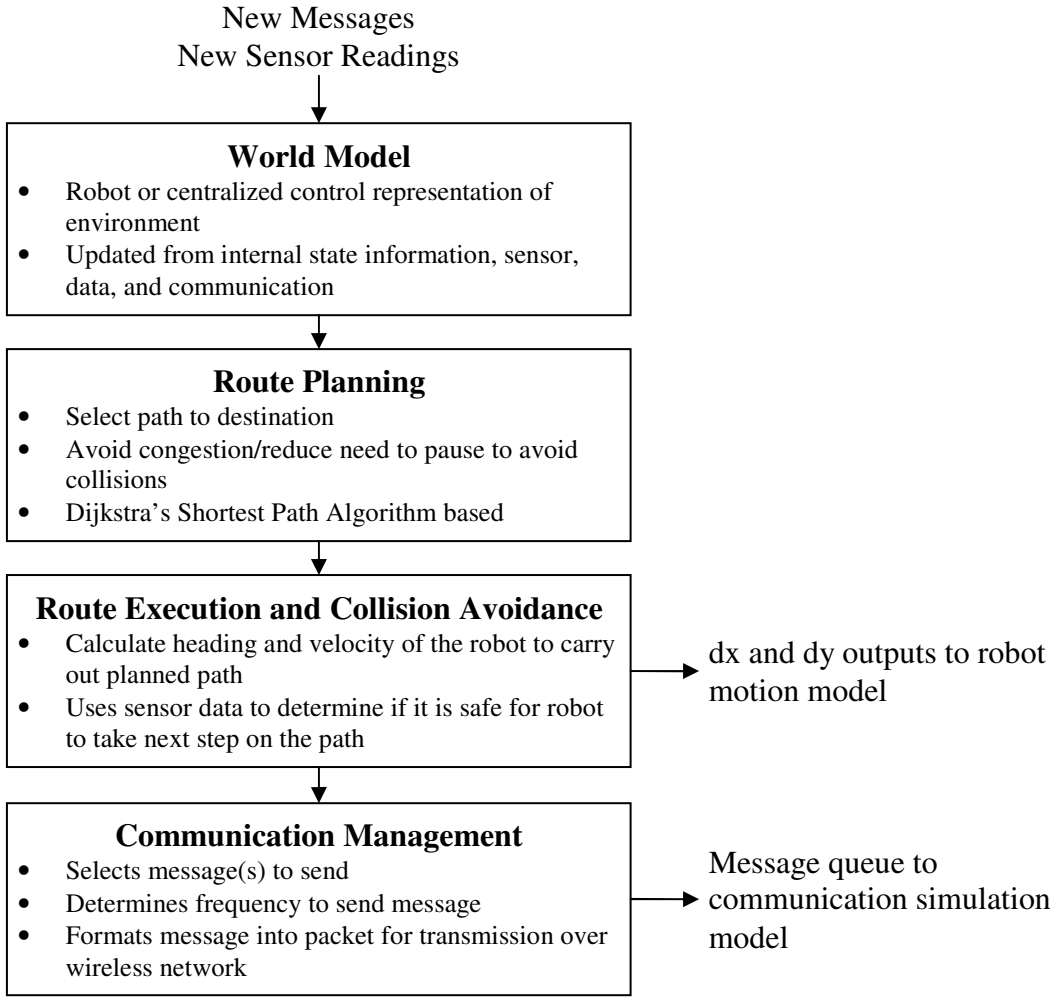


Figure 4-5. General Structure for the Controller Simulation

World Models

The world models are the robots' and central controller's representations of the environment in which they operate and the robots in the MRS. The world models are based on three sources of information – internal state information (what the robot or central controller knows about its own state), sensor data, and communication. The route planner uses the information in the world model combined with the map of the network of roads to select the robot's paths. The distributed controllers keep N world models – one for each of the N robots. The centralized controller keeps one model of the entire MRS. The mixed mode autonomy controllers keep the N world models for each robot and a world model for the centralized controller.

There are two types of data in the world model – location and robot state information and task information. For the robot world models, this data can also be divided into a robot's own information and information about its peers in the MRS.

The sensor data for the robots is position and range to an obstacle. The robots track this information for all of the controllers. The simulator gets the location data from the motion update part of the loop. The simulated sensor data comes from the distance calculations (also used for communication simulation) with a false positive rate (2%).

The location information is generated by the motion update loop. Robots can directly sense their own location in the environment. For all of the controllers except the distributed control without communication, robot share updates of their location. The distributed controller with communication and local robot controller for mixed mode autonomy give position updates to their peers. The centralized controller gets position updates from all of the robots.

The world models keep track of the task information for the robots. The primary task information is the next intersection on the robot's path and the assigned destination. All of the robots know their own task information for all of the controllers. The centralized controller knows the next intersection and destination for all robots since it responsible for making both decisions. For distributed controllers, the robots have a list of tasks a priori. The centralized controllers send tasks as part of the commands.

Communication updates share information between other robots or the centralized controller. They are different for each controller. Distributed control without communication does not get any updates over communication – it is completely sensor based. The rest of the controller use a set of messages to share information to build world models at the robot or centralized controller level.

The information included in the world models, the source of the information, and the part of the simulation used to generate the data is summarized in Table 4-2

Table 4-2a. World Models Summary – Own Information

Control Architecture	Position	Next Intersection	Destination	Obstacle State	Needs Next Intersection
Distributed Control with No Communication – Robot Model	Location sensor	Own route calculation	Pre-assigned	Sensor reading	Distance to intersection
	Motion Update Loop	Dijkstra solution of road network	Simulation Initialization	Distance calculations in Motion Update Loop/Simulated False Positive	Distance calculations in Motion Update Loop
Distributed Control with Communication – Robot Model	Location sensor	Own route calculation	Pre-assigned	Sensor reading	Distance to intersection
	Motion Update Loop	Dijkstra solution of road network with other robot information	Simulation Initialization	Distance calculations in Motion Update Loop/Simulated False Positive	Distance calculations in Motion Update Loop
Centralized Control – Central Control Model	NA				
Mixed Mode Autonomy Control – Robot Model	Location sensor	Own route calculation	Pre-assigned	Sensor reading	Distance to intersection
	Motion Update Loop	Dijkstra solution of road network with other robot information	Simulation Initialization	Distance calculations in Motion Update Loop/Simulated False Positive	Distance calculations in Motion Update Loop
Mixed Mode Autonomy Control – Central Control Model	NA				
Mixed Mode Autonomy Control with Model Updates – Robot Model	Location sensor	Own route calculation	Pre-assigned	Sensor reading	Distance to intersection
	Motion Update Loop	Dijkstra solution of road network with other robot information	Simulation Initialization	Distance calculations in Motion Update Loop/Simulated False Positive	Distance calculations in Motion Update Loop
Mixed Mode Autonomy Control with Model Updates – Central Control Model	NA				

Table 4-2b. World Models Summary – Other Robot Information

Control Architecture	Position	Next Intersection	Destination	Update time	Data Current/ Needs Update	New Route
Distributed Control with No Communication –	NA					

Robot Model						
Distributed Control with Communication – Robot Model	Robot location sensors / Update Messages	Robot route solutions / Update Messages	Robot route solutions / Update Messages	Update Message Time	Would robot reach intersection in time since last update	NA
	Comm. simulation of message queue	Comm. simulation of message queue	Comm. simulation of message queue	Time when message queue is read	Distance to intersection at update time < VΔt	NA
Centralized Control – Central Control Model	Robot location sensors / Update Messages	Central route solution	Assigned by central	Update Message Time	Would robot reach intersection in time since last update	Use location data to find if Robot is close to intersection?
	Comm. simulation message queue	Dijkstra solution of road network with other robot information	Simulation Initialization	Time when message queue is read	Distance to intersection at update time < VΔt	Set flag to calculate new route
Mixed Mode Autonomy Control – Robot Model	Robot location sensors / Update Messages	Robot route solutions / Update Messages	Robot route solutions / Update Messages	Update Message Time	Would robot reach intersection in time since last update	Flag set when robot calculates new route
	Comm. simulation of message queue	Comm. Simulation of message queue	Comm. simulation of message queue	Time when message queue is read	Distance to intersection at update time < VΔt	Comm. simulation of message queue
Mixed Mode Autonomy Control – Central Control Model	Robot location sensors / Update Messages	Robot or central route solutions / Update Messages	Robot or central route solutions / Update Messages	Update Message Time	Would robot reach intersection in time since last update	Flag set when robot calculates new route
	Comm. simulation of message queue	Comm. Simulation of message queue	Comm. simulation of message queue	Time when message queue is read	Distance to intersection at update time < VΔt	Comm. simulation of message queue
Mixed Mode Autonomy Control with Model Updates – Robot Model	Robot location sensors / Update Messages	Robot route solutions / Update Messages	Robot route solutions / Update Messages	Update Message Time	Would robot reach intersection in time since last update *Set flag if old*	Flag set when robot calculates new route
	Comm. simulation of message queue	Comm. Simulation of message queue	Comm. simulation of message queue	Time when message queue is read	Distance to intersection at update time < VΔt	Comm. simulation of message queue
Mixed Mode Autonomy Control with Model Updates – Central Control	Robot location sensors / Update	Robot or central route solutions / Update	Robot or central route solutions / Update	Update Message Time	Would robot reach intersection in time since	Flag set when robot calculates new route

Model	Messages	Messages	Messages		last update *Set flag if old*	
	Comm. simulation of message queue	Comm. Simulation of message queue	Comm. simulation of message queue	Time when message queue is read	Distance to intersection at update time < VΔt	Comm. simulation of message queue

Route Planning

Route planning is the selection of a path to navigate the robot from its current position to its final destination in the environment. The goal of each route planning is to maximize the effective usage rate of the MRS. In other words, the robots should spend as much of their time as possible working to finish a task (travel to a destination) without doing work that does not help finish the task. Each of the route planners are based on Dijkstra's Shortest Path Algorithm[**52**]. A standard implementation of Dijkstra from David Gleich was used as the basis of the algorithm used for the simulations [**53**]. This implementation of Dijkstra was used to solve for the shortest path(s) between all nodes in the environment prior to the start of the simulations. The robots using distributed control without communication used this solution to directly select a path to their destination. The other controllers make adjustments to the Dijkstra's solution to account for potential congestion when selecting a path. The routing algorithm is the same for all of the controllers with communication – the difference is in the way the controllers get the information for the world models. The algorithm starts with the Dijkstra's solution for the environment. The controller identifies all of the possible paths the robot could take h steps from its next intersection (the intersection the robot is traveling to). The base cost of these paths is the Dikstra's shortest distance from the end of the path. The path cost is the base cost plus h plus the penalties for potential delays due to other robots on the path. The penalty is the delay time (or equivalent distance traveled) for the pauses that the robot might have to make to maintain the separation distance weighted by the likelihood that they delay will occur. The router thus favors paths that have fewer robots on them and remain short. For the controllers in this work the two-step paths were considered. The potential congestion was found by identifying all the robots that could be on the two-step path during the time the route would be executed using the world model (the robot's model for distributed control or mixed mode autonomy, the central model for centralized control). If the two-step path is on the shortest path for a robot that may cause a pause, the penalty is given full weight. If the two-step path is not on the shortest path for a robot they may cause a pause, the penalty is given half weight. The weight is not zero because the robot could still end up on that path as part of its own congestion avoidance.

Route Execution and Collision Avoidance

Route execution and collision avoidance is the part of the controller that carries out the planned route while keeping the robots a safe distance away from other robots. This part of the controller functions in the same way for all of the multi-robot controllers. The robots each calculate their own heading and velocity based on the next intersection on their path. The simulator handles this as a Δx and Δy since the robots are constrained to the roads which are parallel to the x or y axes. If the collision avoidance sensor returns a value less than the separation distance (the robot ahead on the road segment is too close or the sensor returned a false positive), Δx and Δy are set to zero to pause the robot to allow the other robot to get far enough away so the robots can

maintain a safe separation distance. The Δx and Δy values are the changes in location in the motion update loop.

Communication Management

Communication management is the selection of which messages to send over the wireless network and how frequently to send them. The messages are used to update the world models that are used to do route planning. The various controllers use three different kinds of messages – robot status updates, central controller commands, and world model updates. Table 4.3 summarizes the messages and update rates for the different controllers.

Table 4-3. Controller Messages and Update Rates

Controller	**Robot/Centralized Controller**	**Message**	**Update Rate (per nominal segment time per robot)**
Distributed Control with No Communication	None		
Centralized Control	Robot	Status update – location, command receive time, next intersection	10
Centralized Control	Central	Command – robot, next vertex, retransmit, time stamp	1 + retransmissions
Distributed Control with Communication	Robot	Status update – location, destination, next intersection	10
Mixed Mode Autonomy Control	Robot	Status update – location, command receive time, next intersection	3+ (frequency increases based on number of neighboring robots, distance to central controller)
Mixed Mode Autonomy Control	Central	Command – robot, next vertex, retransmit, time stamp	As needed
Mixed Mode Autonomy Control with Model Updates	Robot	Status update – location, command receive time, next intersection	3+ (frequency increases based on number of neighboring robots, distance to central controller)
		Model update – source ID, robot info ID, location, destination, next vertex	On request when info available
Mixed Mode Autonomy Control with Model Updates	Central	Command – robot, next vertex, retransmit, time stamp	As needed
		Model update – source ID, robot info ID, location, destination, next vertex	On request when info available

The messages are loaded into a simulation message queue with a transmit send order number. The transmit send order number is a uniform random number between zero and one. Command message send order numbers are between zero and 0.1. Messages are sent in ascending order. This simulates a wireless protocol arbitration scheme to select packet order. Command messages are given generally higher priority, but are not guaranteed to be transmitted first.

4.7 Evaluation of Controllers

Monte Carlo simulations are used to assess the performance of each of the multi-robot control architectures. Simulations are run for a variety of environments and cases within the environments to determine the efficiency of the controller and the communication. Multiple trials are run for each environment and case due to the statistical nature of some of the properties in the model (especially initial conditions and communication).

Simulation Data

For each iteration of the simulations, data was logged about the performance of the MRS and the communication use. This data was used to evaluate how well the controller performed across the range of parameters. The data logged is listed in Table 4-4.

Table 4-4. Simulation Data Logged

Data Class	**Data**	**Per robot/ Per MRS**	**Description**
Robot Usage	Distance traveled	Robot	Total motion by each robot
	Intervals paused	Robot	Times the robot did not move for any reason
	Intervals paused – no communication	Robot	Times the robot did not move because it was waiting on a command
Task	Tasks completed	Robot	Number of tasks (trips) completed by a robot
	Task list	Robot	List of the destinations the robot travel to
	Task distances	Robot	Ideal distance for the task
	Task completion times	Robot	Actual time required for the tasks to be completed for each robot
Collision Avoidance	False Positives	Robot	Number of times each robot's collision avoidance sensor detected an object inside the separation distance in error
	Near Misses	Robot	Number of times each robot caused a near collision due to failure of the sensor to detect another robot
Communication	Messages Sent	Robot	Number of messages each robot and the centralized controller transmits over the wireless network
	Messages Failed	Robot & MRS	Number of messages sent that failed to be received
	Missed Messages	MRS	Messages in the queue that never get sent

Metrics

A set of metrics was calculated from the data logged for each set of the simulations. The metrics were divided into two classes – system efficiency and communication. The system efficiency metrics are used to evaluate the effectiveness of the controllers at completing the MRS tasks. The communication metrics are used to evaluate the relative cost of the controllers. The primary metrics are described in Table 4-5. The secondary metrics are used to explain why the system performed the way it did and are summarized in Table 4-6.

Table 4-5. Performance Metrics

Metric Class	**Metric**	**Description**
System Efficiency	Average robot usage rate	Usage rate is the percent of the time the robots are working to complete the task instead of handling congestion or sensor errors
	Standard deviation of robot usage rate	Variability of the robot usage rate across the MRS
Communication	Messages sent - Robot	Total number of messages transmitted by each robot
	Messages sent – Centralized Controller	Total number of commands sent by the centralized controller
	Messages failed - Robot	Total number of messages sent by the robot that failed to reach the intended target
	Messages failed percentage - Robot	Average fraction of the robots that did not receive peer-to-peer updates (distributed and mixed mode autonomy only)
	Messages failed – Centralized Controller	Total number of commands sent by the centralized controller that failed to reach the intended robot
	Total information sent	Bits of content contained in all the messages transmitted
	Total information received	Bits of content received by MRS
	Information sent standard deviation	Variability across iterations of the simulation for information sent
	Total bandwidth usage	Total bits of the communication channel used to send the messages
	Bandwidth usage standard deviation	Variability of the bandwidth used across the iterations of the simulation
	Percent bandwidth usage	Fraction of the communication channel used to send the messages
	Missed messages	Messages that failed to be sent during the control interval they were generated

Table 4-6. Secondary Metrics

Metric Class	Metric	Description
Robot inactivity rates	Robot total inactivity rate	Inverse of robot usage rate, all of the other inactivity rates sum to this number
	Inactive – sensor false positive	Fraction of the time the robot is not moving
	Inactive – pause to handle congestion	Fraction of the time the robot is waiting for a robot to get out of the separation distance
	Inactive – Effective pause due to longer route to avoid congestion	Fraction of time added to tasks by taking longer routes to avoid congested areas of the environment
	Inactive – pause due to wait for command	Fraction of the time the robot is not moving because it has not received a command (centralize control only)
Task descriptions	Average task length	Average number of road segments per task (nominally should be constant across simulations)
	Standard deviation of the task length	Variability of the length of the tasks
	Average number of tasks completed	Average number of tasks (trips to a destination) per robot
	Standard deviation of the number of task completed	Variability in the number of tasks completed by each robot (essentially how equally was the work load shared)

Chapter 5 – Simulation Results and Discussion

Chapter three presented the control architectures for this work. Chapter four detailed the implementation of the controller for simulation. This chapter presents the results of the Monte Carlo simulations of the multi-robot controllers and a comparison of the performance of the controllers. The presentation of results is focused on the three scenarios that best represent the three example applications.

5.1 Simulation Parameters and Analysis

The simulations were run for a number of combinations of parameters that represent a wide range of potential applications. These simulations were used to evaluate the relative performance of the coordination approaches as the number of robots in the MRS got larger and to identify the sensitivity of the MRS performance to the parameters.

The simulation parameters are used to approximate the many applications where dynamic multi-robot autonomous routing could be used. The parameters were environment bottleneck width, environment connection pattern, communication range, communication overhead, and the task coupling. The simulation were run for the traditional control approaches – distributed control without communication, distributed control with communication, and centralized control and the two methods developed as part of this work – mixed mode autonomy and mixed mode autonomy with model updates.

Bottleneck Width

The bottleneck is a narrow section of the task environment. It limits the number of possible paths between the two task centers. The reduction in possible paths between the task centers leads to more congestion. This would be an environment that could be divided into two parts with only a small number of roads connected the two halves. In the applications, an example would be the haul road connecting a mine pit to the rest of the open pit mine. The simulation environments either have a bottleneck width of four or are open (no restriction).

Connection Pattern

The connection pattern is the how the road sections connect the nodes (potential destinations for dynamic autonomous multi-robot routing). There are three patterns – 1 X 1, 1 X 2, and 1 X 4. The higher the second number in the pattern, the more sparsely connected the nodes. More sparsely connected nodes raise the average traffic level on each road which increases the opportunities for congestion. In real applications, a 1 X 1 pattern would be like urban driving (lots of cross streets/parallel routes to reach the same destination) and a 1 X 4 pattern would be like rural or suburban driving (long sections of uninterrupted roads, many destinations with a single way in or out).

Communication Range

The communication range is the distance the wireless signal can be reliably transmitted. All messages are sent directly (no repeaters) so this is the range that a robot can share its information. In a short range system, information is most effectively shared with immediate neighbors. A long range system allows most of the robots to receive any information transmitted. Communication range is defined as the distance from the transmitter at which the Packet Error Rate (PER) increases to 10% (90% packets successfully received). The ranges for simulation are a radius of 15 (short range) or 45 (long range) road segment lengths.

Communication Overhead

The communication overhead is the bits added to a packet other than the data required for control of the MRS. This can be two kinds of information – the communication protocol information (origin address, destinations address, packet length, error checking, etc.) or information that is not part of the coordination, but may be useful in the MRS (robot fuel levels, maintenance status, etc.). A low value allows more coordination information to be shared with the available bandwidth. The overhead values are 48 or 240 bits per message.

Task Coupling

Task coupling is the standard deviations of the distribution of the dynamic multi-robot routing destinations about the two task centers in the environment. Task coupling describes how close the destinations are to each other. The closer the destinations the more likely there will be congestion near the task centers. A value of infinity represents a uniform distribution – all of the intersections have an equal probability of being a destination for a robot. The lower the number the more closely grouped the destinations will be. The task coupling pairs (one value for each of the two task centers) used for simulation were ∞-∞, ∞-5, ∞-2.5, 5-5, 5-2.5, and 2.5-2.5. The units for task coupling are the radius in the number of road segments.

Controllers

The first three controllers simulated represent the most common approaches to multi-robot coordination. Distributed control without communication is a stigmergic controller. It was used as the baseline for performance of all of the controllers during the development and implementation. All of the controllers should perform better with the additional information provided by communication. If they did not it was a sign that the controller needed to be reworked. Centralized control treats the robots like distributed sensors and actuators – all of the decision making for coordination is handled by a single controller in the system. As long as the communication is reliable and there is sufficient bandwidth the centralized control performance should be high. Distributed control with communication allows each robot to make its own decisions about coordination with the rest of the MRS. It is more robust to communication failure, but can be less predictable.

The mixed mode autonomy controllers were developed to leverage the strengths of centralized and distributed control with communication and to scale well as the number of robots in the

MRS grew large. Mixed mode autonomy is a distributed controller with centralized controller oversight of the MRS. Each robot makes coordination decisions based on information shared with its peers. This means that the system does not fail with a communication failure (performance may be lowered however). A centralized controller monitors the coordination in the MRS and can override the individual robot decisions if a better solution exists. This provides the predictability and higher performance of centralized control. The improved scaling comes from the sharing information at a variable rate to reduce the bandwidth requirements. Mixed mode autonomy with model updates adds an additional kind of message to the system – a world model update for other robots in the MRS. These messages are sent to improve quality of the information the robots or centralized controller have in their world models when they are not getting the information directly. This is especially valuable is situations where robots have an obstructed line-of-sight or are far from the central controller. For the simulations in this work, there were no obstructions so a robot's communication failures were mostly due to distance. This meant that most of the additional messages with mixed mode autonomy with model updates were sent to the centralized controller to update its model of robots far its access point.

Analysis Methods

The two classes of metrics used to assess the MRS performance were system efficiency and communication. The metrics were explained in detail in Chapter 4. The controller goals were to achieve a high effective robot usage rate and scale well as the number of robots in the MRS got large. Effective robot usage rate is the percent of the time the robots in the system are contributing to task completion and is a measure of the system efficiency. The effective robot usage rate can be compared directly between each of the controllers for a set of simulation parameters. A controller is scales well if the effective robot usage rate decreases slowly as the number of robots increase and the bandwidth usage remains under the available limit.

Two levels of comparison of data sets were used. When data is observed informally to be different, the difference is suggested by the data. When the difference is statistically significant the difference is shown. Twenty iterations (or more) were used for each of the standard parameter combinations. For most of the data sets this was enough to clearly show the differences. For some of the non-standard parameters and control implementations that were run with fewer iterations, the differences are suggested, but there are not enough data points to show the statistical significance of the difference. These non-standard sets are discussed as part of the future work section in Chapter 6.

The mixed mode autonomy with model updates results are not displayed on the plots for the majority of the simulations because the differences are so small compared to the standard mixed mode autonomy control. The simulation conditions did not sufficiently stress communication to see the benefit of the model updates. Section 5.7 in this chapter discusses the results of the two levels of mixed mode autonomy in more detail.

5.2 Motivation for Mixed Mode Autonomy

The initial simulations of the more traditional control approaches demonstrated the utility of a control architecture like mixed mode autonomy. The effective robot usage rate for centralized control drops off sharply once the bandwidth limits are reached. This is due to the latency in

transmission of messages, especially the control messages. Figure 5-1 shows the effective robot usage rate for centralized control for the high and low overhead values. The blue points are low overhead communication and the green are high overhead communication. This plot is for the non-bottlenecked environment with the 1 X 4 grid connection pattern. The MRS reaches the 1 Mbps bandwidth limit around 400 robots in the high overhead case. The saturation of the wireless network clearly causes degradation in the MRS performance. At 500 robots, this difference in effective robot usage rate does not seem large; however, a 1% difference amounts to an effective decrease of 5 robots. In environments with a bottleneck, the effective robot usage rates are much lower and the differences are exaggerated.

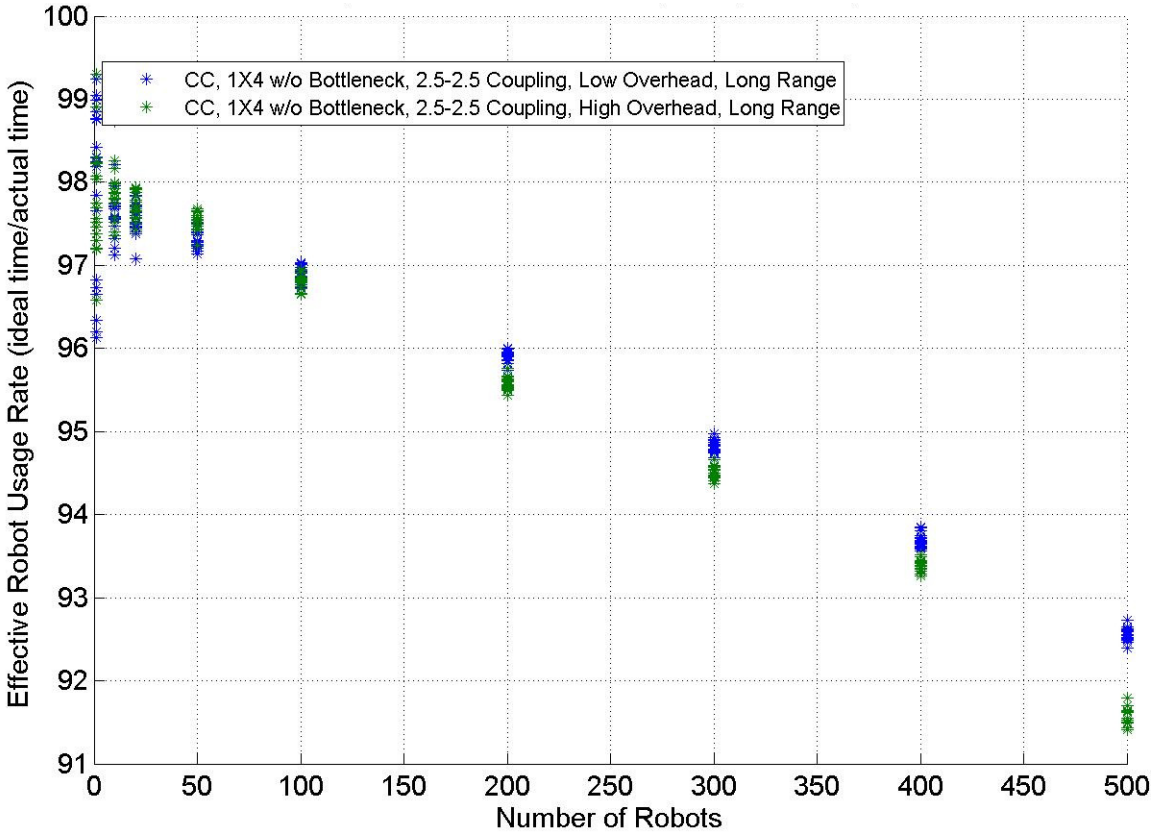


Figure 5-1. Effective Robot Usage Rate for Centralized Control with High and Low Overhead

The distributed controller with communication scales a little better with respect to the bandwidth limits, but still reaches the limit near 500 robots in the high overhead case. The performance, however, is significantly lower than the centralized controller. A comparison of the effective robot usage rates for distributed control with communication, distributed control without communication, and centralized control is show in Figure 5-2. This plot is for a bottlenecked environment with a 1 X 2 connection grid, a long range signal with low overhead communication, and coupled tasks. The red points are for centralized control, blue are for distributed with communication control, and green is for distributed control without communication (the baseline case). As the number of robots increase, the performance difference between centralized control and distributed control with communication grows. Both

are better than distributed control without communication. Based on the relative slopes of the effective robot usage rate, these trends would continue as robot group sizes got even larger.

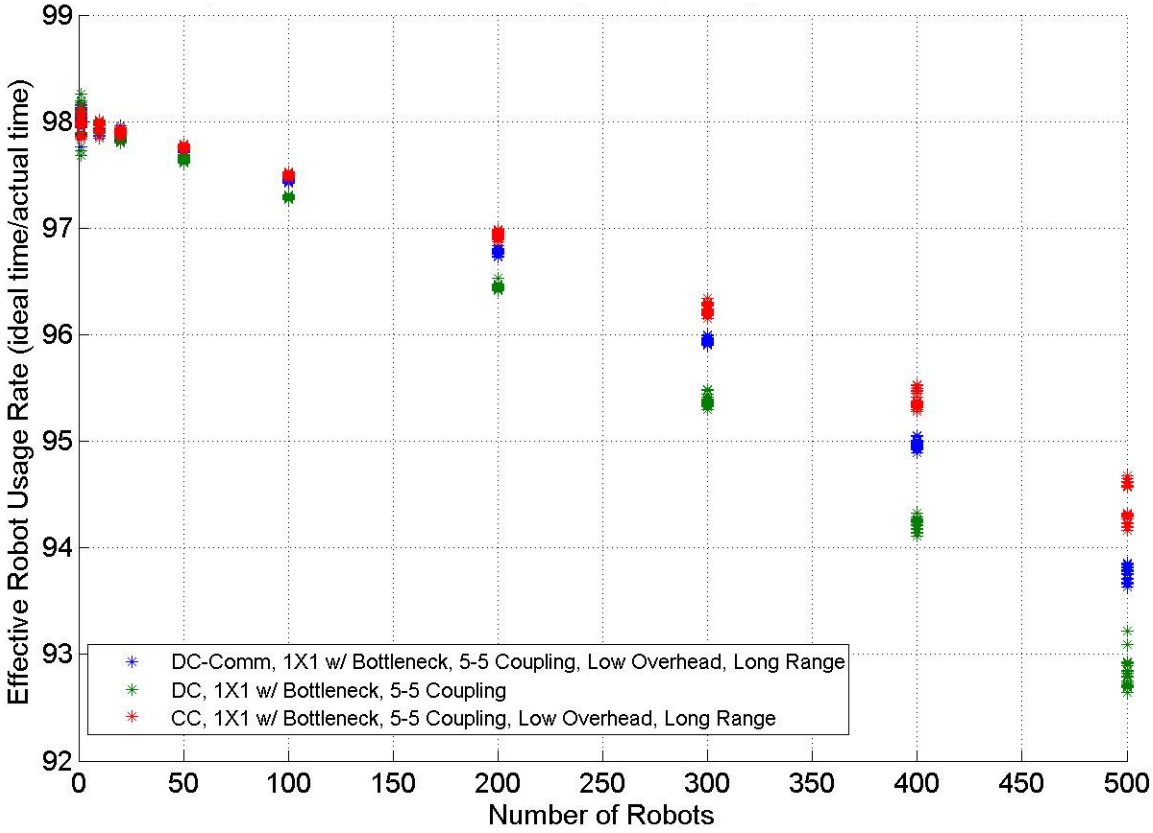


Figure 5-2. Effective Robot Usage Rate for Centralized Control and Distributed Control with Communication

To maintain or improve the performance of the MRS past the current limits in group size, significant reductions in the amount of information transmitted is required. Traditional distributed control with communication does not reduce the bandwidth requirements enough to allow the group size to increase much and results in a loss in performance. Further reductions in the bandwidth for distributed control with communication (by updating robot status less frequently) would allow more robots, but reduces the performance much more. Mixed mode autonomy was designed to perform as well as or better than other controllers with bandwidth requirements that are lower than even the distributed control with communication. The rest of this chapter presents the results for three example applications of the dynamic multi-robot autonomous routing problem.

5.3 Example Applications

Many multi-robot applications can be represented as dynamic multi-robot autonomous routing problems. The parameters presented in section 5.1 describe the application environments, tasks, and communication settings. The simulations of dynamic multi-robot routing and the control architectures were run for a wide range of the parameters, but the data is presented for three specific combinations that best represent three of the most likely applications. The applications

are automation of open pit mines, container terminals, and warehouses. Each of these applications has the potential to require many hundred (and possibly thousands) of robots or autonomous systems per site. Table 5-1 lists the parameter settings for each of the applications. The applications are described in more detail and the simulation results are presented in the following sections.

Table 5-1. Simulation Parameter Settings for the Example Applications

	Open Pit Mine	Container Terminal	Warehouse
Bottlenecked or Open	Bottlenecked	Bottlenecked	Open
Grid connection pattern	1 X 1	1 X 2	1 X 4
Task coupling	2.5 – 2.5	5 – 2.5	∞ - 2.5
Communication range	Long	Long	Short
Communication overhead	High	Low	Low

5.4 Open Pit Mine

Application

Open pit mines are commonly used to extract materials such as coal, diamonds, granite, marble, or metal ores (copper, iron, nickel, gold, etc.). The concentration of the metals can range from a few tenths of a percent to 1 or 2 ppm for gold. A large hole is dug in the ground to remove the overburden (material covering the ore). This overburden is moved to another spot on the mine site for storage. The ore is then removed by truck and transported over the haul road(s) to an area to be crushed and loaded for transportation to another site to process the ore to separate the wanted material from the rest. Each site can include many different kinds of machines such as crushers, loaders, haul trucks, road maintenance vehicles, etc. The dynamic multi-robot autonomous routing problem is for specifically coordinating the haul trucks as they travel between the pit and the crushers. An example open pit mine is show in Figure 5-3. This particular mine site is to extract diamonds. Figure 5-4 shows the functional layout of an open pit mine. An example task for the dynamic multi-robot autonomous routing problem in the mine site is highlighted in green.

For simulation, the dynamic multi-robot autonomous routing problem can be described as a loosely constrained (1 X 1 connection pattern) environment with a bottleneck, with tightly coupled tasks, a long range communication system, and a high communication overhead. The bottleneck in the environment is the haul road. This is one connection between the pit and the crusher/loading area. The 1 X 1 grid connection pattern is used since the haul trucks can move fairly freely through the mine pit and crushing/loading areas. For safety reasons the communication system must cover the entire mine site so the long range communication is used. The messages would include information other than just the data needed for routing, such as fuel levels, engine status, etc. so the overhead rate is high. The destinations (tasks) are tightly grouped about the areas where material is loaded and unloaded so the tightest coupling is used for both tasks (2.5-2.5).

Figure 5-3. Example Open Pit Mine Layout [**54**] (Alexander Stapanov. Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:Udachnaya_pipe.JPG , Used under fair use guidelines, 2011.)

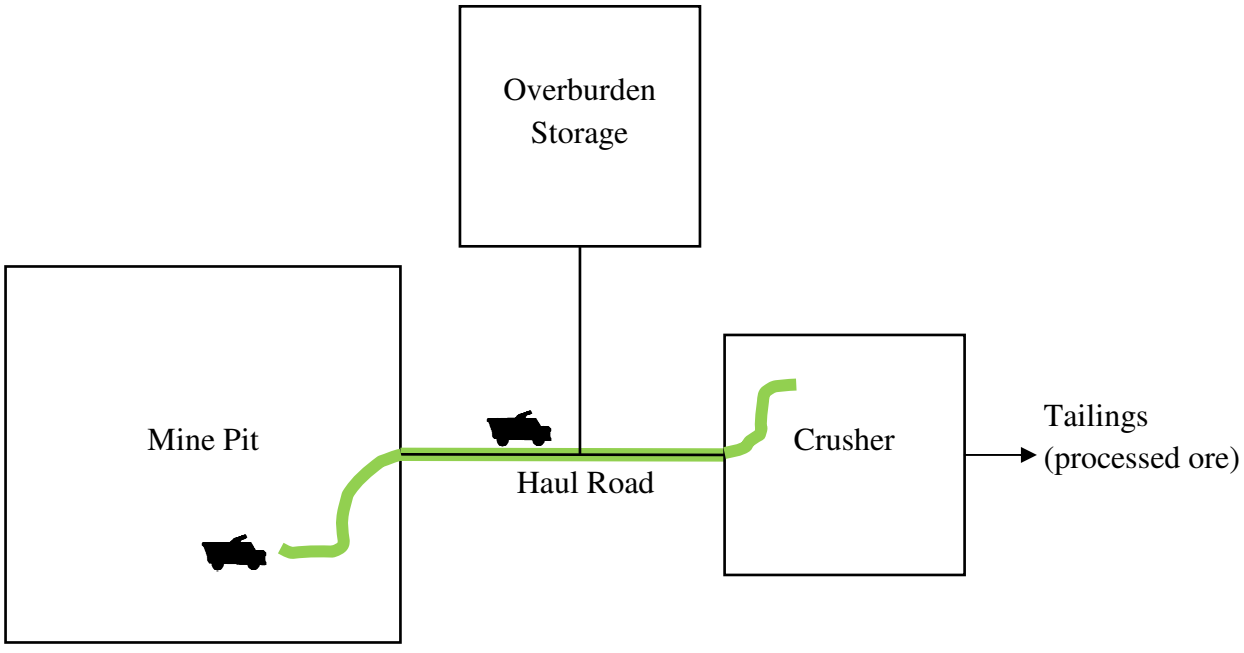


Figure 5-4. Open Pit Mine Layout

Simulation Results

The dynamic multi-robot autonomous routing problem was simulated for the open pit mine parameters at least twenty times for each of the controllers. This section presents the metrics derived from the data logged for each simulation.

The effective robot usage rate tracks how efficiently the MRS operates for the given parameters for the environment, tasks, and communication. The effective robot usage rate for each of the controllers for the open pit mine problem is shown in Figure 5-5. Table 5-2 lists the rate of decrease of the effective robot usage rate for each of the controllers. The rate of decrease is calculated by fitting the data for higher number of robots to a line. For the open pit mine problem, the effective robot usage rate is clearly higher for mixed mode autonomy than the other controllers for all numbers of robots over 100. The rate of decrease is also lower for mixed mode autonomy. This lower rate of decrease means that the difference in performance will continue to grow larger as the number of robots in the MRS increases. Centralized control performs poorly in these conditions – with some numbers of robots it is equivalent to distributed control without communication, the baseline for performance. The high communication overhead and the latency of messages it causes is the parameter most responsible for the low effective usage rate since the robots will wait for command messages. Distributed control with communication is generally better than centralized control for the open pit mine, but does not perform as well as mixed mode autonomy.

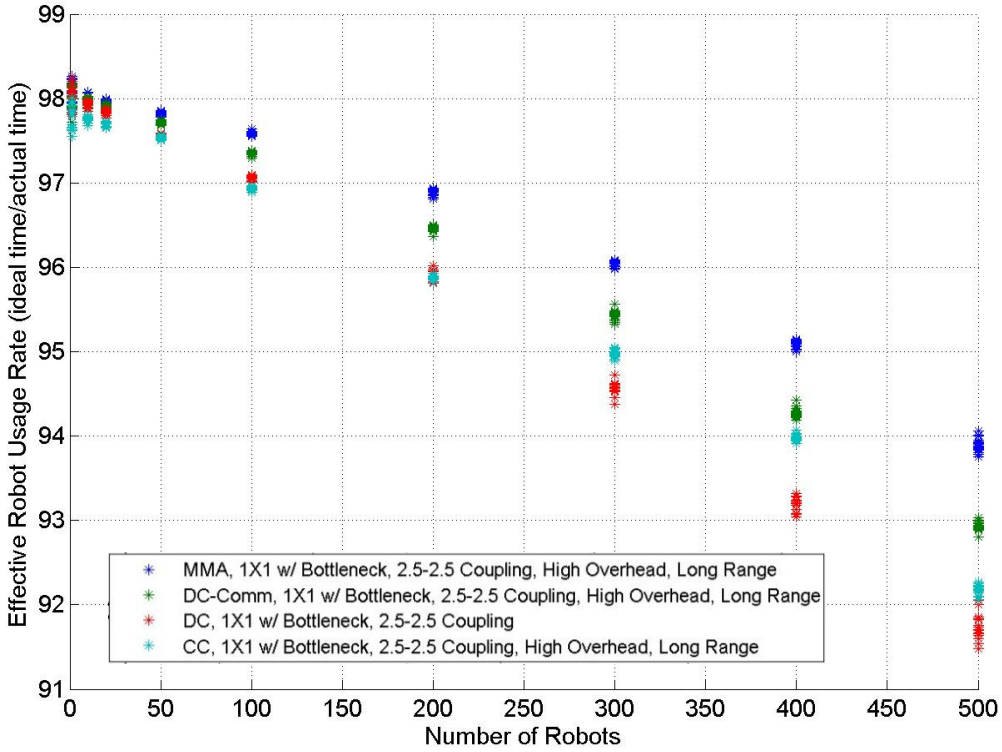


Figure 5-5. Effective Robot Usage Rates for Each Controller for the Open Pit Mine Problem

Table 5-2. Rate of Decrease of Effective Robot Usage Rate for the Open Pit Mine Problem

Controller	**Rate of Decrease of Effective Robot Usage Rate (% / additional robot)**
Distributed Control without Communication	0.0146
Distributed Control with Communication	0.0135
Centralized Control	0.0179
Mixed Mode Autonomy	0.0119

The standard deviation of the effective robot usage rate is a measure of how evenly the load is spread across the robots in the MRS. A low value means that all of the robots are completing their tasks with similar levels of efficiency. A high value means some robots are much more efficient. If the variability is high, it will make performance more unpredictable since some robots will contribute much less to completing the set of tasks for the MRS. Figure 5-6 shows the standard deviation of the effective robot usage rate for each controller for the open pit mine problem. The variation increases with the number of robots due to the increased complexity of the problem. Centralized control is the most consistent, as is expected since every routing decision is made with fairly complete knowledge of the system. Mixed mode autonomy is a little more consistent than distributed control with communication with a high number of robots. This demonstrates part of the value of central oversight. Distributed control without communication is the most variable due to the fact the robots do not share any information and control is only handled by sensing.

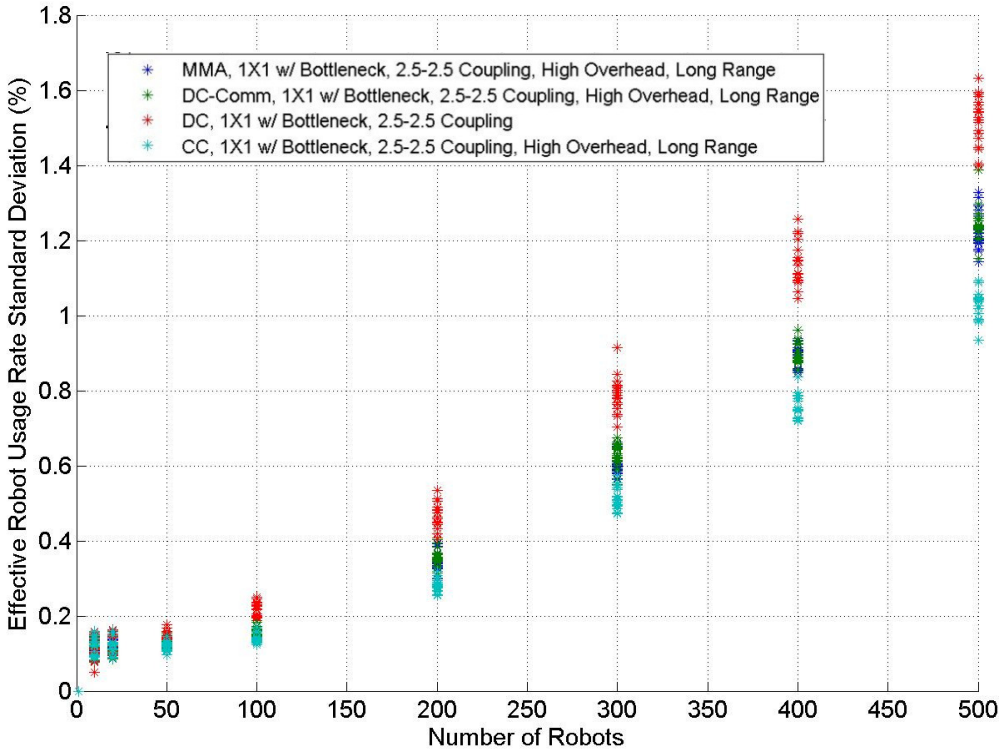


Figure 5-6. Standard Deviation of the Effective Robot Usage Rates for Each Controller for the Open Pit Mine Problem

The other significant aspect of performance and scalability is the bandwidth usage (communication costs). The available bandwidth for all of the simulations was 1 Mbps. Figure 5-7 shows the average bandwidth usage for each of the controllers for the automated open pit mine. Even with the highest effective robot usage rate, mixed mode autonomy has the lowest bandwidth usage. Mixed mode autonomy could scale to over 900 robots before the bandwidth limits would be reached. Centralized control and distributed control with communication, however, are already reaching their limit at or before 500 robots. Centralized control saturates the wireless communication system at 400 robots for the open pit mine. This means that not all of the messages that the controllers intend to send can be transmitted. Distributed control with communication reaches the 1 Mbps limit a little before 500 robots. This, however, has less impact on performance than with centralized control. The robots are required to receive command messages before they enter an intersection which means that unsent commands lead to idled robots. Unsent status updates slowly degrade the quality of the information used for routing. This means that the performance drop will be more gradual for distributed control with communication or mixed mode autonomy that are not dependent on the centralized controller.

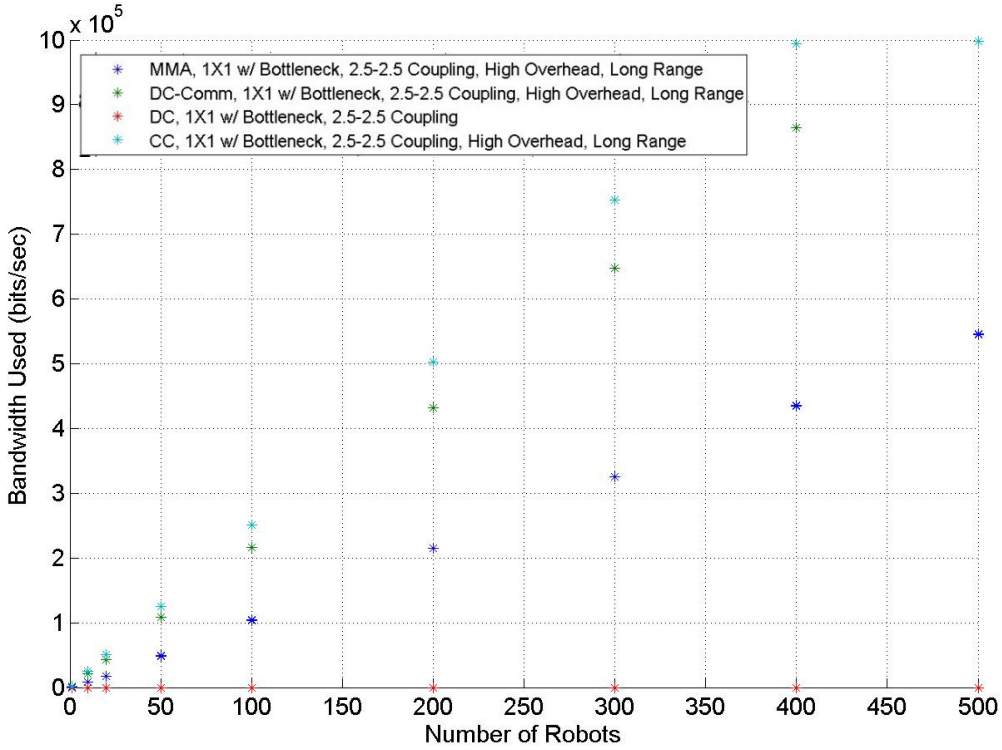


Figure 5-7. Bandwidth Usage for Each of the Controllers for the Open Pit Mine

Reaching the bandwidth limit at over any communication interval causes messages to not be sent during that interval. The command messages that are missed are still sent, but at a later time. This is the source of latency in any of the controllers. Robot status updates that are not sent are replaced with newer update messages since it does not make sense to send information that is not current. Figure 5-8 shows the missed messages for each controller. Each missed message is an instance of a command sent late or a status update that was not sent. None of the controllers experience missed messages before 400 robots.

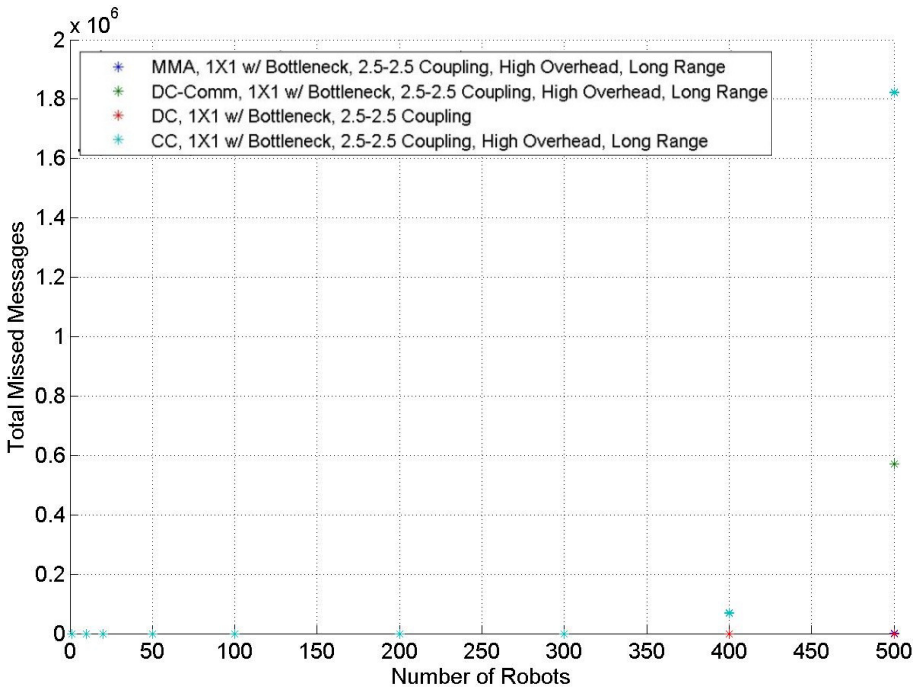


Figure 5-8. Missed Messages for Each of the Controllers for the Open Pit Mine

The bandwidth is used by the robots to send status update messages and the centralized controller to send command messages. Figure 5-9 shows the number of messages sent by the robots for each of the controllers for the automated open pit mine. Figure 5-10 shows the number of messages sent by the centralized controller for each of the controllers. The number of update messages sent per robot with centralized control and distributed control with communication are constant until the bandwidth limitation is reached. Mixed mode autonomy gradually increases to a fairly steady rate starting around 100 robots. The update message rate is a function of the number of robots in the vicinity of the robot and how far the robot is from the central controller. Distributed control with communication does not have any command messages from the central controller. The command messages are a small part of the total bandwidth use for mixed mode autonomy. The total number of command messages set by centralized control does not drop as the number of robots increase since all of the messages must be sent. The number of messages actually increases a little at 500 robots since commands start to make up a larger percentage of the total messages which leads to a higher number of retransmitted commands due to communication failures.

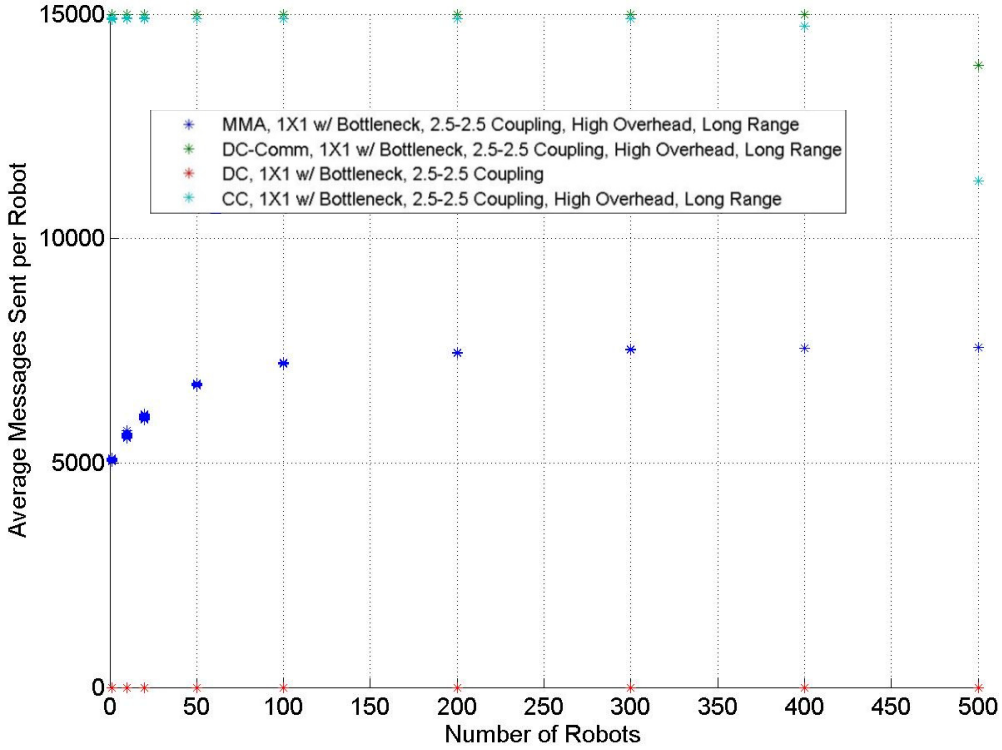


Figure 5-9. Messages Sent per Robot for Each of the Controllers for the Open Pit Mine

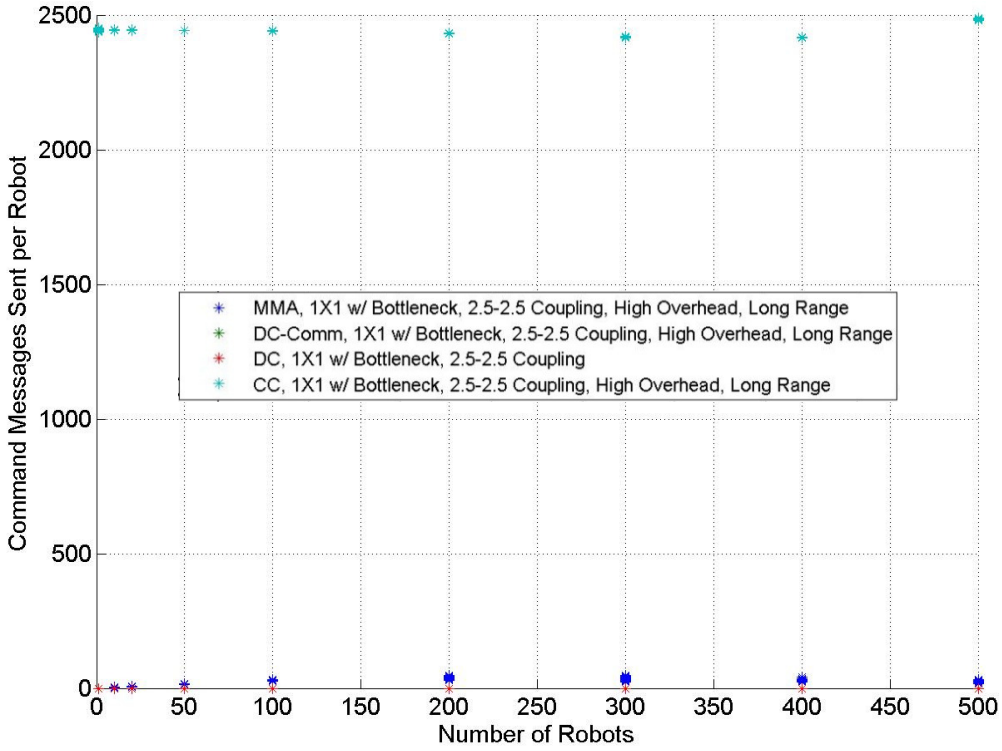


Figure 5-10. Messages Sent by the Centralized Controller for Each of the Controllers for the Open Pit Mine

The remainder of this section examines the sensitivity of the performance to the parameters selected to represent the open pit mine problem. This is to show how likely the appropriate controller for the open pit mine problem might change as a parameter changes. The parameters examined are task coupling, communication, and environment settings.

The task coupling is how tightly grouped the tasks are to one point in the environment. The assumption for the open pit mine was that the tasks were tightly coupled at both the mine pit and the crusher/loading area (2.5-2.5 task coupling). Different methods of mine operation could impact this assumption. For example a site could include many sites where trucks could pick up material, which would lead to a more uniform task coupling on the pit side. Figure 5-11a-c shows the sensitivity of the MRS performance to the task coupling for the different controllers.

All of the controllers show that the more tightly coupled the tasks the lower the effective robot usage rate. The case used for the open pit mine is the most tightly coupled (yellow plot, lowest curve). In all cases mixed mode autonomy control maintains a higher effective robot usage rate than the other controllers. The rate of decline as the number of robots increase is still lower for all task coupling values for mixed mode autonomy as well. The differences in effective robot usage rate are fairly consistent across each of the values for task coupling.

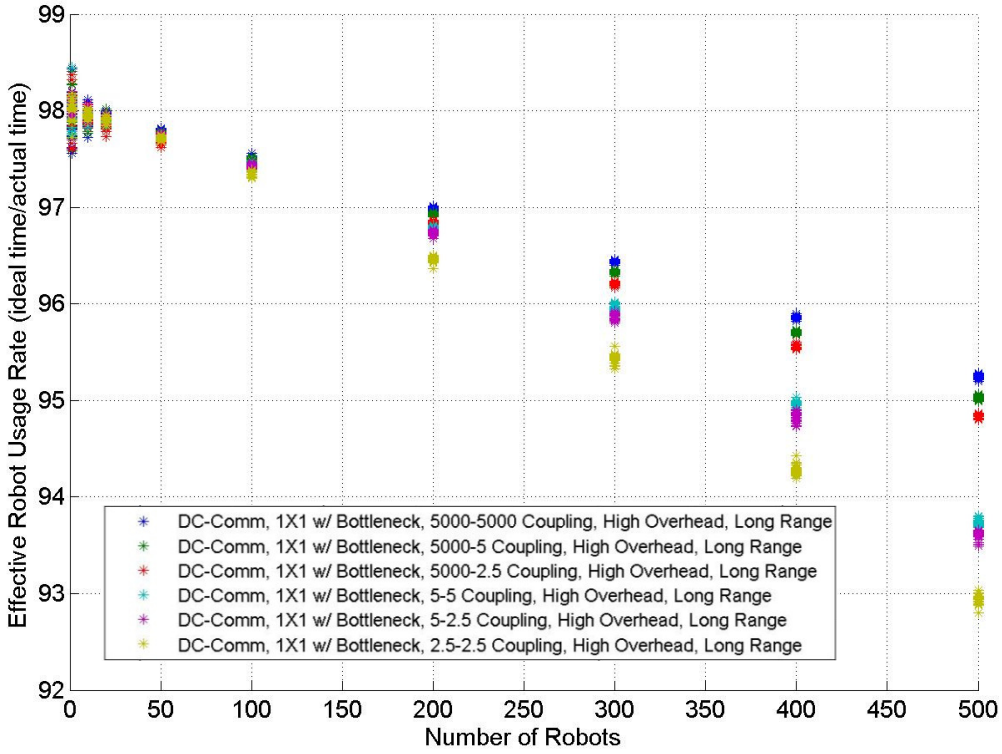


Figure 5-11a. Effective Robot Usage Rates for Distributed Control with Communication for the Open Pit Mine Problem with Varying Task Coupling

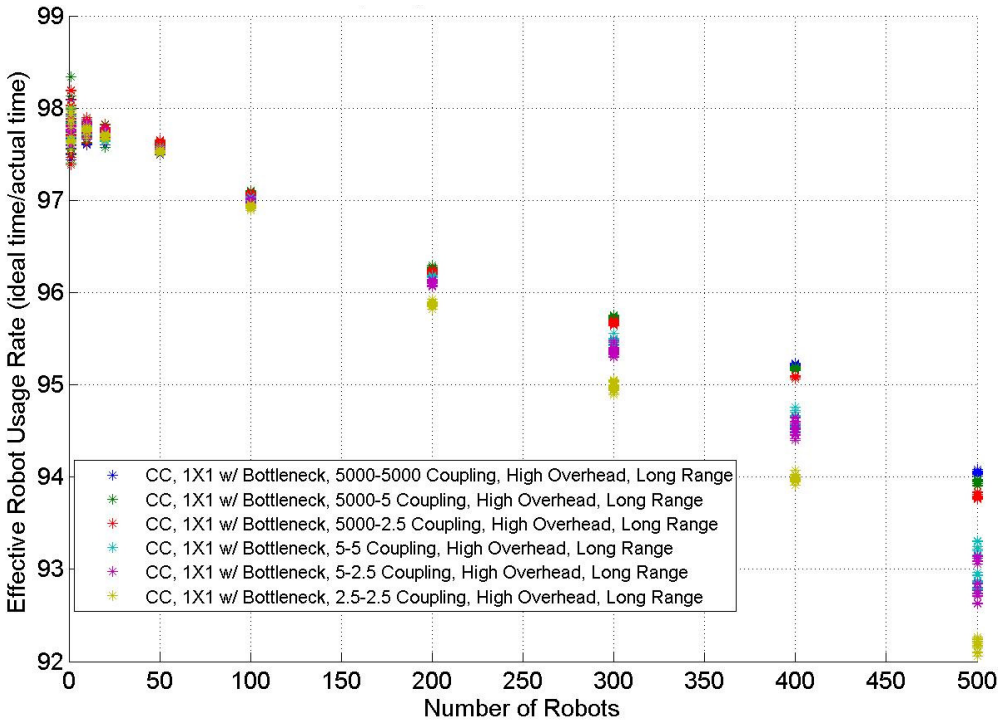


Figure 5-11b. Effective Robot Usage Rates for Centralized Control for the Open Pit Mine Problem with Varying Task Coupling

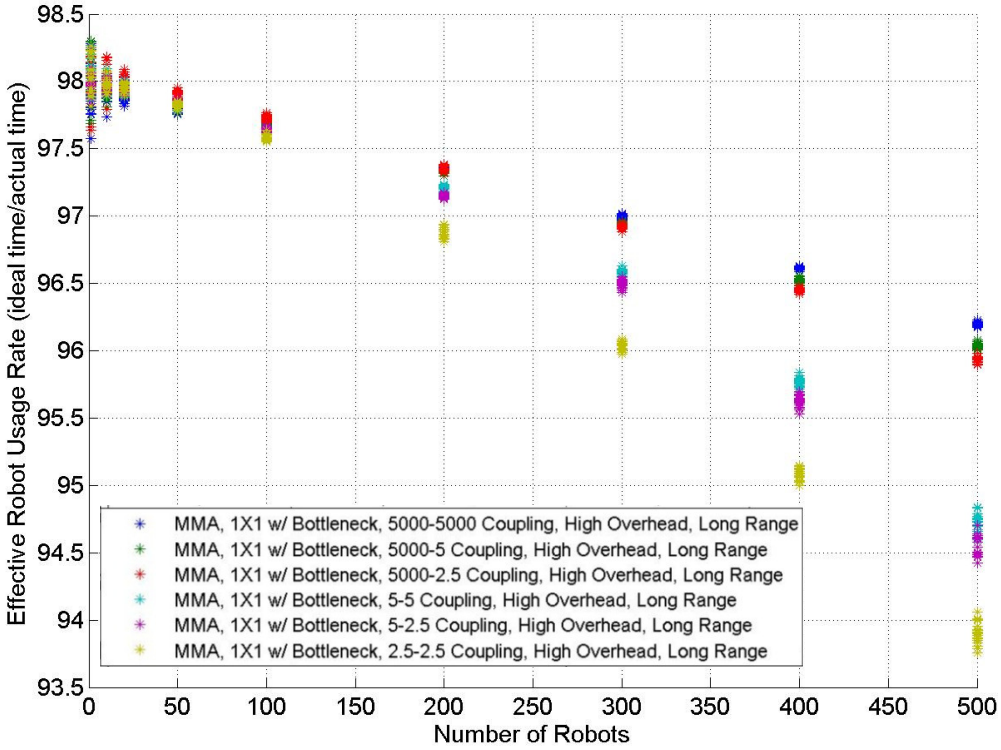


Figure 5-11c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Open Pit Mine Problem with Varying Task Coupling

Communication settings are a function of the wireless protocol used for the communication system and some assumptions about the communication requirements other than to solve the dynamic multi-robot autonomous routing problem. For the open pit mine, the assumption is that there will be high overhead to account for updates unrelated to routing. These updates might include things like fuel level, engine health, road surface conditions, etc. The other assumption is that a wireless communication should cover the entire environment (long range communication). Figure 5-12a-c shows the sensitivity of the MRS performance to the communication settings (range and overhead) for the different controllers.

The communication settings do not have much effect on the effective robot usage rate for decentralized control with communication or mixed mode autonomy for the open pit mine problem. Centralized control was only used with long range communication – when robots traveled to tasks far from the central controller the high packet error rate led to long delays before commands or updates could be received. This led to effective robot usage rates that dropped below 50% which is well below what would be feasible in most any application. The increased overhead drops the effective robot usage rate with centralized control at all numbers of robots. The difference grows with the number of robots. Centralized control with low overhead performs almost as well as mixed mode autonomy. Reducing the latency which reduces the idle time for the robots by allowing the command messages to be sent during the interval they are generated and proving more time to retransmit failed command messages.

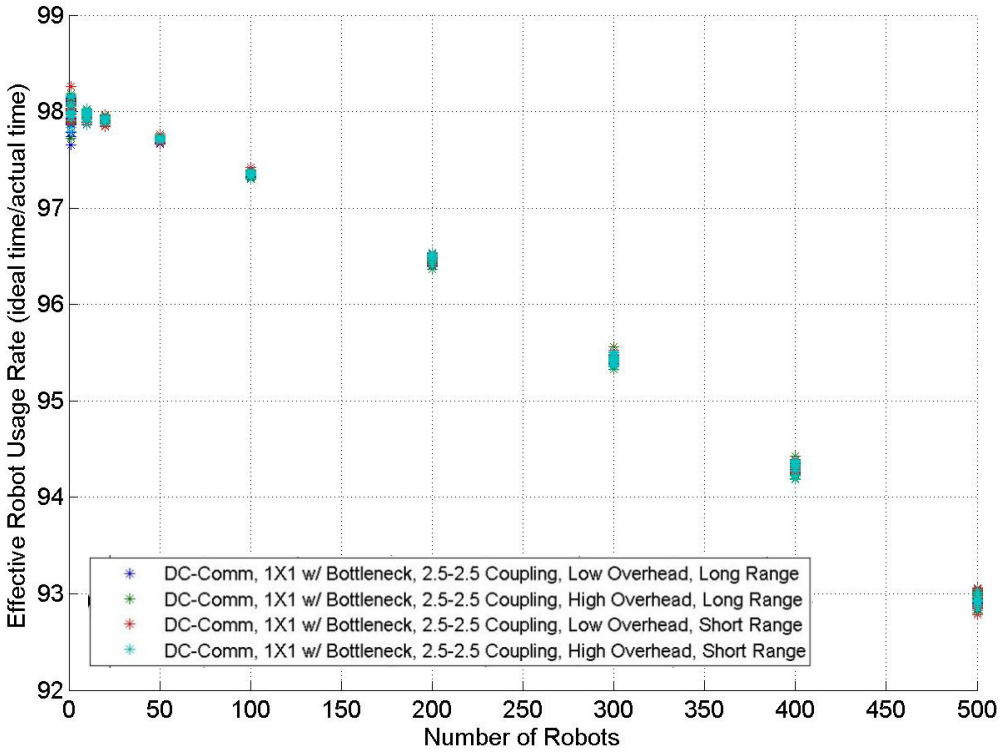


Figure 5-12a. Effective Robot Usage Rates for Distributed Control with Communication for the Open Pit Mine Problem with Varying Communication Settings

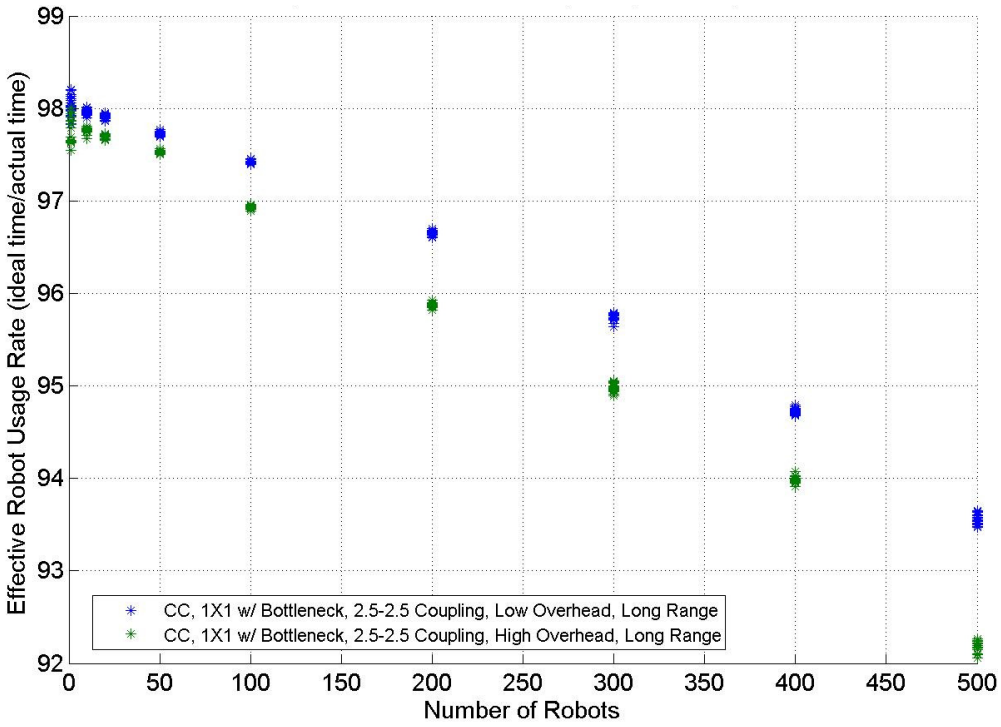


Figure 5-12b. Effective Robot Usage Rates for Centralized Control for the Open Pit Mine Problem with Varying Communication Settings

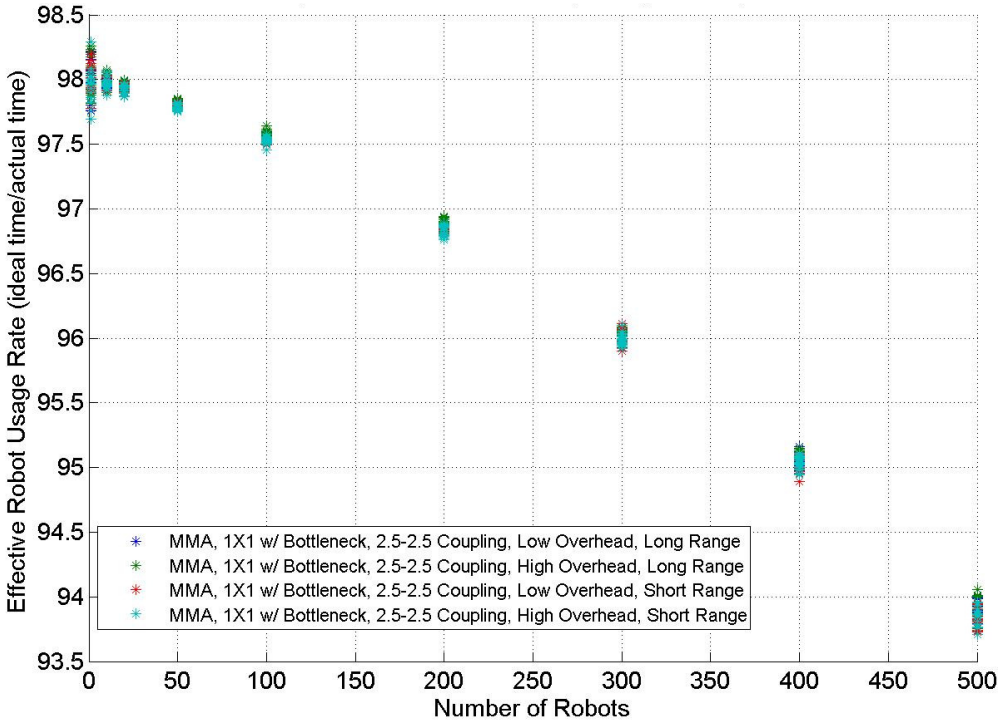


Figure 5-12c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Open Pit Mine Problem with Varying Communication Settings

The environment parameters are shaped by the constraints on the robot travel in the dynamic multi-robot autonomous routing problem. The two constraints are the scarcity of the road connections (grid connection pattern) and the presence or absence of a restriction between the two task centers (bottleneck). The open pit mine problem assumes that the robots can travel freely within the pit and crusher/loading areas (1 X 1 connection pattern) and that the haul road is a bottleneck. Figure 5-13 shows the sensitivity of the MRS performance to the environment parameters (bottleneck width and grid connection pattern) for the different controllers.

For all the controllers, the presence or absence of the bottleneck has the most impact to reduce the effective robot usage rate for the open pit mine problem. Reducing the connectivity of the environments also reduces the effective robot usage rates. The base case (1 X 1 with a bottleneck) is in blue and is nearly the same as the 1 X 4 without a bottleneck case (yellow). Mixed mode autonomy has the highest effective robot usage rate for all cases except for the 1 X 4 connection pattern with a bottleneck (distributed control with communication is better) which was by far the toughest environment for dynamic multi-robot autonomous routing. The rate of decline of the effective robot usage rate is lower for all but that one case. This was a function of penalties assigned in the router for potential congestion on the possible paths. When the penalties were turned down a little the difference between controllers reversed for the 1 X 4 with bottleneck case. It however, only modestly improved the performance of the mixed mode autonomy for this case at a significant expense to all of the other cases.

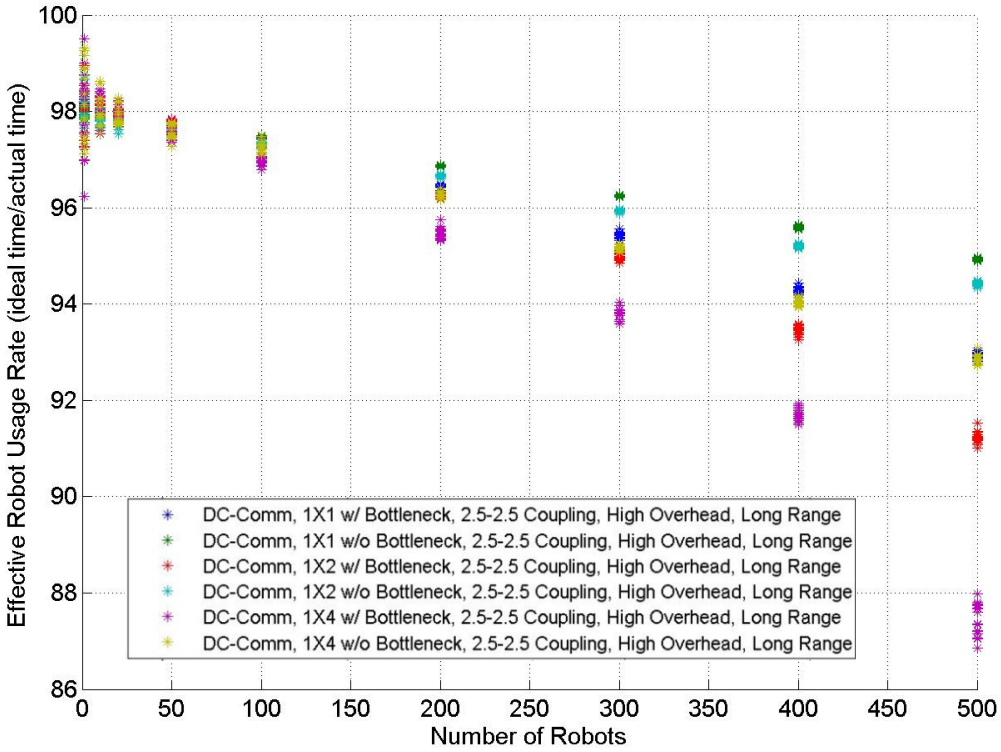


Figure 5-13a. Effective Robot Usage Rates for Distributed Control with Communication for the Open Pit Mine Problem with Varying Environmental Parameters

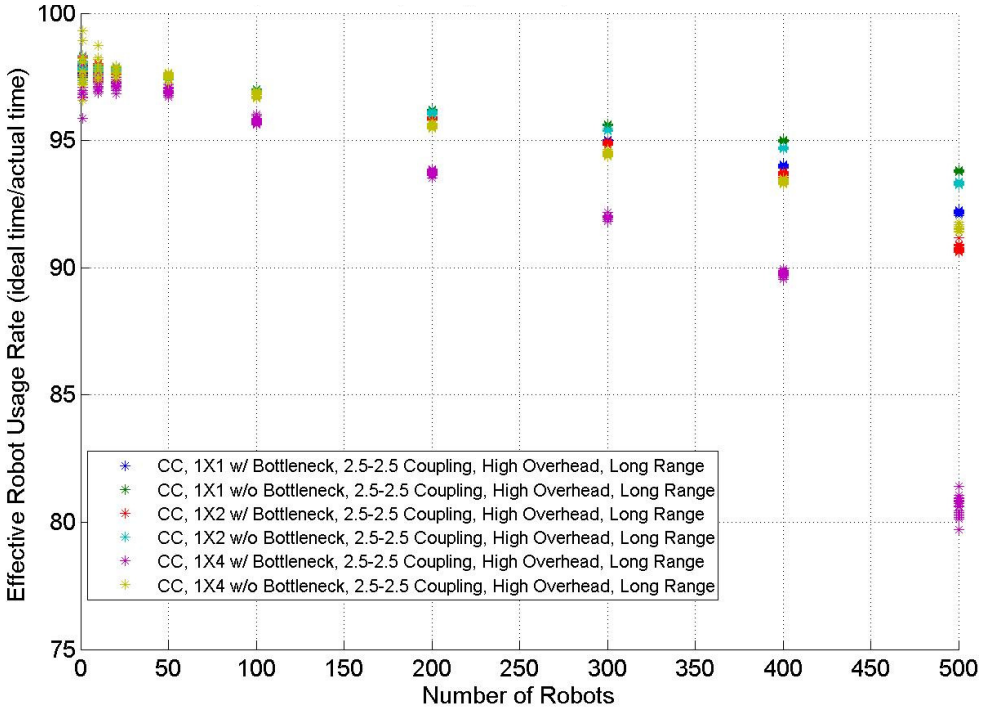


Figure 5-13b. Effective Robot Usage Rates for Centralized Control for the Open Pit Mine Problem with Varying Environmental Parameters

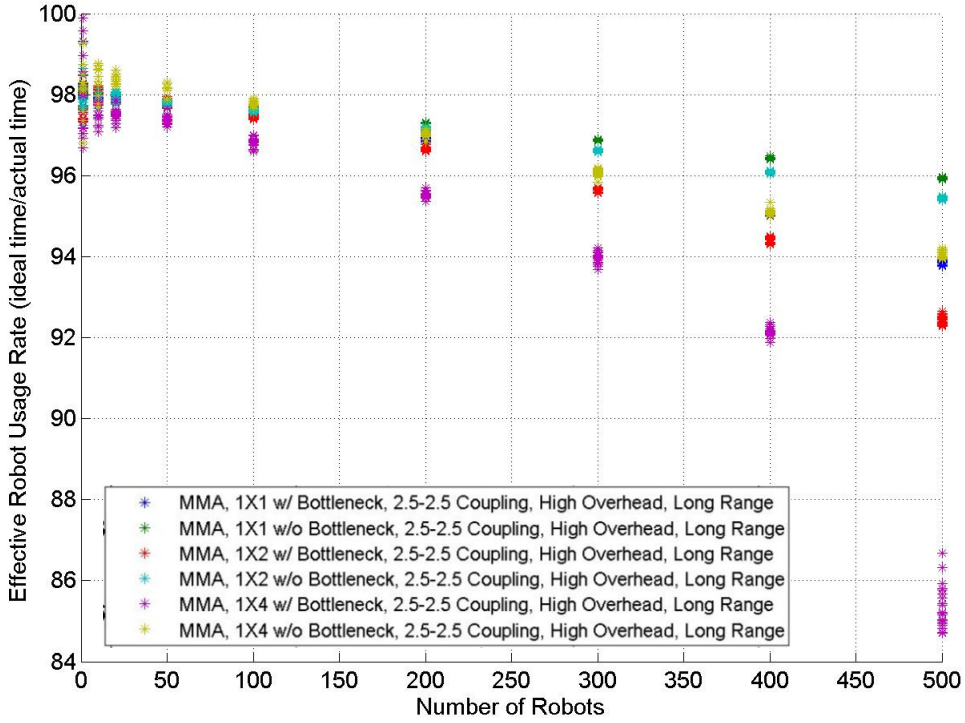


Figure 5-13c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Open Pit Mine Problem with Varying Environmental Parameters

5.5 Container Terminal

Application

Container terminals are where boats, trains, or planes are loaded and unloaded with cargo containers. A cargo container is a standard sized steel box 8' X 8' by 20 or larger used to transport goods. Most non-bulk cargo transported worldwide is moved in containers. Container terminals include a storage yard where the containers are kept between when they are unloaded and loaded. An example of a container terminal is shown in Figure 5-14. The typical functional layout of a container terminal is shown in Figure 5-15. The dynamic multi-robot autonomous routing problem is to transport a container from the loading berths to the container yard

For simulation the dynamic multi-robot autonomous routing problem can be described as a moderately constrained (1 X 2 connection pattern) environment with a bottleneck, with tightly and moderately coupled tasks, a long range communication system, and a low communication overhead. The bottleneck in the environment is the interchange area. This is one connection between the loading berths and the container yard. The 1 X 2 grid connection pattern is used since the robots can move through the container yard and loading berths with some restrictions. For safety reasons the communication system must cover the entire terminal site so the long range communication is used. The messages would be primarily focused on the dynamic multi-robot autonomous routing problem so the overhead rate is low. The destinations (tasks) are tightly grouped about the loading berths since many robots would be used to unload one ship or train and moderately grouped about the container yard since there is likely to be more variability in where the open space is to store a container. For these reasons, the container terminal uses 5-2.5 task coupling.

Figure 5-14. Example of a Container Terminal [**55**] (Missy Schmidt. Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:Norfolk_International_Terminal.jpg, Used under fair use guidelines, 2011.)

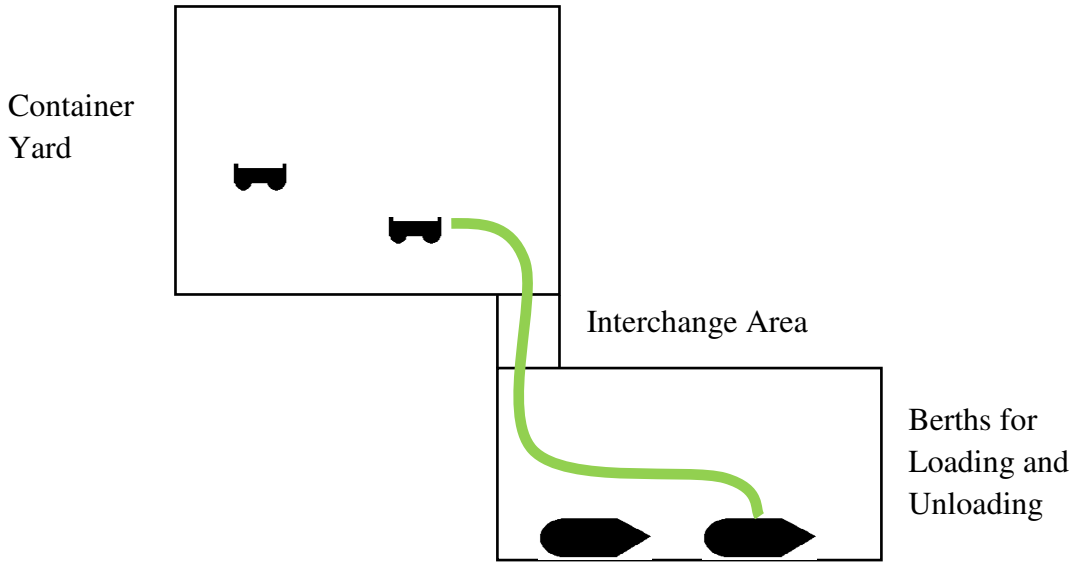


Figure 5-15. Container Terminal Layout

Simulation Results

The dynamic multi-robot autonomous routing problem was simulated for automated container terminal parameters at least twenty times for each of the controllers. This section presents the metrics derived from the data logged for each simulation.

The effective robot usage rate estimates the efficiency of the MRS for each of the controllers for the automated container terminal problem and is shown in Figure 5-16. Table 5-3 lists the rate of decrease of the effective robot usage rate for each of the controllers.

For the automated container terminal problem, the effective robot usage rate is highest for mixed mode autonomy and centralized control. For most number of robots the effective robot usage rate for mixed mode autonomy is slightly higher, but this difference is not statistically significant. Both are clearly better than distributed control with communication. The use of communication has a larger impact in this case than with the open pit mine. Distributed control without communication drops to 80% effective robot usage at 500 robots, a difference of almost 12% from the next poorest performing controller. The difference with the open pit mine was only 3.5% to the best performing controller.

The rate of decrease is lowest for centralized control and mixed mode autonomy. This lower rate of decrease means that the difference in performance will continue to grow larger as the number of robots in the MRS increases. The rate of decrease would get sharper for centralized control when the bandwidth for the communication system is saturated at around 1100 robots. Distributed control with communication has a faster rate of decrease in effective robot usage rate. Distributed control without communication already performs poorly compared to any of the other controllers and that difference will only get worse as the MRS gets larger. The simulation results suggest that distributed control without communication will reach the point of

diminishing return as soon as 600 robots (in other words more robots would not add any additional tasks completed by the MRS).

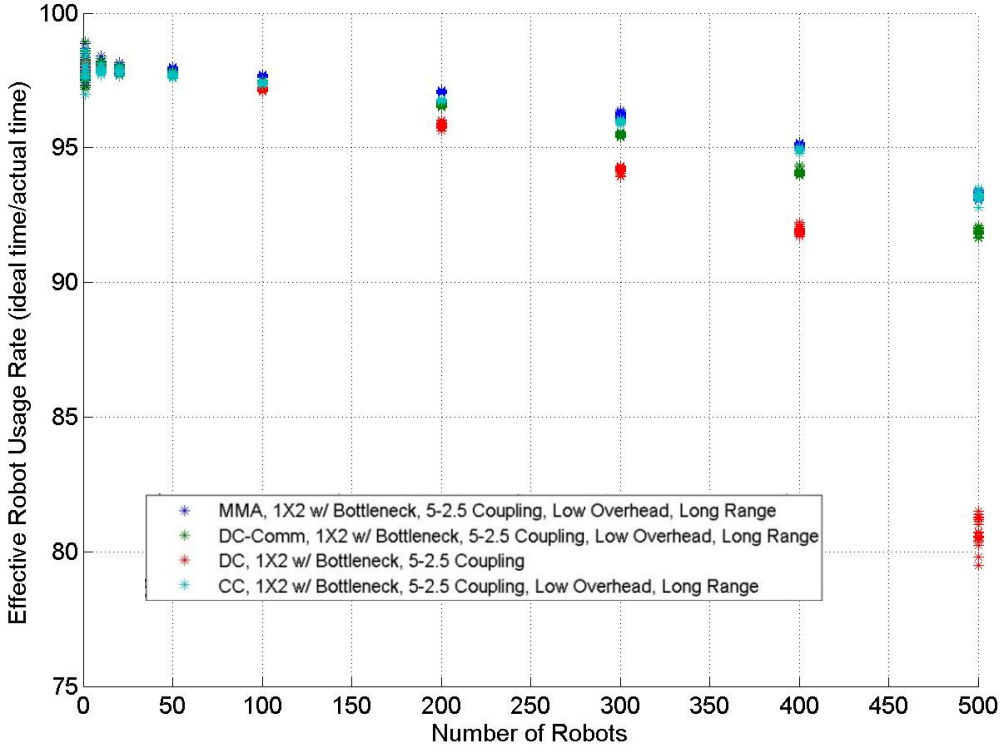


Figure 5-16. Effective Robot Usage Rates for Each Controller for the Container Terminal Problem

Table 5-3. Rate of Decrease of Effective Robot Usage Rate for the Automated Container Terminal Problem

Controller	**Rate of Decrease of Effective Robot Usage Rate (% / additional robot)**
Distributed Control without Communication	0.1116
Distributed Control with Communication	0.0216
Centralized Control	0.0173
Mixed Mode Autonomy	0.0189

The standard deviation of the effective robot usage rate is a measure of how evenly the load is spread across the robots in the MRS. Figure 5-17 shows the standard deviation of the effective robot usage rate for each controller for the automated container terminal problem.

Centralized control has the most consistent effective robot usage rate with higher number of robots when the variability starts to increase for all of the controllers. Mixed mode autonomy lags behind centralized control, but is better than distributed control with communication starting

at 300 robots. As would be expected based on the low effective robot usage rate, distributed control without communication has a much higher variability in performance.

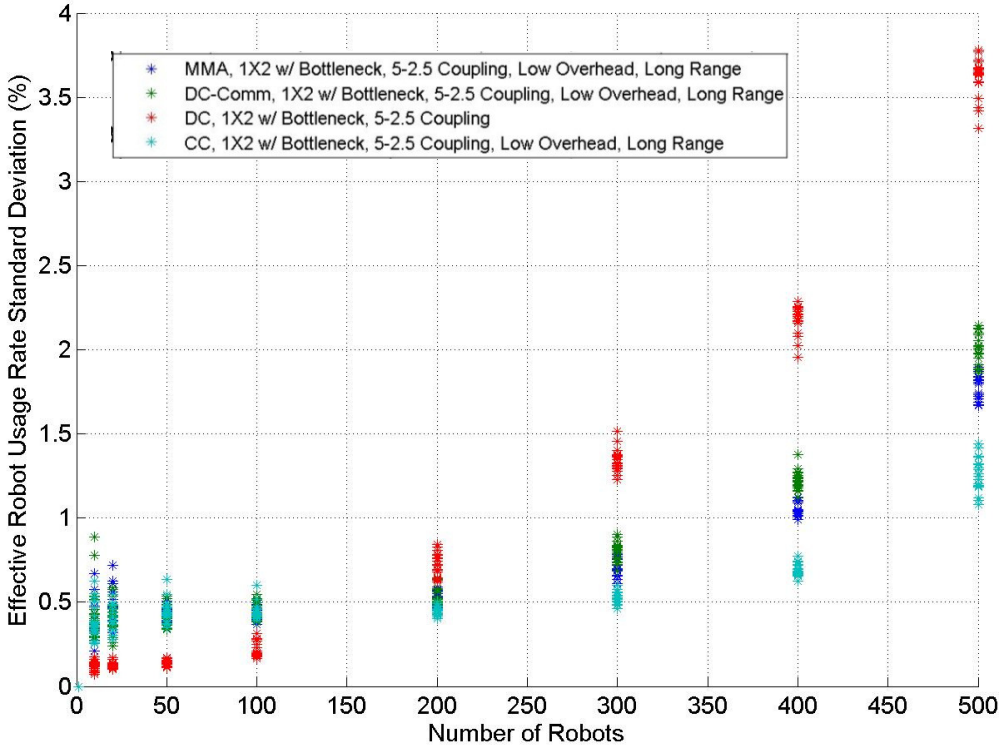


Figure 5-17. Standard Deviation of the Effective Robot Usage Rates for Each Controller for the Container Terminal Problem

Bandwidth usage is the other significant aspect of performance and scalability for dynamic multi-robot autonomous routing. It, however, is not as big a factor for the automated container terminal problem since none of the controllers reach the available bandwidth limit of 1 Mbps. Figure 5-18 shows the average bandwidth usage for each of the controllers for the automated container terminal problem. Centralized control has the highest bandwidth usage followed by distributed control with communication. Mixed mode autonomy has the lowest bandwidth usage of the controllers with communication. If the bandwidth trends were carried out centralized control could scale to 1100 robots, distributed control with communication to 1400 robots, and mixed mode autonomy to 2700 robots before the bandwidth limits were reached. It is likely that the MRS would reach a point where the system would not be able to accommodate the number of robots while maintaining the separation distance for all them before it reached the upper limit for mixed mode autonomy. An analysis of the environment suggests that for typical combination of tasks as many as 6 robots would have to be on each road segment – the upper limit with the separation distance is 5.

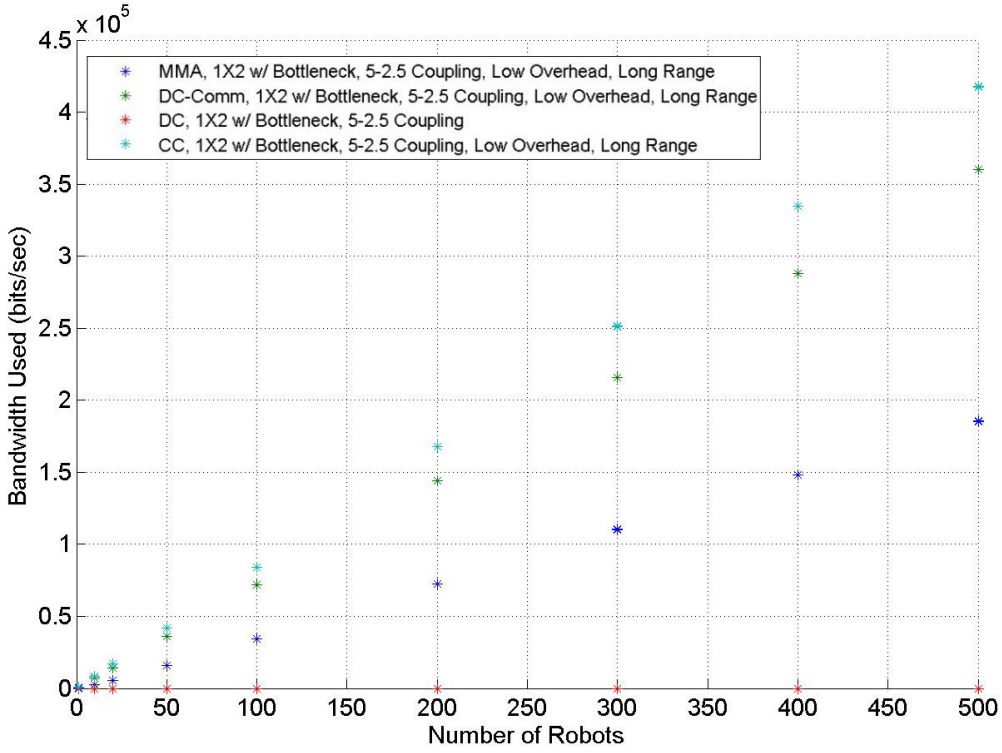


Figure 5-18. Bandwidth Usage for Each of the Controllers for the Container Terminal

The bandwidth is used by the robots to send status update messages and the centralized controller to send command messages. Figure 5-19 shows the number of messages sent by the robots for each of the controllers for the automated container terminal. Figure 5-20 shows the number of messages sent by the centralized controller for each of the controllers.

The number of update messages sent per robot with centralized control and distributed control with communication are constant since the bandwidth limitations are never exceeded. Mixed mode autonomy gradually increases to a fairly steady rate starting around 200 robots. The update message rate is a function of the number of robots in the vicinity of the robot and how far the robot is from the central controller. Distributed control with communication does not have any command messages from the central controller. The command messages are a small part of the total bandwidth use for mixed mode autonomy, but a significant contributor to the higher effective robot usage rates. The total number of command messages set by centralized control drops as the number of robots increases since the effective robot usage drops with increasing number of robots. This is due to the fact that the robots complete few tasks and thus enter fewer intersections where they would need commands when the effective robot usage rate is lower.

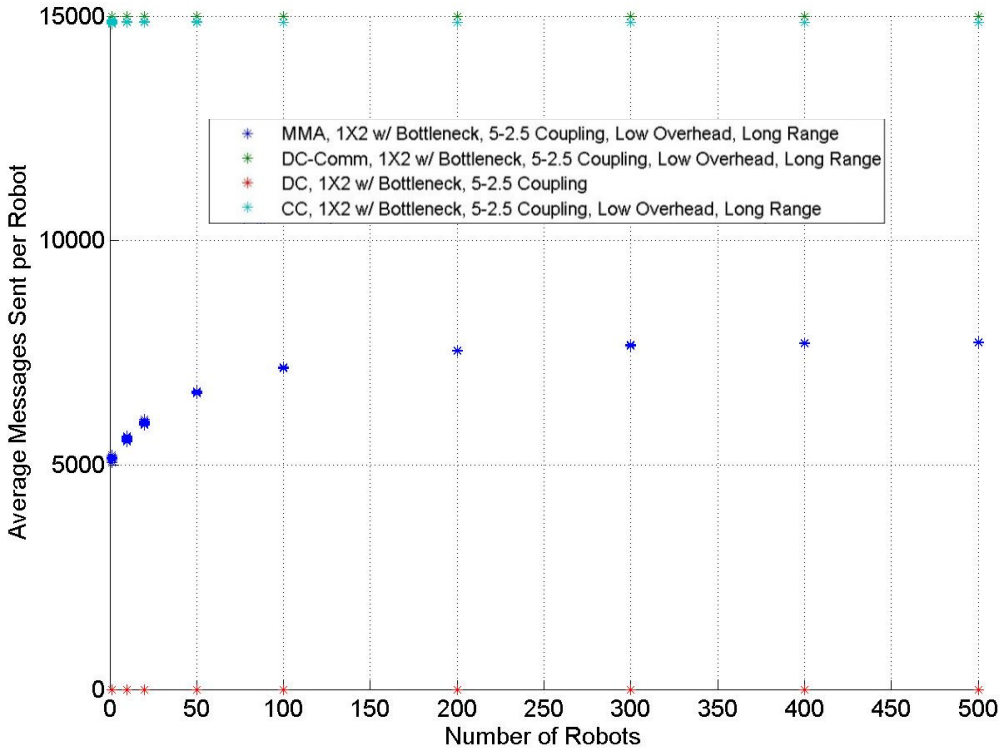


Figure 5-19. Messages Sent per Robot for Each of the Controllers for the Container Terminal

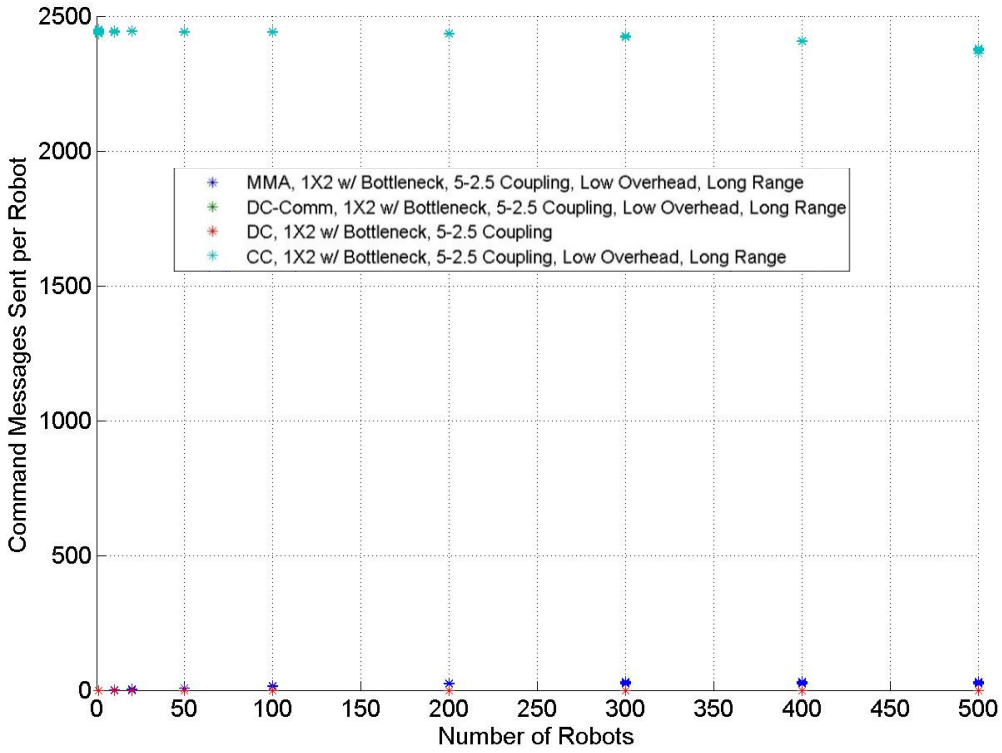


Figure 5-20. Messages Sent by the Centralized Controller for Each of the Controllers for the Container Terminal

The remainder of this section examines the sensitivity of the performance to the parameters selected to represent the automated container terminal problem. This is to show how likely the appropriate controller for the automated container terminal problem might change as a parameter changes. The parameters examined are task coupling, communication, and environment settings.

The task coupling is how tightly grouped the tasks are to one point in the environment. The assumption for the automated container terminal was that the tasks were tightly coupled at the loading berth and moderate coupled at container yard (5-2.5 task coupling). Different methods of container terminal operation could impact this assumption. For example the containers could be stored anywhere in the container yard, which would lead to a more uniform task coupling for that half of the tasks. Figure 5-21a-c shows the sensitivity of the MRS performance to the task coupling for the different controllers.

All of the controllers show that the more tightly coupled the tasks the lower the effective robot usage rate. The case used for the automated container terminal is the second most tightly coupled (magenta plot). The same performance trends hold each of the tightly and moderately grouped task coupling pairs across all of the controllers – mixed mode autonomy and centralized control perform the best – roughly 1 % better than distributed control with communication for higher number of robots. However, when one of the tasks is uniformly distributed mixed mode autonomy is has a higher effective robot usage rate than centralized control. These same trends also hold for the rate of decline of the effective robot usage rate.

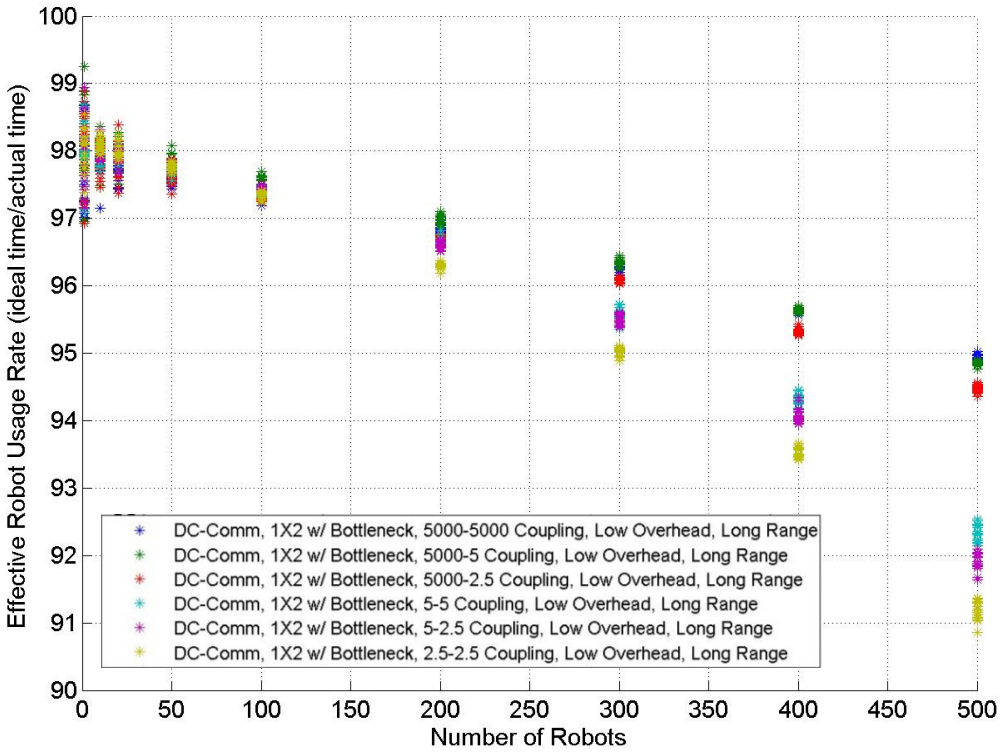


Figure 5-21a. Effective Robot Usage Rates for Distributed Control with Communication for the Container Terminal Problem with Varying Task Coupling

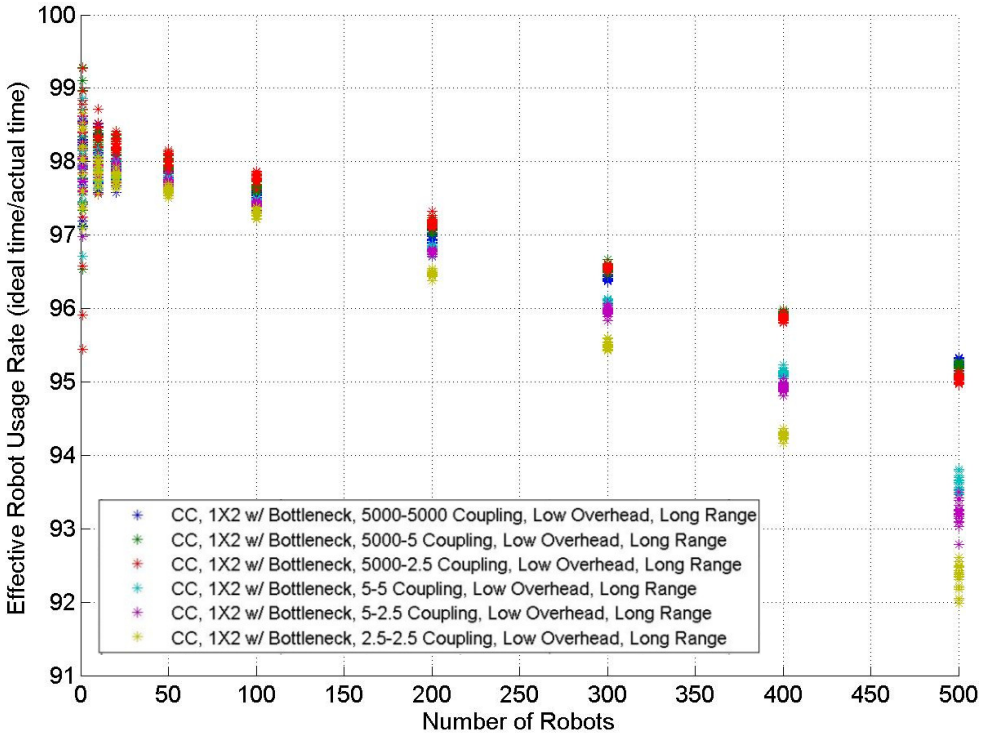


Figure 5-21b. Effective Robot Usage Rates for Centralized Control for the Container Terminal Problem with Varying Task Coupling

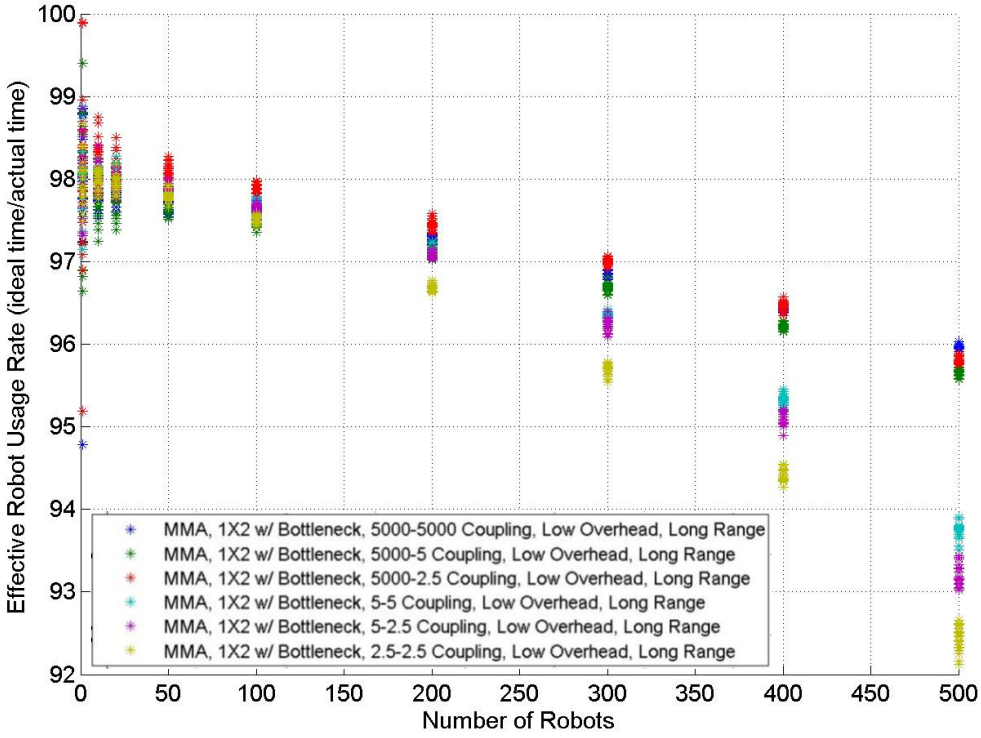


Figure 5-21c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Container Terminal Problem with Varying Task Coupling

The communication settings are a function of the wireless protocol used for the communication system and some assumptions about the communication requirements other than to solve the dynamic multi-robot autonomous routing problem. For the automated container terminal, the assumption is that the messages will be almost entirely about routing (low overhead). The other assumption is that a wireless communication should cover the entire environment (long range communication). Figure 5-22a-c shows the sensitivity of the MRS performance to the communication settings (range and overhead) for the different controllers. The automated container terminal parameters (low overhead, long range) are the blue plot on the graphs.

The communication settings do not have much effect on the effective robot usage rate for decentralized control with communication or mixed mode autonomy for the automated container terminal problem. For mixed mode autonomy, there was a small improvement for the low overhead, long range case. Centralized control, which was only used with long range communication, had lower effective robot usage rates with the high overhead case. The difference grows with the number of robots. Unlike in the nominal case for the automated container terminal, centralized control with high overhead performs more poorly than mixed mode autonomy. The others trends for the nominal case are consistent for the other communication setting and controllers.

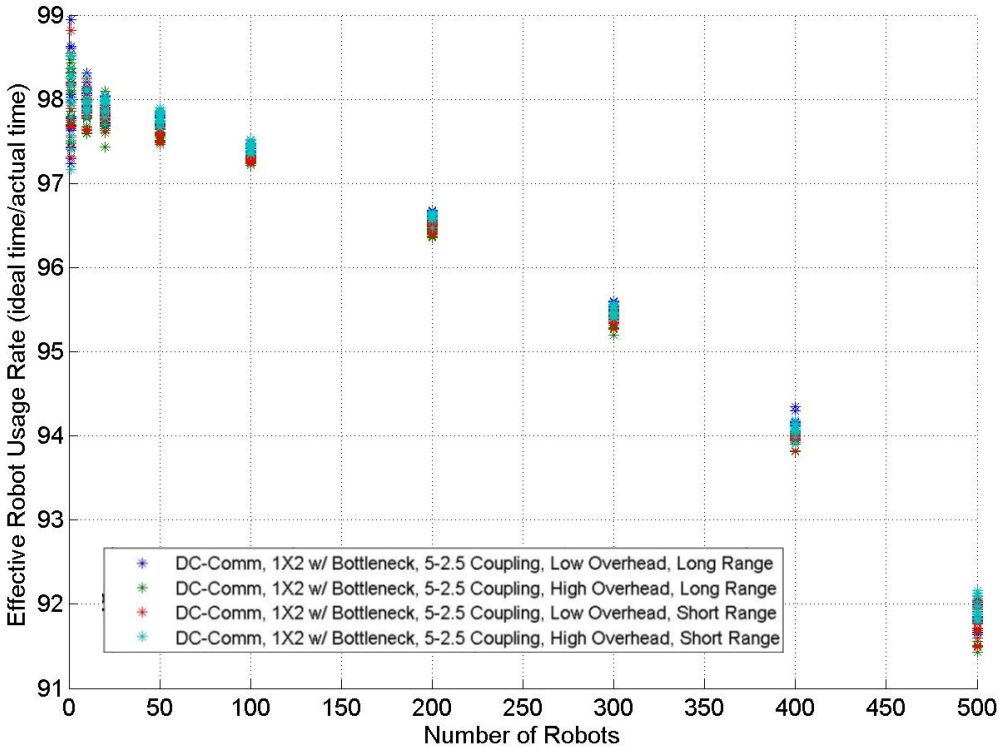


Figure 5-22a. Effective Robot Usage Rates for Distributed Control with Communication for the Container Terminal Problem with Varying Communication Settings

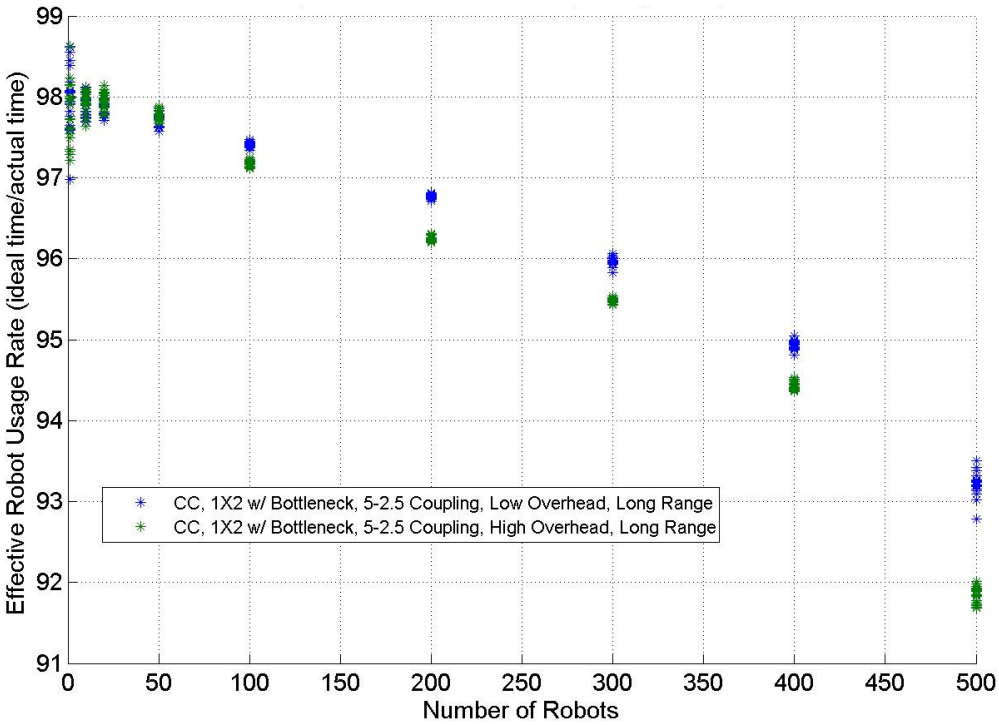


Figure 5-22b. Effective Robot Usage Rates for Centralized Control for the Container Terminal Problem with Varying Communication Settings

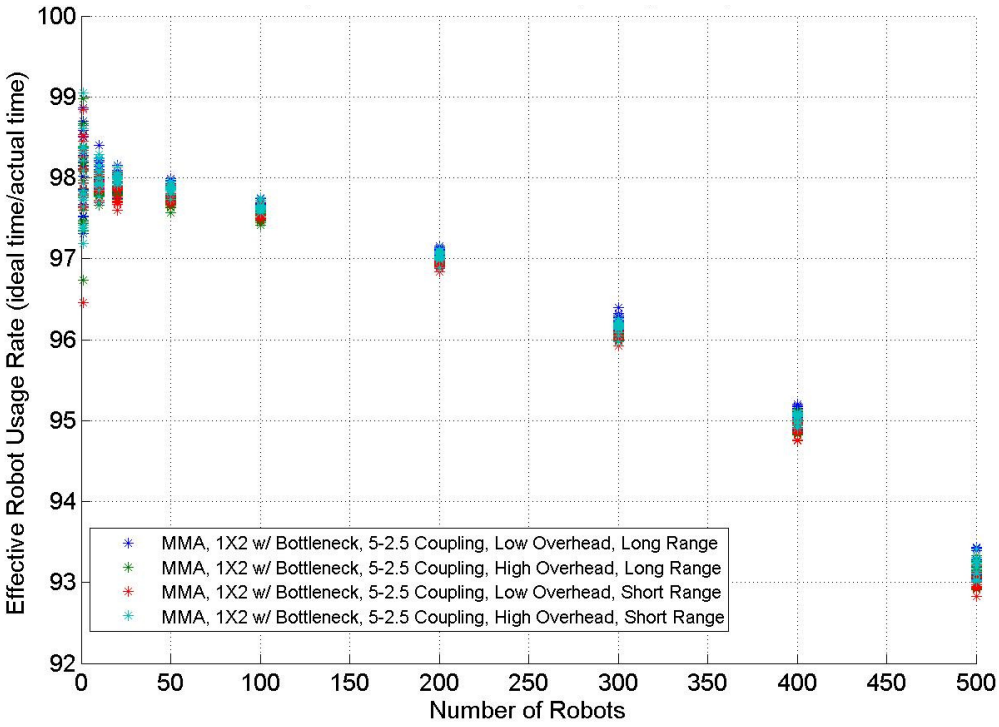


Figure 5-22c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Container Terminal Problem with Varying Communication Settings

The environment parameters are shaped by the constraints on the robot travel for the dynamic multi-robot autonomous routing problem. The automated container terminal problem assumes that the robots can move somewhat freely in environment (1 X 2 connection pattern) and that the interchange area is a bottleneck. The storage yard is organized in rows which restricts travel some in one direction which is why the 1 X 2 pattern is used. Figure 5-23a-c shows the sensitivity of the MRS performance to the environment parameters (bottleneck width and grid connection pattern) for the different controllers. The base case (1 X 2 with a bottleneck) for the automated container terminal is in red.

For all the controllers, the presence or absence of the bottleneck has the most impact to reduce the effective robot usage rate for the open pit mine problem. Reducing the connectivity of the environments also reduces the effective robot usage rates.

Mixed mode autonomy and centralized control perform similarly for most environments except for the sparsest environments with or without a bottleneck (1 X 4 connection pattern) when mixed mode autonomy is better. For the sparsest environment with a bottleneck, distributed control with communication is the best. The other controllers outperform it for all other environments.

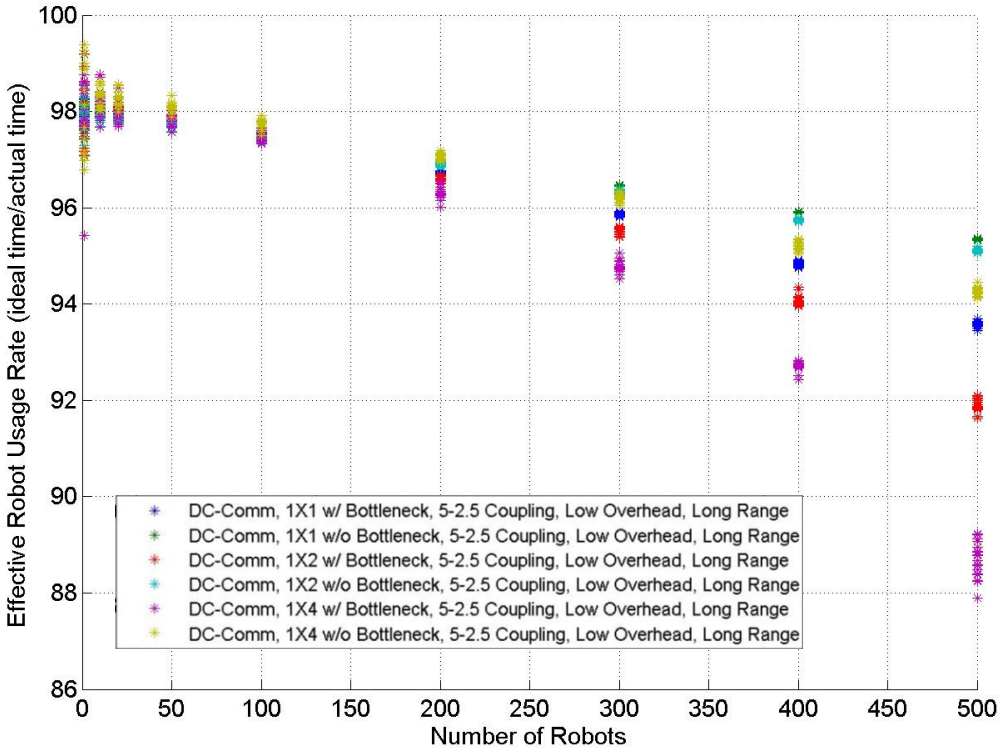


Figure 5-23a. Effective Robot Usage Rates for Distributed Control with Communication for the Container Terminal Problem with Varying Environmental Parameters

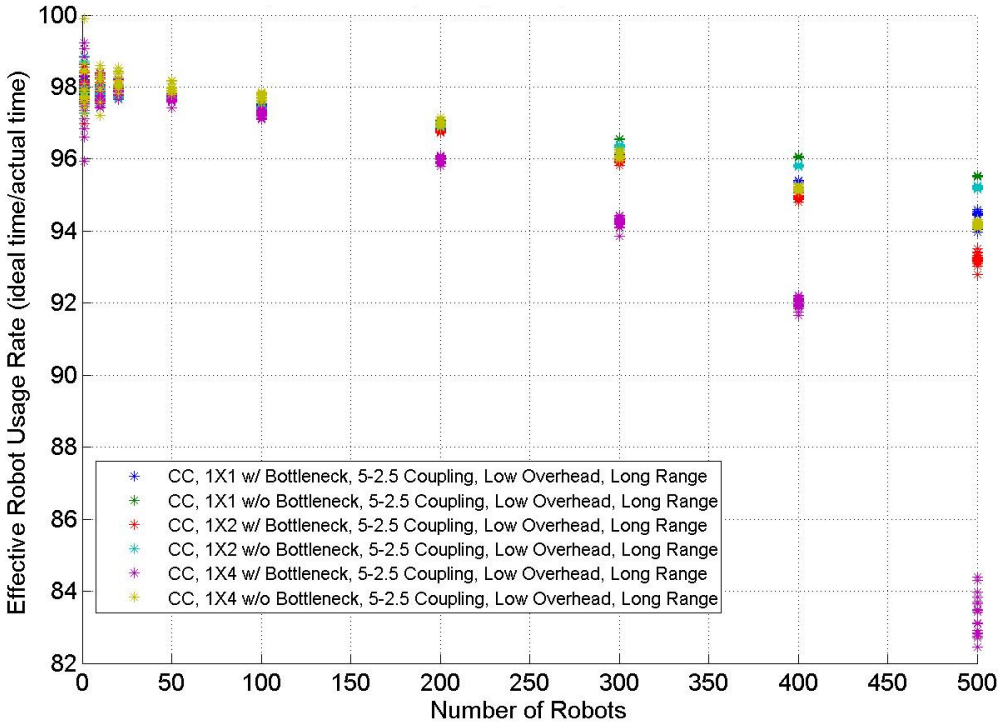


Figure 5-23b. Effective Robot Usage Rates for Centralized Control for the Container Terminal Problem with Varying Environmental Parameters

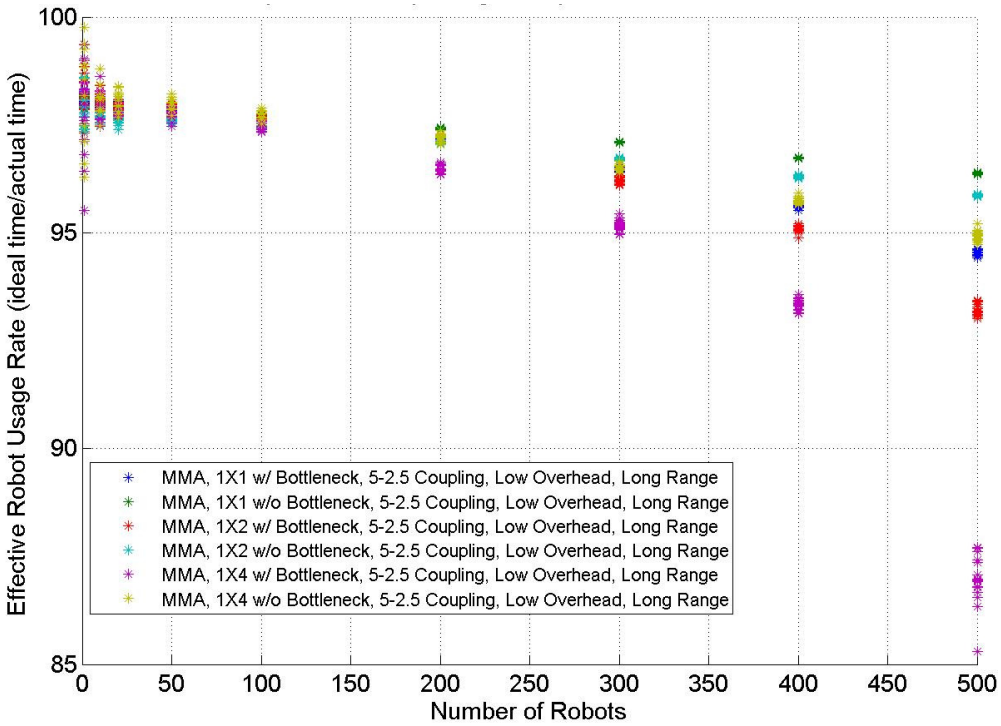


Figure 5-23c. Effective Robot Usage Rates for Mixed Mode Autonomy for the Container Terminal Problem with Varying Environmental Parameters

5.6 Automated Warehouse

Application

Automated warehouses use robots to pickup items from the storage areas and bring them to a packing station so they can be shipped. Items need to either taken from storage to the shipping area to be packed for shipment or incoming items need to sorted and moved to storage in the warehouse. An example warehouse is show in Figure 5-24. A layout for an automated warehouse is shown in Figure 5-25. The dynamic multi-robot autonomous routing problem is the transport items between the shipping and sorting area to the storage area or vice versa.

For simulation, the dynamic multi-robot autonomous routing problem can be described as a highly constrained (1 X 4 connection pattern) environment without a bottleneck, with tightly and uniformly coupled tasks, a short range communication system, and a low communication overhead. There is no bottleneck since the robots can move freely from the shipping and sorting area to the storage area. The 1 X 4 pattern is used since most warehouses are organized in long rows with limited areas to cross between rows. The communication system is primarily to coordinate with nearby robots so short range communication is used. Since centralized control is not used with short range communication, the long range case was used as an additional comparison for that controller. The messages would be primarily focused on the dynamic multi-robot autonomous routing problem so the overhead rate is low. The destinations (tasks) are tightly grouped about the shipping and sorting areas since many robots could be used to pack a shipment and uniformly grouped about the storage since items could be stored anywhere with that area. For these reasons, the container terminal uses ∞(5000)-2.5 task coupling.

Figure 5-24. Example Warehouse [**56**] (Green Logistics. (2011) Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:Warehouse_md17.jpg, Used under fair use guidelines, 2011.)

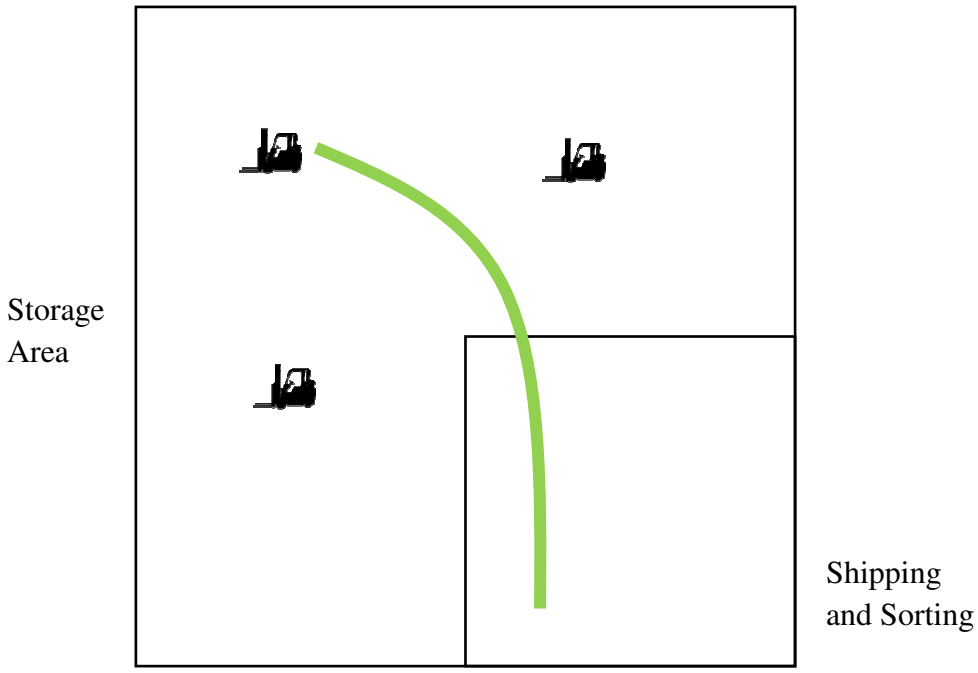


Figure 5-25. Automated Warehouse Layout

Simulation Results

The dynamic multi-robot autonomous routing problem was simulated for automated warehouse parameters at least twenty times for each of the controllers. This section presents the metrics derived from the data logged for each simulation.

The effective robot usage rate estimates the efficiency of the MRS for each of the controllers for the automated warehouse problem and is shown in Figure 5-26. The effective robot usage rate for the automated warehouse problem with long range communication is shown in Figure 5-27. Table 5-4 lists the rate of decrease of the effective robot usage rate for each of the controllers.

For the automated warehouse problem, mixed mode autonomy clearly outperforms the distributed controllers. Distributed control with communication barely does better than distributed control without communication. With long range communication, mixed mode autonomy performs about the same as with short range communication. Centralized control is a little worse than mixed mode autonomy, but is better than distributed control.

The rate of decrease of effective robot usage rate is lowest for mixed mode autonomy. This lower rate of decrease means that the difference in performance will continue to grow larger as the number of robots in the MRS increases. The rate of decrease would get sharper for centralized control when the bandwidth for the communication system is saturated at around 1100 robots. Although distributed control with communication performs similarly to distributed control without communication, it will maintain more of its performance as the MRS gets larger.

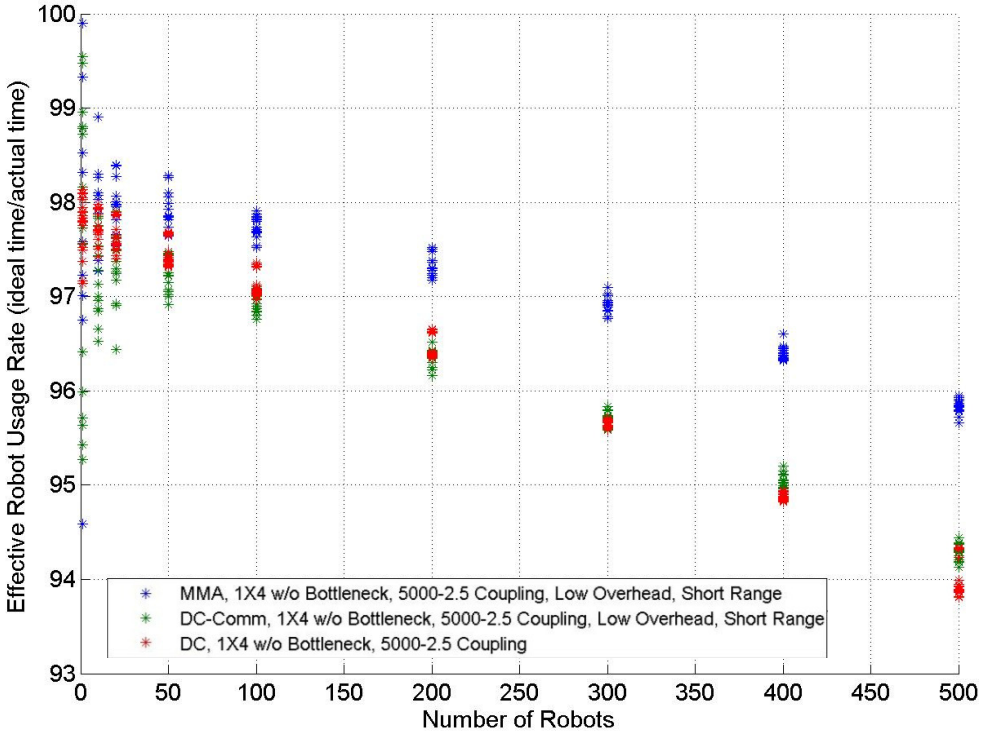


Figure 5-26. Effective Robot Usage Rates for Each Controller for the Automated Warehouse Problem

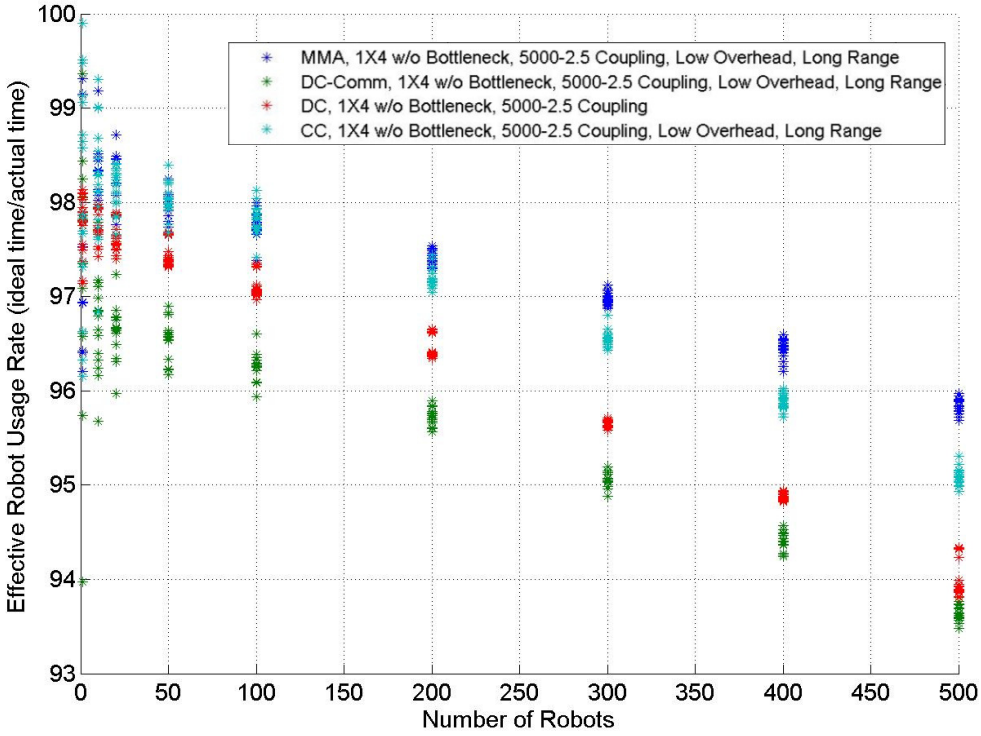


Figure 5-27. Effective Robot Usage Rates for Each Controller for the Automated Warehouse Problem with Long Range Communication

Table 5-4. Rate of Decrease of Effective Robot Usage Rate for the Automated Warehouse Problem

Controller	**Rate of Decrease of Effective Robot Usage Rate (% / additional robot)**
Distributed Control without Communication	0.0088
Distributed Control with Communication	0.0076
Centralized Control	0.0079*
Mixed Mode Autonomy	0.0058

* for long range communication instead of short range

The standard deviation of the effective robot usage rate is a measure of how evenly the load is spread across the robots in the MRS. Figure 5-28 shows the standard deviation of the effective robot usage rate for each controller for the automated warehouse problem.

Mixed mode autonomy has much lower variation in effective robot usage rate for all numbers of robots. As with their overall performance, distributed control with and without communication are more variable and largely indistinguishable from each other.

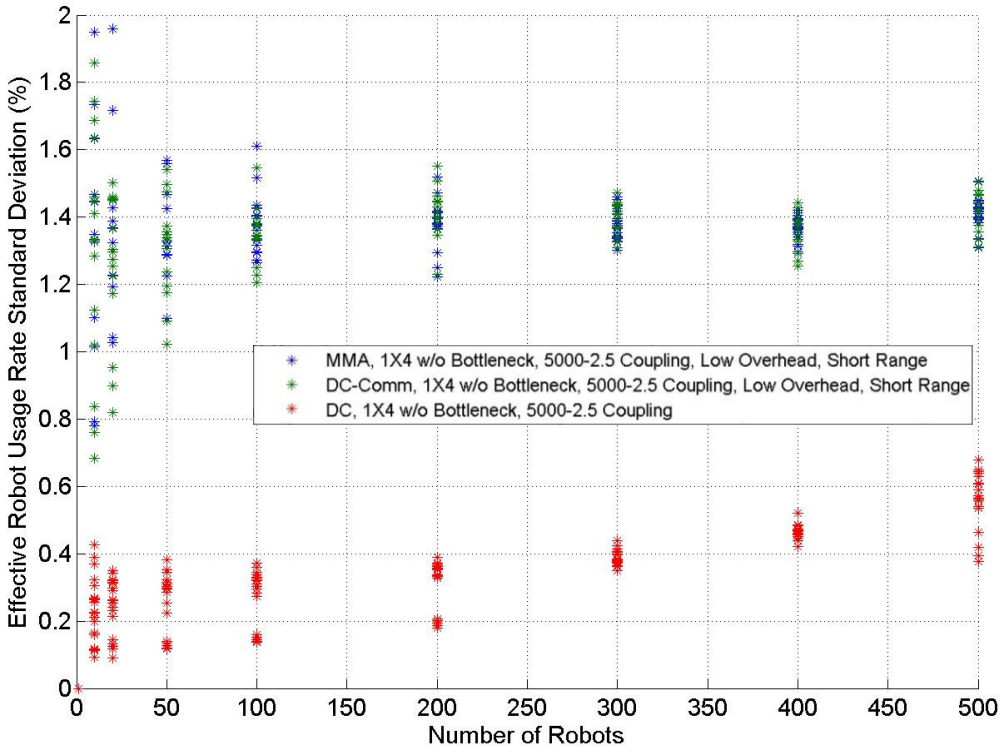


Figure 5-28. Standard Deviation of the Effective Robot Usage Rates for Each Controller for the Warehouse Problem

Bandwidth usage is the other significant aspect of performance and scalability for the automated warehouse. It is not as big a factor for the automated warehouse problem since none of the controllers reach the available bandwidth limit of 1 Mbps. Figure 5-29 shows the average bandwidth usage for each of the controllers for the automated warehouse problem for short range communication (distributed control with communication and mixed mode autonomy). Figure 5-30 shows the bandwidth usage for the long range case that includes centralized control. The bandwidth usages for distributed control and mixed mode autonomy are basically identical for the short and long range cases.

The bandwidth usage is largely the same as with the automated container terminal for each of the controllers except mixed mode autonomy, which requires more bandwidth for the automated warehouse. Centralized control has the highest bandwidth usage followed by distributed control with communication. Mixed mode autonomy has the lowest bandwidth usage of the controllers with communication. If the bandwidth trends were carried out centralized control could scale to 1100 robots, distributed control with communication to 1400 robots, and mixed mode autonomy to 2500 robots before the bandwidth limits were reached. It is likely that the MRS would reach a point where the system would not be able to accommodate the number of robots while maintaining the separation distance for all them before it reached the upper limit for mixed mode autonomy. An analysis of the environment for the automate warehouse problem suggests that for typical combination of tasks as many as 8 robots would have to be on each road segment – the upper limit with the separation distance is 5. The increase in number of robots over the automated container terminal is due to the sparseness of the environment.

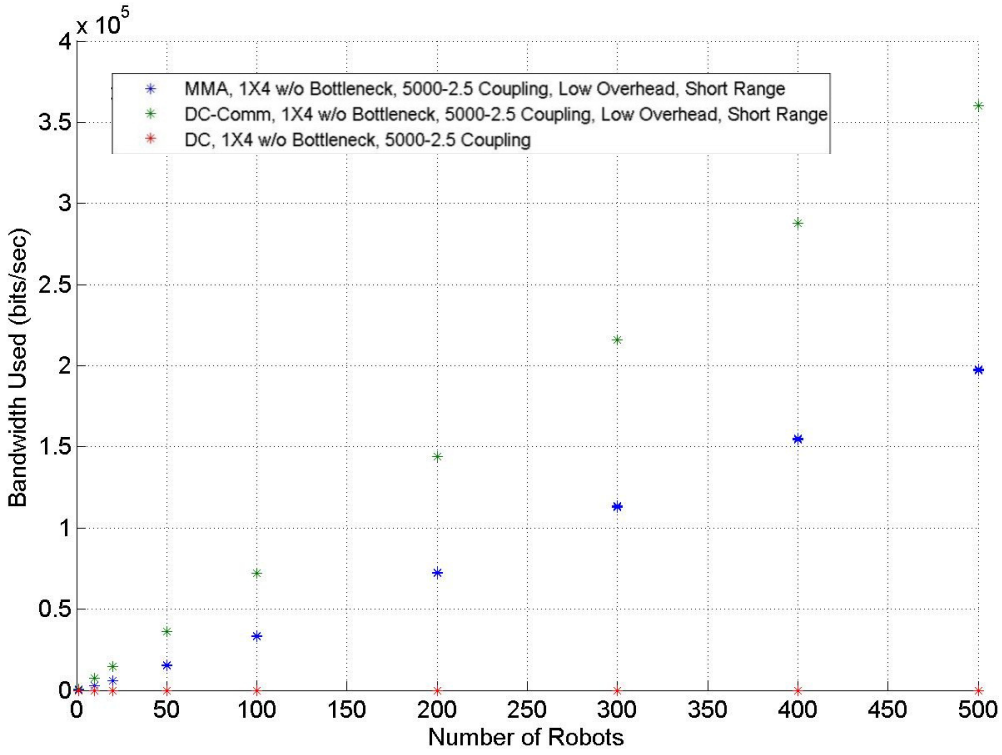


Figure 5-29. Bandwidth Usage for Each of the Controllers for the Automated Warehouse

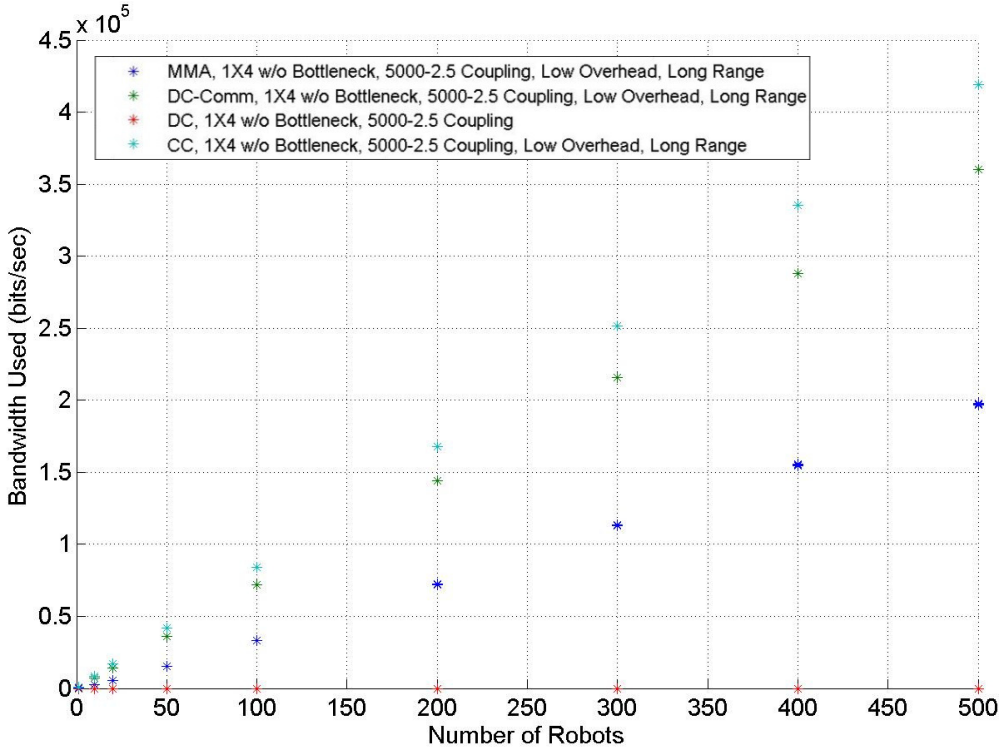


Figure 5-30. Bandwidth Usage for Each of the Controllers for the Automated Warehouse with Long Range Communication

The bandwidth is used by the robots to send status update messages and the centralized controller to send command messages. Figure 5-31 shows the number of messages sent per robot for each of the controllers for the automated warehouse. Figure 5-32 shows the number of command messages sent by the centralized controller for each controller.

The number of update messages sent per robot with distributed control with communication is constant since the bandwidth limitations are never exceeded. The number of updates for mixed mode autonomy gradually increases as the number of robots increases. The update message rate is a function of the number of robots in the vicinity of the robot and how far the robot is from the central controller. Mixed mode autonomy uses more update messages with the automated warehouse than the automated container terminal because of the additional congestion caused by the sparser environment.

Distributed control with communication does not have any command messages from the central controller. The command messages are a small part of the total bandwidth use for mixed mode autonomy, but a significant contributor to the higher effective robot usage rates. There is a high variability in how many times each robot receives a command with mixed mode autonomy. The ranges for an individual iteration of 500 robots are from zero to 45 with an average of about five.

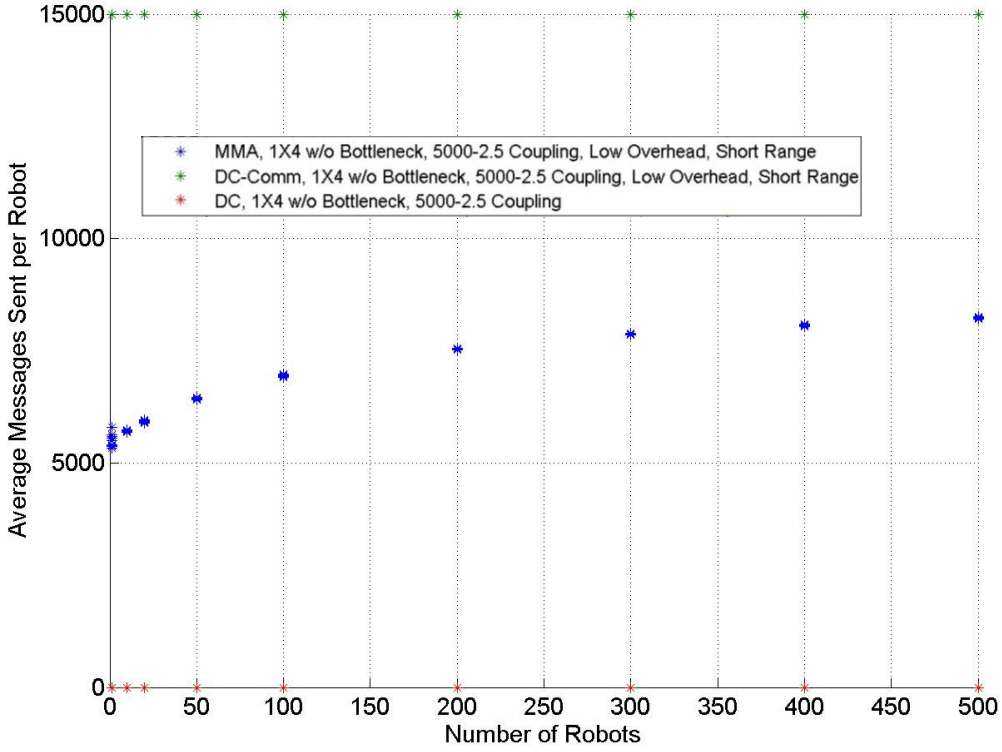


Figure 5-31. Messages Sent per Robot for Each of the Controllers for the Warehouse

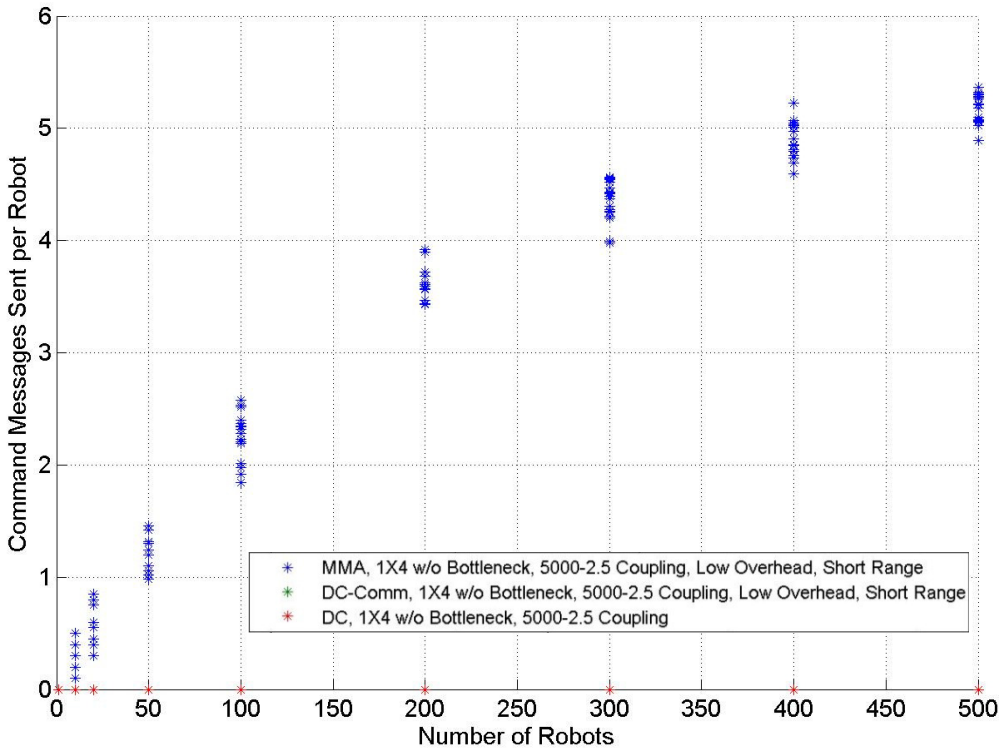


Figure 5-32. Messages Sent by the Centralized Controller for Each of the Controllers for the Warehouse

The rest of this section examines the sensitivity of the performance to the parameters selected to represent the automated warehouse problem. This is to show how likely the appropriate controller for the automated warehouse problem might change as a parameter changes. The parameters examined are task coupling, communication, and environment settings.

The task coupling is how tightly grouped the tasks are to one point in the environment. The assumption for the automated warehouse was that the tasks were tightly coupled at the shipping and sorting area and uniformly coupled at storage area (5000-2.5 task coupling). Different methods of warehouse operation could impact this assumption. For example the most common items could be stored nearer to the shipping and sorting area which would lead to a moderate task coupling for that half of the tasks. Figure 5-33a-b shows the sensitivity of the MRS performance to the task coupling for the different controllers.

All of the controllers show that the more tightly coupled the tasks the lower the effective robot usage rate. The case used for the automated warehouse is the uniform and tightly coupled case (red plot).

The same performance trends hold each of the task coupling pairs for both of the controllers – mixed mode autonomy outperforms distributed control with communication, especially for higher numbers of robots. For the sparse environment with the automated warehouse, the difference between tight and moderate coupling for one of the two tasks was small for both controllers.

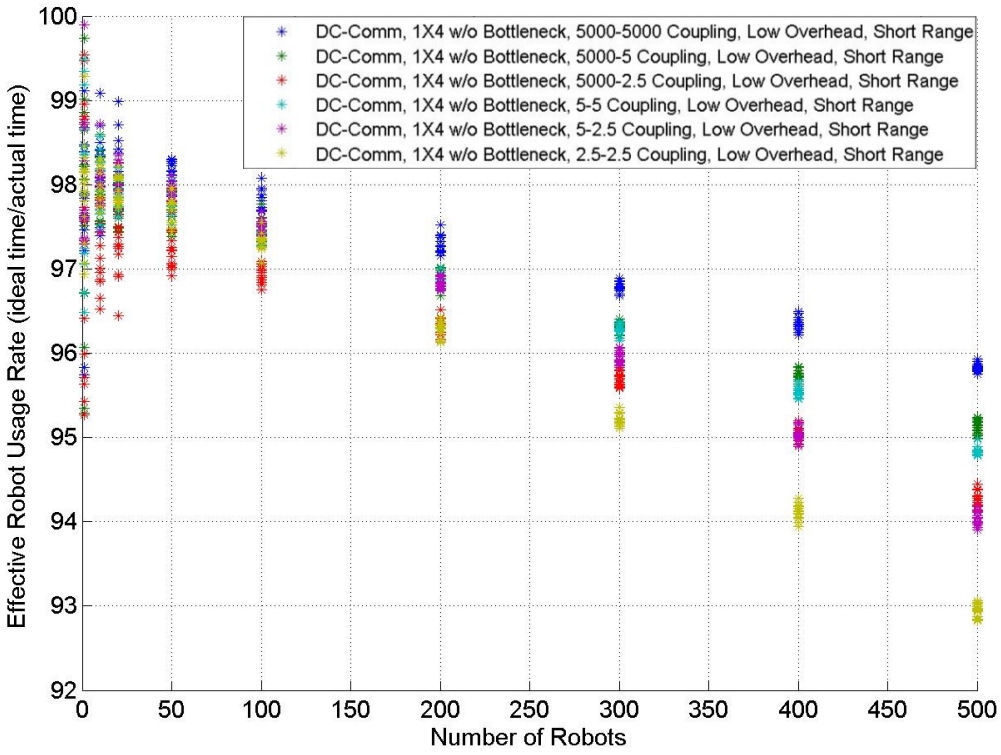


Figure 5-33a. Effective Robot Usage Rates for Distributed Control with Communication for the Warehouse Problem with Varying Task Coupling

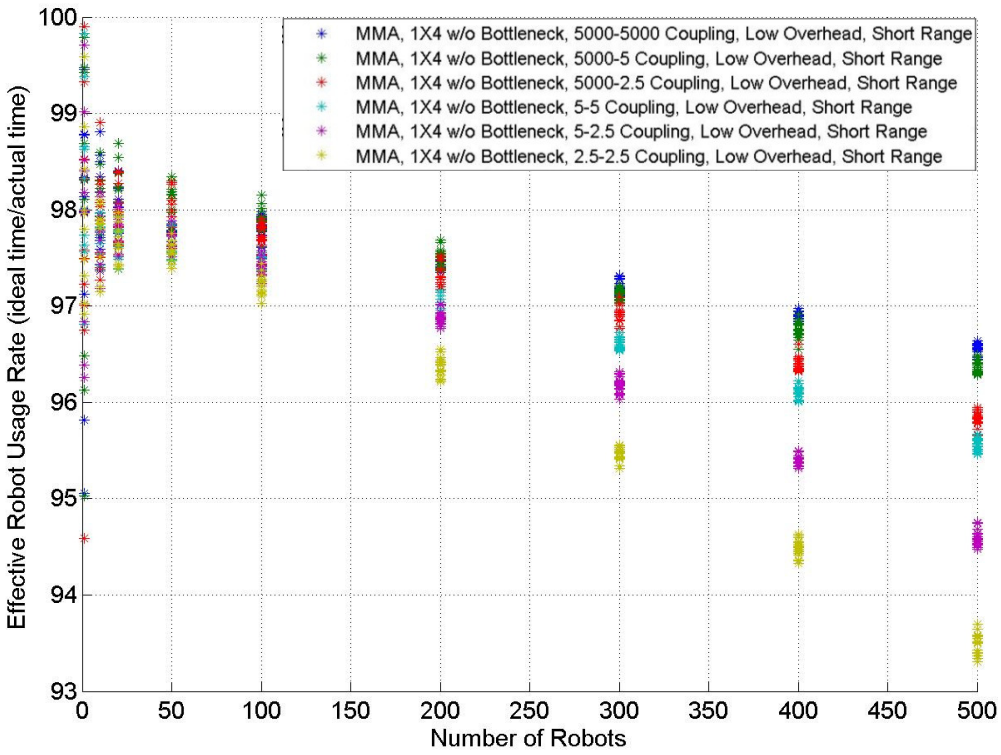


Figure 5-33b. Effective Robot Usage Rates for Mixed Mode Autonomy for the Warehouse Problem with Varying Task Coupling

The communication settings are a function of the wireless protocol used for the communication system and some assumptions about the communication requirements other than to solve the dynamic multi-robot autonomous routing problem. For the automated warehouse, the assumption is that the messages will be almost entirely about routing (low overhead). The other assumption is that a wireless communication could function with a shorter range communication. Figure 5-34a-b shows the sensitivity of the MRS performance to the communication settings (range and overhead) for the different controllers. The automated warehouse parameters (low overhead, short range) are the red plot on the graphs.

The communication settings do not have much effect on the effective robot usage rate for decentralized control with communication or mixed mode autonomy for the automated warehouse problem for most cases. For mixed mode autonomy there was a drop in effective robot usage rate for the high overhead, short range case that was consistent for all of the numbers of robots. This suggests that increasing the packet length for the messages for mixed mode autonomy has a negative impact for this environment and task coupling.

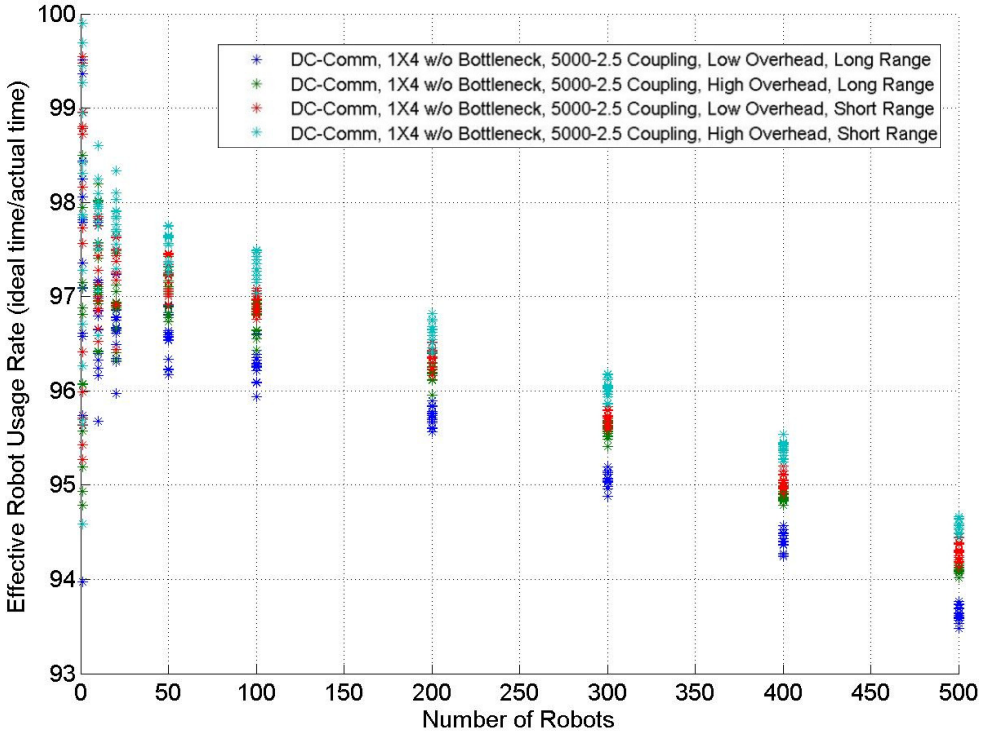


Figure 5-34a. Effective Robot Usage Rates for Distributed Control with Communication for the Warehouse Problem with Varying Communication Settings

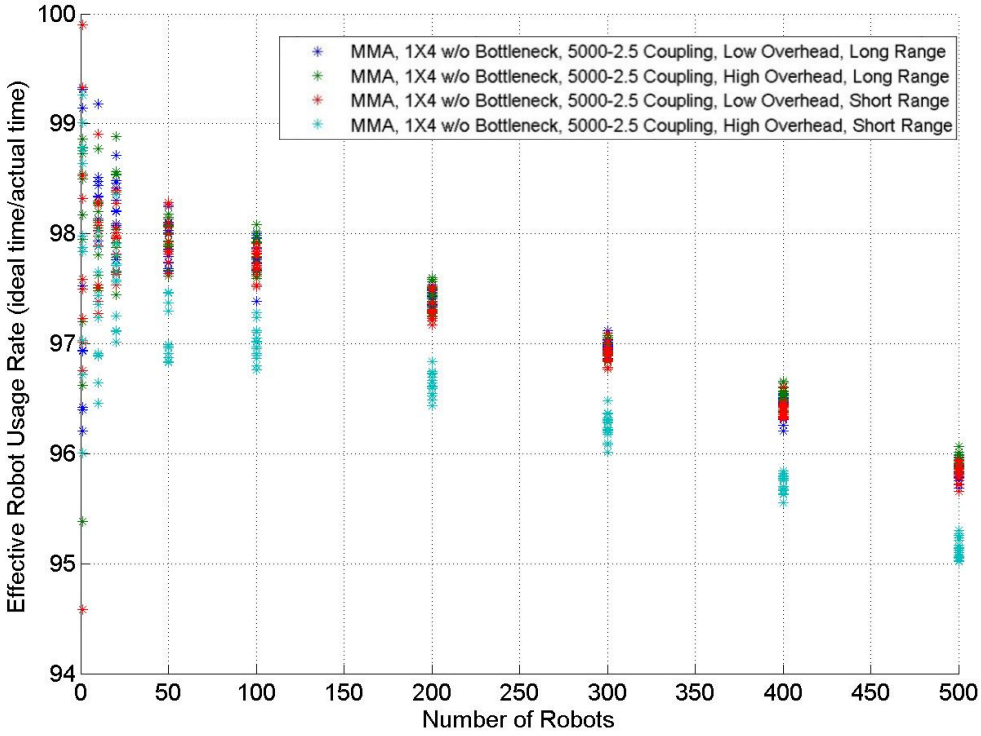


Figure 5-34b. Effective Robot Usage Rates for Mixed Mode Autonomy for the Warehouse Problem with Varying Communication Settings

The environment parameters are shaped by the constraints on the robot travel for the dynamic multi-robot autonomous routing problem. The automated warehouse problem assumes that the robot motion is restricted in one direction of travel in the environment (1 X 4 connection pattern) and that there is no bottleneck. The storage area is organized in long rows which restrict travel some in one direction which is why the 1 X 4 pattern is used. Figure 5-35a-b shows the sensitivity of the MRS performance to the environment parameters (bottleneck width and grid connection pattern) for the different controllers. The base case (1 X 4 without a bottleneck) for the automated container terminal is in yellow.

For both of the controllers, the environment has much less effect on the effective robot usage rate for the automated warehouse than with the other two example applications. This is mostly due to half of the tasks being uniformly coupled. When both tasks are uniformly distributed the environmental effects are even smaller. There is a gradual degradation in performance as the environments add a bottleneck and get sparser. The lower performance is most noticeable with 1 X 4 with a bottleneck environment, which is the most difficult environment for all of the applications. Mixed mode autonomy has higher effective robot usage rates for each of the environmental parameters sets than distributed control with communication.

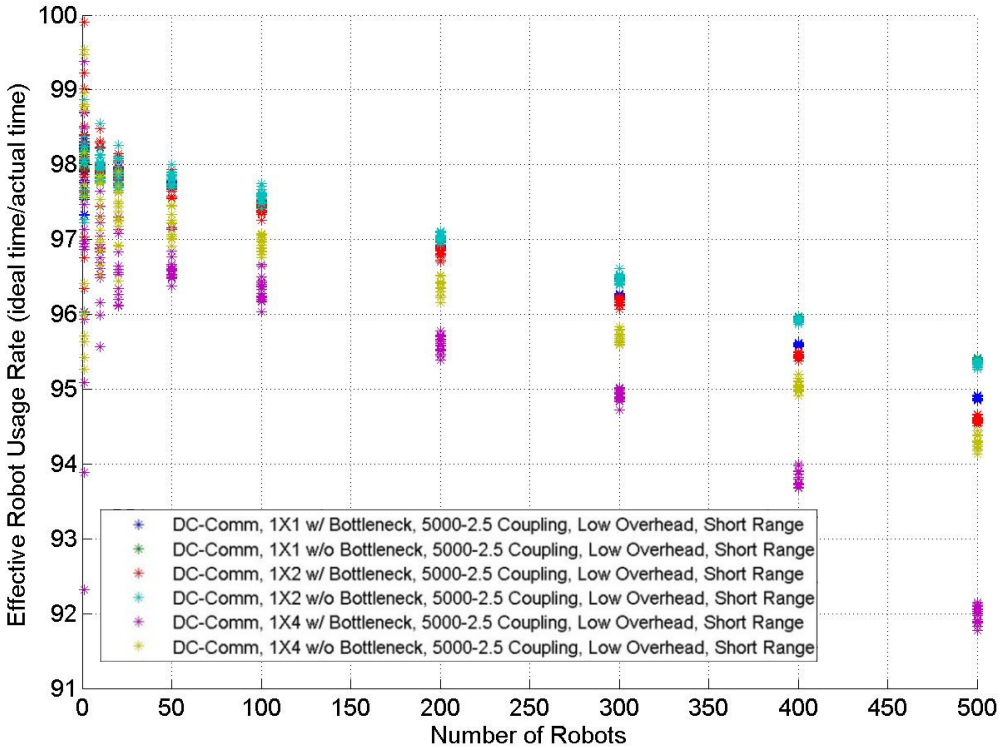


Figure 5-35a. Effective Robot Usage Rates for Distributed Control with Communication for the Warehouse Problem with Varying Environmental Parameters

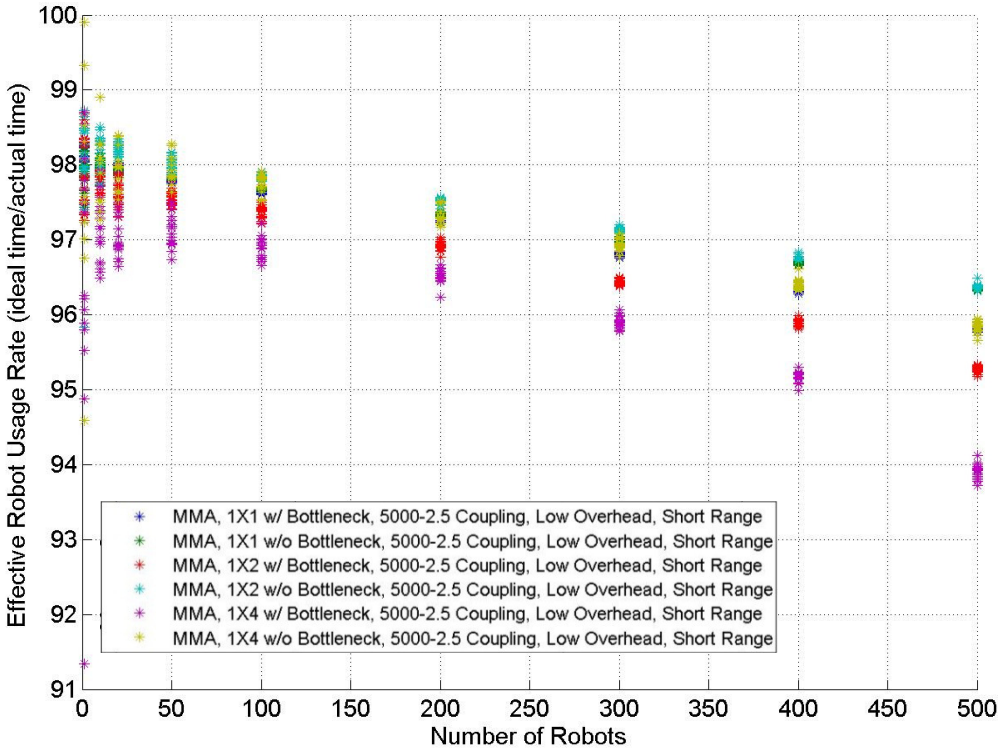


Figure 5-35b. Effective Robot Usage Rates for Mixed Mode Autonomy for the Warehouse Problem with Varying Environmental Parameters

5.7 Mixed Mode Autonomy with Model Updates

Mixed mode autonomy with model updates does not perform significantly better than the normal mixed mode autonomy control for the situations simulated as part of this work. The effective robot usage rate and bandwidth usage are compared for the automated warehouse problem in Figures 5-36 and 5-37. The differences are small and not statistically significant. This application is actually the set of conditions with the largest difference between the two variants of mixed mode autonomy. Mixed mode autonomy averages about 25 update messages per robot across the entire range of the number of robots. Almost all (90+%) of these updates are to the centralized controller when the robots are far from centralized controller. This is true for all of the short range communication cases. In the long range communication cases, there are almost no updates sent at all (one or two per robot). Basically, the simulation conditions do not stress the communication enough to see significant difference in between normal mixed mode autonomy and mixed mode autonomy with model updates. Changes to the simulation that would test this better are discussed in the Future Work section in the Chapter 6.

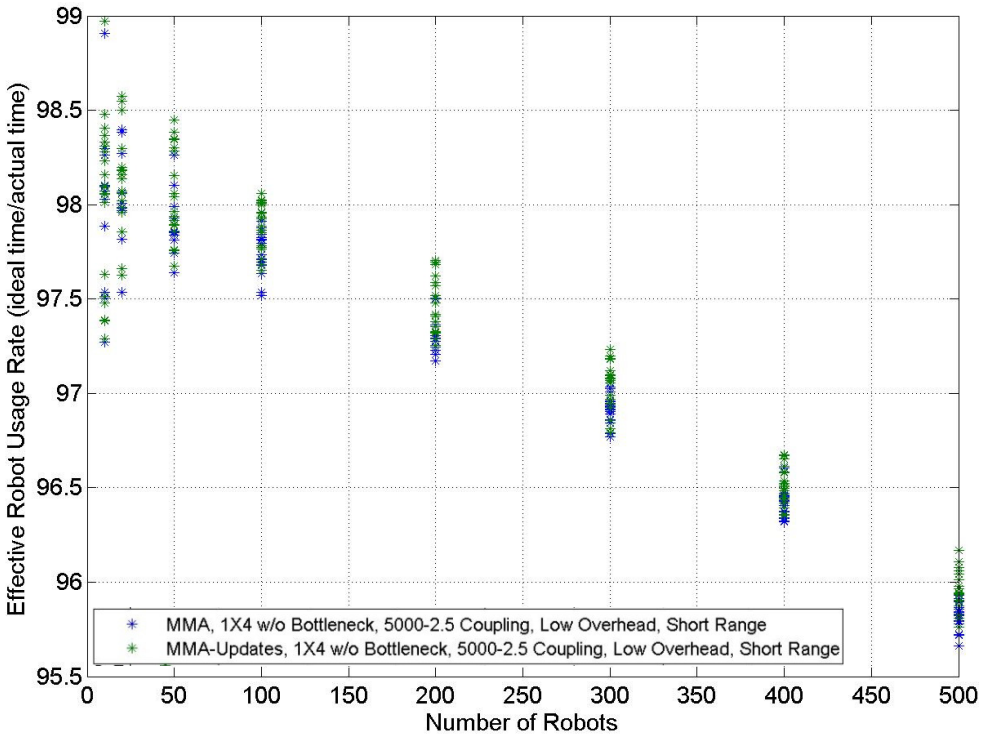


Figure 5-36. Comparison of Effective Robot Usage Rates for Mixed Mode Autonomy with and without Model Updates for the Automated Warehouse Problem

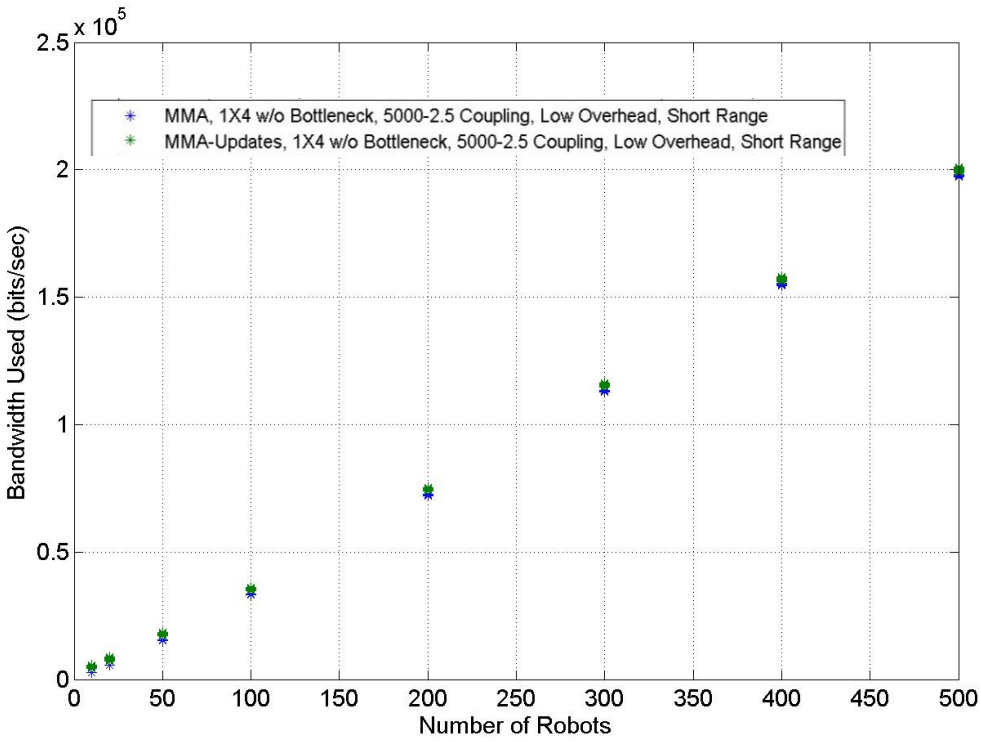


Figure 5-37. Comparison of Bandwidth Usage for Mixed Mode Autonomy with and without Model Updates for the Automated Warehouse Problem

5.7 Communication Update Rate

As part of the original development of the controller, several different update rates were considered for the centralized and distributed controllers. The status update rate for the simulation results presented in this chapter was ten times per road segment. The bandwidth requirements for this update rate are much higher for centralized control and distributed control with communication. Lower update rates were also simulated in a few cases to examine the effect of the update rate. The simulations were run for the open pit mine conditions with six and eight status updates per road segment. Figure 5-38 shows the effective robot usage rate for mixed mode autonomy and centralized control with update rates of ten (standard), eight, and six. Figure 5-39 shows the bandwidth usage for these cases. Figure 5-40 shows the effective robot usage rate for mixed mode autonomy and distributed control with communication with update rates of ten (standard), eight, and six. Figure 5-41 shows the bandwidth usage for these cases. The performance for lower update rates is worse than the standard update rate. When the bandwidth usage is similar for mixed mode autonomy and centralized or distributed control with communication, the performance differences are even larger than in the normal case.

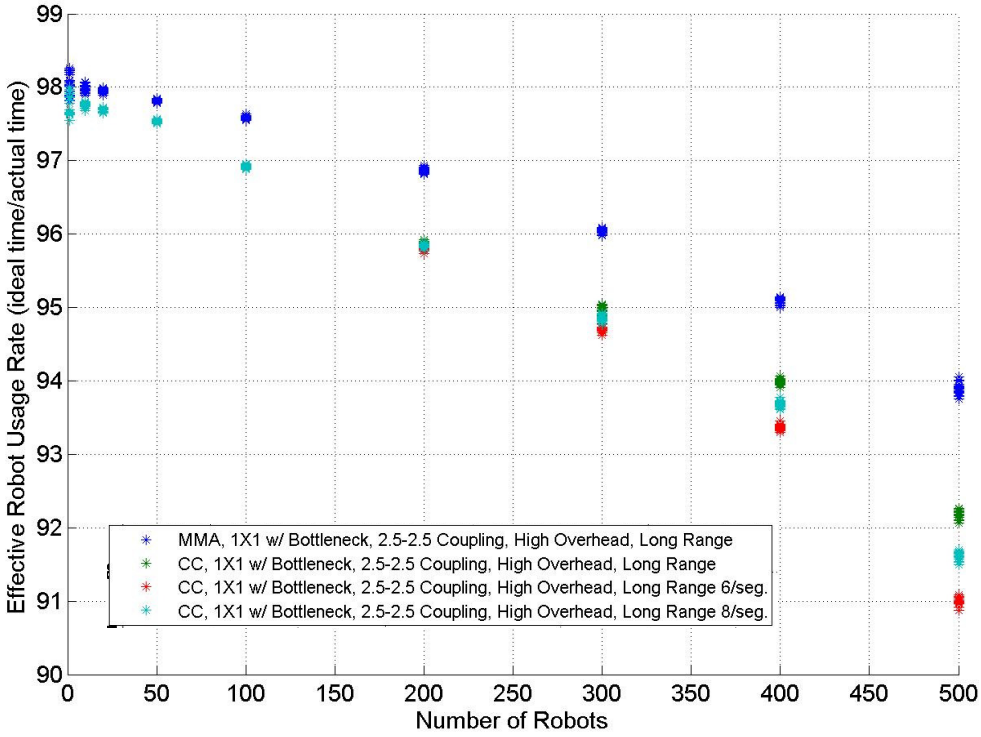


Figure 5-38. Effective Robot Usage Rate for Mixed Mode Autonomy and Centralized Control with Update Rates of 10, 8, and 6 per road segment

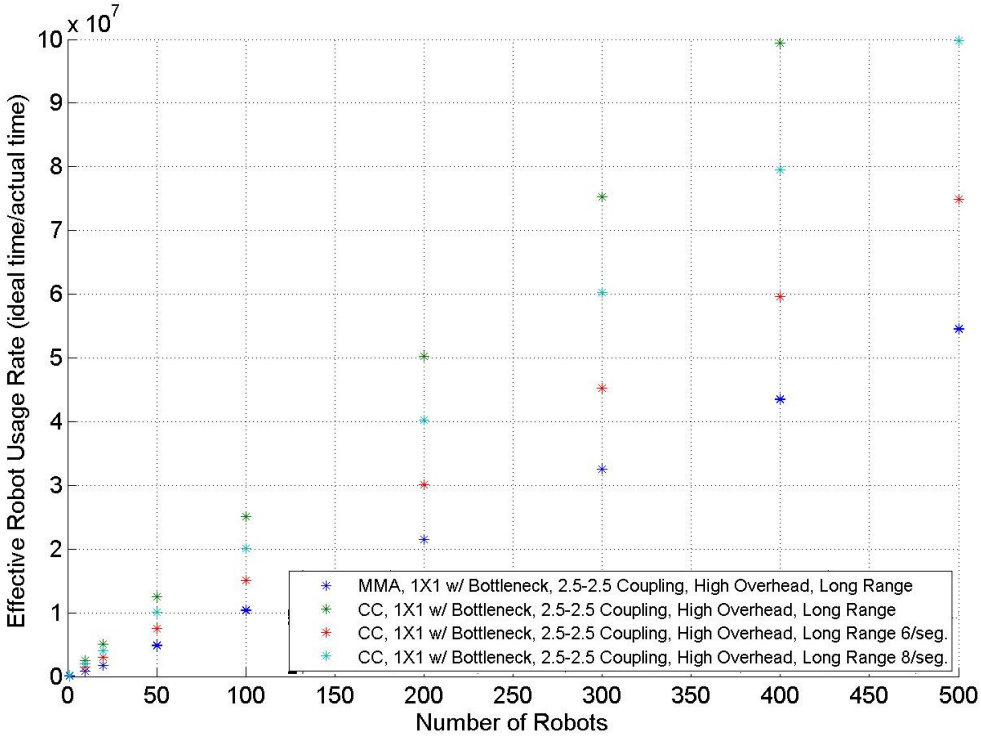


Figure 5-39. Comparison of Bandwidth Usage for Mixed Mode Autonomy and Centralized Control with Update Rates of 10, 8, and 6 per road segment

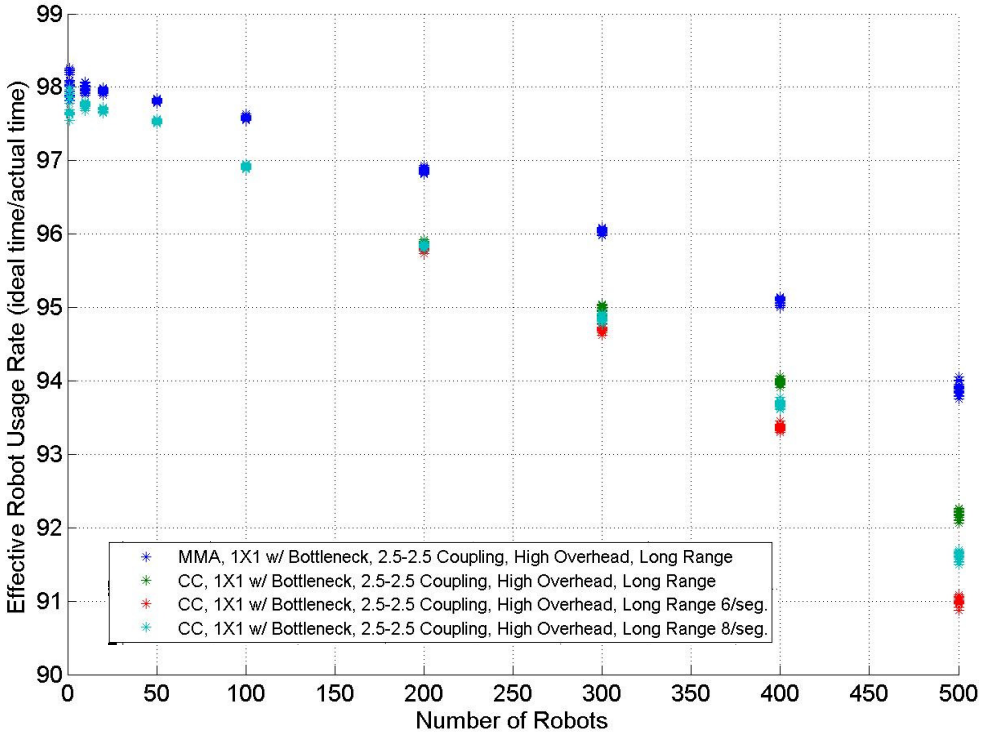


Figure 5-40. Effective Robot Usage Rate for Mixed Mode Autonomy and Centralized Control with Update Rates of 10, 8, and 6 per road segment

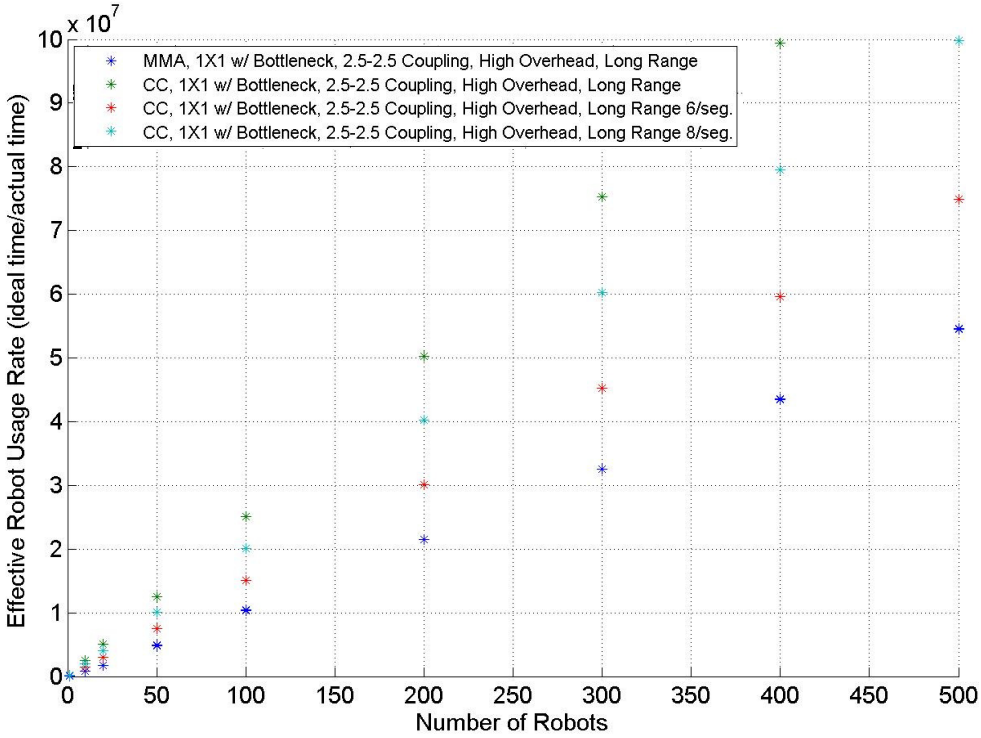


Figure 5-41. Comparison of Bandwidth Usage for Mixed Mode Autonomy and Centralized Control with Update Rates of 10, 8, and 6 per road segment

Chapter 6 – Conclusions and Recommendations for Future Work

Mixed mode autonomy showed significant improvements in performance for the dynamic multi-robot autonomous routing problem for the three example applications – open pit mine, automated container terminal, and automated warehouse when compared to the traditional centralized or distributed control architectures. Mixed mode autonomy also had the benefits of a slower decline in the effective robot usage rate as the number of robots increased and reduced bandwidth requirements for all numbers of robots. This helps the MRS scale to larger group sizes for the dynamic multi-robot autonomous routing problem.

This chapter discusses the recommendations for the example applications, presents some MRS design guidelines based on the simulations, makes recommendations on how best to continue this work, and states the primary contributions of this work to the state of the art.

6.1 Recommendations for the Example Applications

Out of all of the various combinations of simulation parameters, three were used to represent example applications for dynamic multi-robot autonomous routing. The three applications were automation of open pit mines, container terminals, and warehouses. The results of the simulations for these applications were presented in Chapter 5. This section discusses the recommendations based on the simulation results.

Overall mixed mode autonomy had higher effective robot usage rates for most all sets of parameters. For lower numbers of robots, the method of control did not have much impact on performance. This reflects much of the previous work done for problems similar to dynamic multi-robot autonomous routing. Mixed mode autonomy, however, still has the benefit of requiring less bandwidth even with low numbers of robots.

Open Pit Mine

The open pit mine was represented as tightly coupled tasks in a fully connected environment with a bottleneck using long range communication with high overhead. Mixed mode autonomy gave the best overall performance in terms of effective robot usage rate and bandwidth usage for the open pit mine. The high overhead for this problem, limited the effectiveness of both centralized control and distributed control with communication with higher numbers of robots. The rate of decrease of the effective robot usage rate as the number of robots increased was much lower for mixed mode autonomy. The primary benefit of centralized control for the open pit mine was lower variability of the effective robot usage rate compared to the other controllers. The available bandwidth was fully used for centralized control and distributed control with communication for the higher number of robots. This leads to missed messages that reduce the MRS performance.

The results of the simulations also identify trends that influence the design of the automated open pit mine system. System performance for all of the controllers is improved when the coupling

between tasks is reduced. For the open pit mine, this would mean spreading out the areas in the pit where the trucks load up ore and having multiple crushers or areas to unload into the crusher. This decoupling of tasks would be especially helpful as the number of robots in the MRS increases.

The communication settings for the system have a small impact on the effective robot usage rates for decentralized control with communication and mixed mode autonomy. The total differences in performance were at most 0.2%. This suggests that many different hardware and protocol combinations would work similarly for the open pit mine with these controllers. Centralized control, however, did not work acceptably with short range communication. The effective robot usage rates were low enough that the simulations for short range communication and centralized control were aborted (lower than 50% in several cases that were run). Increasing the communication overhead also reduced the effective robot usage rate. With low enough overhead the centralized controller got almost as good of a performance as mixed mode autonomy.

The road configurations (environment) also impact the performance of the MRS. Any bottleneck between the two task centers (the pit and crusher for the open pit mine) significantly reduces the effective robot usage rate for all controllers. Dynamic multi-robot autonomous routing problems would benefit from eliminating any bottlenecks. For the open pit mine, however, this is not really possible – all mines will have a limited number of haul roads out of the pit. The cost associated with building more roads would be too high to realistically expand this part of the road network. There is a smaller reduction in effective robot usage rate as the environments get more sparsely connected. Basically as the number of different routes to the task destination decrease, the congestion is increased which reduces the effective robot usage rate. This suggests that, for open pit mine site, designs there is a clear benefit to a road network that maintains multiple options to get between two points where possible. Realistically, this should be feasible at both the pit and crusher/unloading areas.

Across the range of possible configurations of the open pit mine version of the dynamic multi-robot autonomous routing problem, mixed mode autonomy is still consistently the most efficient controller for the system. Improvements in the design of the mine site, communication system, and task configuration can make a 2-4% improvement in the effective robot usage rate, even with the bottleneck (haul road) included in the system.

Automated Container Terminal

The automated container terminal was represented as a moderately sparse environment with a bottleneck, one tightly coupled task and one moderately coupled task with long range communication and low overhead. Over the range of the number of robots simulated, there was little difference in the effective robot usage rate between mixed mode autonomy and centralized control for the automated container terminal. With low overhead all of the controllers are well under the available bandwidth limit at 500 robots, however, mixed mode autonomy is more likely to scale well as the MRS grows even larger due to the lower bandwidth requirements. The variation of the effective robot usage rates are lower for centralized control that the other controllers. The selection of a controller for the automated container would depend on which

was more important to the specific installation – bandwidth usage or consistency in performance for all robots across the MRS.

The simulation results show how trends in performance can be used to refine the system design for the automated container terminal. The more tightly coupled the tasks for the automated container terminal, the lower the effective robot usage rate. These effects are most noticeable with the higher number of robots in the MRS. For the automated container terminal, spreading out the storage locations throughout the container yard would reduce the coupling of one of the tasks. This could improve the effective robot usage rate by as much as 1 to 2% at 500 robots. The task coupling on the loading berth side is harder to change since it is set by the number of ships trying to load or unload at any given time.

The communication settings have a marginal impact on the effective robot usage rate for mixed mode autonomy and distributed control with communication for the numbers of robots in MRS for the simulations. This means that a wide range of hardware and communication protocols could be used to get the kind of results predicted by the simulations. Centralized control was only useful for the long range communication – the effective robot usage rate was lower than 50% in some cases with short range communication. Increasing the overhead dropped the effective usage rate significantly. The nominal case for the automated container terminal was for the low overhead case. The results suggest that the overhead for the system would need to stay low for centralized control to remain a viable option to mixed mode autonomy. This means that the update messages could not contain much other than the routing information need for the dynamic multi-robot autonomous routing problem. In other words, the system could not add much about maintenance, local environment conditions, etc. and stay competitive with mixed mode autonomy.

The influence of the environmental parameters for the automated container terminal was similar to the open pit mine. Bottlenecks and sparser connection patterns decreased the effective robot usage rate. For the automated container terminal, increasing the connectivity in either the loading berth or the container yard would yield a small improvement in efficiency. Reducing the bottleneck at the interchange area would have a larger impact and may be feasible in a number of container terminals.

Mixed mode autonomy and centralized control are both viable for automate container terminals under the conditions simulated. They are a small improvement over the distributed control with communication. Based on the trends, mixed mode autonomy would be more appropriate if the MRS were to grow much beyond the 500 robots that was the highest number of robots simulated. Some modifications to the environment and task configuration could improve the effective robot usage rate by as much as 3% to 4%.

Automated Warehouse

The automated warehouse was represented as a sparsely connected environment with no bottleneck, a tightly coupled task and a uniformly distributed task with short range communication and low overhead. The automated warehouse was the one example application with short range communications. The use of short range communication eliminated centralized

control as an option since the effective robot usage rates were so low when the communication range was decreased. There were too many packet errors between the centralized controller and the robots. Due to the dependency on successful transmission and reception of messages to work, centralized control will not work well when communication is not reliable. To fully evaluate the automated warehouse problem with all the controllers, the long range case was also examined.

For short range communication in the automated warehouse environment, mixed mode autonomy was much more effective. In the long range communication case, centralized control was still lagged mixed mode autonomy in performance. Since the performance of mixed mode autonomy was the same with short or long range communication, it does not make sense to use centralized control since it will require more power for the long range communication and there will not be any performance benefit.

The results from the simulations can be used to improve the design of automated warehouses. Like for the open pit mine and the container terminal, the more tightly coupled the tasks the lower the effective robot usage rate. The effect, however, is not quite as strong. For the automated warehouse, performance will be reduced if the task coupling for the storage area does not remain uniform. It is unlikely that the task coupling of the shipping and sorting area can be loosened all that much due to restrictions on bringing things in and out of the warehouse.

The communication parameters have more effect on distributed control with communication and mixed mode autonomy for the automated warehouse than with the other two applications. For mixed mode autonomy there is a small decrease in effective robot usage rate that is constant across all numbers of robots for short range communication with high overhead. The performance of mixed mode autonomy with all of the other communication settings is basically the same. This means that the system performance will fall if a communication protocol with high overhead is chosen or the update messages include significantly more information than just the information required for routing.

The environmental parameters affect performance in much the same way for the automated warehouse as the other two applications. Adding a bottleneck or reducing the connectivity of the environment reduces the effective robot usage rate. System performance would fall if a bottleneck were added between the shipping and sorting area and the storage area. This can be avoided for most warehouse layouts. There would be some benefit to the effective robot usage rate from increasing the number of lanes across the rows (increased connections in environment), but this is not practical for most warehouse since this significant reduces the effective storage area in the warehouse.

Mixed mode autonomy performs much better for the automated warehouse than distributed control with communication and beats the performance of centralized control even with shorter range communication. The performance with the parameters used for the automated warehouse is nearly the best that can reasonably expected with the practical constraints of warehouse design.

6.2 System Design Recommendations

Dynamic multi-robot autonomous routing could be used as part of many applications other than the three discussed in this work. Overall there were 144 different combinations of simulations parameters used to simulate the controllers to solve the dynamic multi-robot autonomous routing problem. This section outlines the overall performance trends for design of systems and selections of controllers for the MRS. The trends for the four groups of parameters (control type, environment, task coupling, and communication) are presented.

Controller Trends

The five controllers for the dynamic multi-robot autonomous routing problem simulated for this work were distributed control without communication, distributed control with communication, centralized control, mixed mode autonomy and mixed mode autonomy with model updates. Over the range of conditions simulated there was not a statistically significant difference between mixed mode autonomy and mixed mode autonomy with model updates.

Distributed control without communication was the lower baseline case for the dynamic multi-robot autonomous routing problem. In every case, distributed control without communication had the lowest effective robot usage rate for 300 robots or more. With a lower number of robots it was, at best, equivalent to the worst performing controller. This shows that the information provided to the other controllers successfully improved the system performance.

Mixed mode autonomy was the top performing controller for almost all of the combinations of parameters. It had high effective robot usage rates, low bandwidth requirements, a low rate of decrease of the effective robot usage rate as the MRS increases in size, and low sensitivity to communication overhead or range changes. Due to the prioritization of the messages from individual robot and the distributed nature for the base of the controller, the total number of messages sent by the robots and central controller is much lower than for any of the other controllers with communication. This results in much lower bandwidth usage for mixed mode autonomy.

When the communication overhead is low, centralized control can compete with mixed mode autonomy over the range of the numbers of robots simulated. The effective robot usage rates almost the same or just slightly lower for centralized control in these cases. Centralized control has the benefit of a lower standard deviation in the effective robot usage rate. This means that the robot performance is more consistent across the MRS and that the work load is spread more evenly for centralized control.

Distributed control with communication does not match the performance of mixed mode autonomy in any of the cases except for the sparsest environment with a bottleneck. The performance in this environment is the poorest for all of the controllers. Some optimization of the controller settings might eliminate this benefit. Distributed control with communication also has the highest standard deviation of effective robot usage rate of the controllers other than distributed control without communication. Mixed mode autonomy outperforms distributed control with communication because it has the same controller as its base, but adds the oversight of the central controller.

Environment Trends

The environmental parameters have a strong effect on the effective robot usage rates as the number of robots in the MRS increases. Any bottleneck (constrictions between the two task centers) in the environment leads to significant drop in the effective robot usage rate. This effect is especially large as the number of robots increases. Essentially, the bottleneck creates a point where all of the robots in the environment must pass which increases the congestion in the MRS. Some of this congestion can be managed by the controllers, but there are limits to the improvement. The performance drop caused by a bottleneck is increased as the environments get less connected. For the toughest environment, a 1 X 4 connection pattern with a bottleneck, the performance difference with the next hardest environment can be 4% to 10%. The drop with this environment is worst with centralized control. Overall, reducing the connection level drops the system performance, but the effect is much smaller when there is no bottleneck.

For a given application, it is beneficial to eliminate restrictions that lead to bottlenecks and to keep the environment as connected as possible. Basically, a system design should try to give robots as many options for routes as possible. If the tasks are more spread through the environment this becomes less important for the system design since the loss in performance will be lower.

Task Coupling Trends

The biggest difference in tasks comes between uniformly distributed tasks and tasks that are tightly or moderately coupled. The effective robot usage rates are much higher when at least one of the tasks is uniformly distributed. Making the coupling of one of the tasks from moderate instead of tight causes a small drop in the effective robot usage rate. More tightly coupled tasks create opportunities for more congestion at the task center, because more robots will need to in that area. With a low number of robots, the times that the robots will need to be at the task center can be staggered to avoid congestion. This is essentially what happens after the first few tasks as the robots pause to maintain the separation distance. This is why there is not much variation in performance for task coupling with lower number of robots. As the number of robots gets higher, it is no long possible to effectively space out the robots.

When possible, the tasks for dynamic multi-robot autonomous routing should be as decoupled as possible. This reduces the probability of congestion within the environment. This is especially important for environments that are less connected or have a bottleneck. In some applications, the desire to spread out the tasks will be balanced with the desire to keep the average task distance short. For example if the most shipped items are kept close to shipping and sorting area in the automated warehouse, it will decrease the average task length, but make the tasks tightly coupled. For any combination of parameters, there will be a point at which the total number of tasks the system can complete can be increased by making the tasks longer since that could help reduce some of the congestion.

Communication Trends

The bandwidth usages for each of the controllers were fairly steady across the communication parameters that were simulated. Centralized control with short range communication was the exception due to high packet error rates which led to a high rate of retransmission of commands. Ultimately, the extremely low effective robot usage rates for centralized control with short range communication kept this from being a viable controller configuration.

Communication parameters matter far more for centralized control than any of the other controllers. High communication overhead drops the effective robot usage rate. This due to the combination of high bandwidth usage (a high number of messages per robot with long messages) and a strict dependence on robot receiving commands before they can proceed to the next step on their route. When the network is saturated, it introduces latency to sending the commands which causes robots to idle until they receive the message.

Mixed mode autonomy and distributed control with communication offer more flexibility in the configuration of communication systems. Centralized control will require a low overhead, long range system to work appropriately. All of the controllers clearly benefit from communication when compared to distributed control without communication.

6.3 Future Work

This section suggests ways the best ways to continue this work that build off of mixed mode autonomy, the MRS control architecture developed in this work and the simulation models for the example applications. Future work on mixed mode autonomy can be divided into five areas – other communication scenarios, controller optimization, effects of uncertainty, improved environments and parameters for example applications, and integration with task allocation algorithms.

Other Communication Scenarios

The communication models used for the simulations were realistic, but represent a small portion of the possible communication scenarios for the dynamic multi-robot autonomous routing problem. All of the wireless communication nodes for the simulations had a clear line-of-sight, were homogenous, and could only transmit one at a time. Each of these assumptions simplified the number of iterations and parameters sets needed to simulate the controllers, but it could be useful to test the controllers in situations without the assumptions. Adding communication obstructions to block some of the line-of-sight would be useful to test mixed mode autonomy with model updates. This version of mixed mode autonomy sends peer status messages to update the world models of the central controller or other robots when they are not getting the information they need directly. The communication simulations did not have enough situations where a robot or the central controller did not regularly get a normal status update message from the nearby robots. A communication obstruction would be added to the environments used for the simulations. An example of what this kind of environment would be is shown in Figure 6-1.

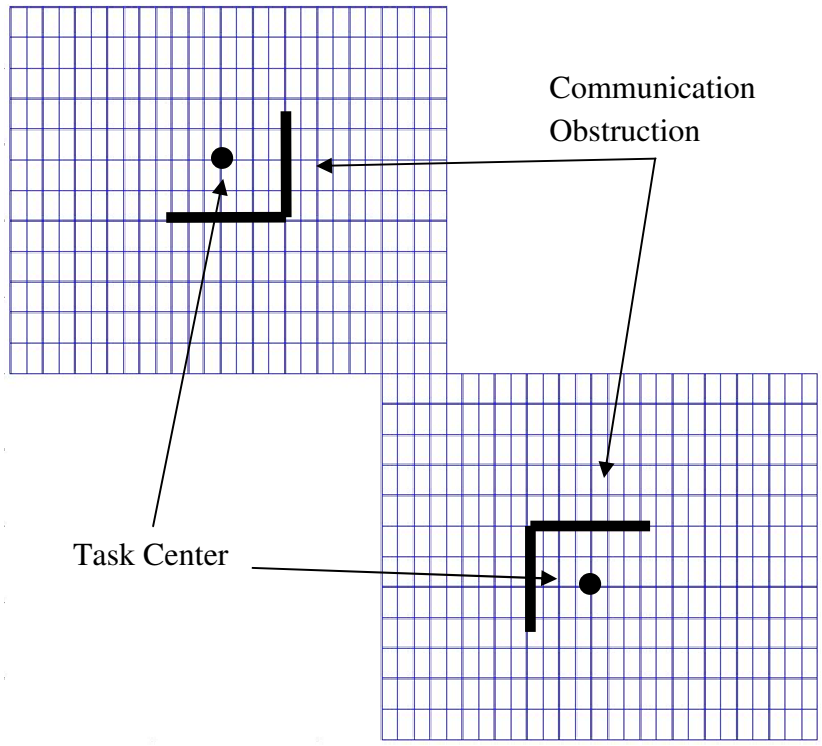


Figure 6-1. Alternative Environment for Restricted Communication

Another likely scenario for many applications of dynamic multi-robot routing would be heterogeneous communication settings. Some of the examples of this include different communication hardware on different robots or the centralized controller, variability in the performance of the hardware (manufacturing tolerances, etc.), and damaged hardware. For mixed mode autonomy with model updates, the damaged communication hardware case is particularly interesting. One kind of damage would be to the antenna, which could significantly reduce the communication range. This would test the ability of mixed mode autonomy with model updates to keep neighboring robots' and the centralized controller's world models current.

The last of the three communication scenarios is to allow multiple robots to transmit at the same time. This would only apply for the short range cases, since the long range communication covers virtually the entire environment. The environment could be divided into cells in a mesh network. This change increases the chances of communication failure, but increases the available bandwidth by a factor equal to the number of communication cells in the environment.

Controller Optimization

The controller settings were iterated during development to until things worked well. None of the settings were optimized for MRS performance. The controller settings include the number of steps ahead considered for the route selection, the penalties assigned for probable congestion, the separation distance between robots, the controller time intervals, and the update time intervals. The same settings were used for each of the controllers. In addition to varying the application parameters, such as grid connection pattern, bottleneck, task coupling, communication overhead,

and communication range, the controller settings could be changed during the simulations as well. Initially, the simulations would be used to more fully understand the effect of the settings on the MRS performance (effective robot usage rate, bandwidth usage, etc.). A second set of simulations could then be used to determine settings that were close to optimal.

The current controllers look two steps ahead during route planning (with the exception of distributed control without communication that only uses sensor information). This was a balance between controller performance and computational costs of the controller (and thus simulations). With the current level of uncertainty (2% sensor false positive rate), the controller could predict with reasonable accuracy the possible locations of all of the robots in their world models as much as three or four steps ahead before the predictions become worthless. As the uncertainty increases, it becomes harder to make these predictions. As the number of steps ahead considered for route planning increase, the penalty assigned for congestion in the later steps would need to be reduced to account for the decreased chance that congestion will actually occur.

During controller development, it was observed that the penalties for potential congestion had a significant impact on how the systems performed. In particular when the values were too high, the robots took routes that were considerably sub-optimal. For the most connected environments, the longer routes actually helped improve performance compared to the delays involved with handling congestion. The problem was with less connected environments – the longer routes usually took longer than waiting for the congestion to clear. When the penalties were too low, the systems performed like the distributed controller without communication.

The separation distance is partially a controller setting and partially an application parameter. It is the minimum distance required between robots. Physically, it would relate to the stopping distance of the robot and the sensor ranges. From the controller standpoint, it also sets how many robots can be on the same road segment in the network and an upper limit (theoretical and practical) on the number of robots an MRS can have for an environment or application.

The controller update intervals determine how frequently the robot sends messages to the centralized controller and/or its peers in the MRS. Currently the update interval is set so that there are ten communication periods during the time a robot would travel a road segment. Robots using distributed control with communication and centralized control send an update each communication period. Robots with mixed mode autonomy used information about distance to neighboring robots and location in the environment to determine how frequently to update their status. Ten communication periods per road segment was a tradeoff between bandwidth usage and keeping the world models current. With anything much lower, the world models were not accurate enough to plan routes very well. With higher update rates, the available bandwidth was not sufficient for decentralized control with communication or centralized control. Ten times per road segment is not the optimal update rate – it, however, is a good starting point to determine the best rate. The optimal rate will also depend on the assumed level for uncertainty of robot execution of their routes.

Effects of Uncertainty

Uncertainty was introduced to the system through the false positives for the obstacle detection sensor. One of the areas that would be interesting to explore would be to examine the effects of increasing the uncertainty in the execution of the tasks. For the standard cases that were presented in Chapter 5, the false positive rate was 2%. Several sets were run for the standard parameters with a 5% false positive rate and a proportionally higher velocity (so robots would travel the same average velocity for the standard case and the higher false positive case). Figure 6-2 shows the effective robot usage rates for mixed mode autonomy and centralized control for the open pit mine problem with 2% and 5% false positive rates. The effective robot usage rates declined at faster with the higher false positive rate, especially for centralized control. Not enough iterations were run to show the effects with any statistical significance, but they suggest that the reduction in performance was lower for mixed mode autonomy than the other controllers. Centralized control particularly seemed to have a difficult time with the increased uncertainty. More iterations would need to be run for additional values of false positive rate to reach firm conclusions, but the preliminary results suggest that mixed mode autonomy could be particularly advantageous as the uncertainty in robot execution increases. Trials or the standard parameters should be run for false positive rates ranging from one to fifteen percent.

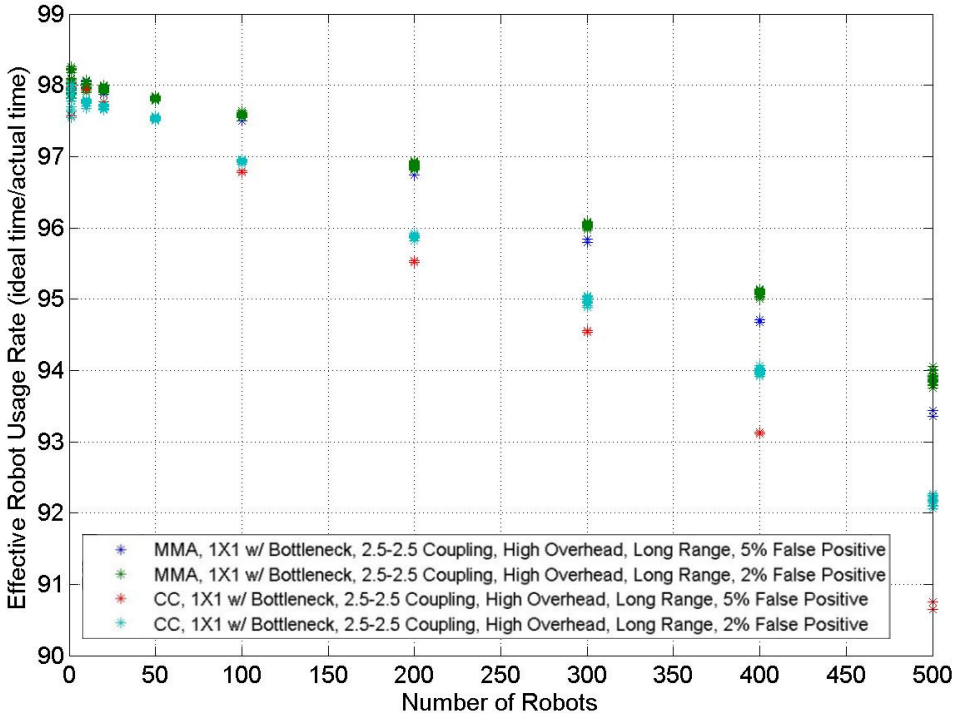


Figure 6-2. Effective Robot Usage Rates for Mixed Mode Autonomy and Centralized Control for the Open Pit Mine Problem and Sensor False Positive Rates of 2% and 5%

Improved Environments and Parameters for Example Applications

The simulations were run for generic environments, tasks, and communications that could represent a wide range of dynamic multi-robot autonomous routing problems. The combinations approximate most of the conditions of interest for likely applications and identified the performance trends for the parameters, but were not set for any specific application. A possible next step would be to design simulations specific to real instances of the example applications. One parameter that could be explored for all of the example applications is the robot velocity. This would affect the relationship update rates and bandwidth usage. Specific communication hardware and protocols could be selected rather than abstracting some of that detail to capture the general trends.

For the open pit mine, application specific parameters would mean changing the environment to more accurately reflect the configuration of the haul road(s) and differences in the grid connection patterns between the pit and crusher/loading area. A more advanced simulation might also include elevation changes between the pit and crusher/loading area. This would capture the real environment well, but would be computationally intensive to simulate well. The task coupling and communication parameters could be set to approximate a real open pit mine.

The automated container terminal parameter that least accurately represents the real application is the task coupling at the loading berth. The distribution would be wide along the ship, but almost zero perpendicular to it. There would also be multiple loading berths, instead of one single task center. The ratio of the size of the container yard to the size of the loading berths should be higher than one. For many container terminals the ratio is closer to three or four.

The automated warehouse parameters are the closest match to the real application. The environment and task coupling were reasonable approximations to typical warehouses. The primary difference that could be tested is the presence of communication obstructions due to the rows of shelves in the environment. This would particularly be interesting with mixed mode autonomy with model updates.

Integration with Task Allocation Algorithms

One of the central assumptions of the simulations was online assignment for all of the dynamic multi-robot autonomous routing tasks. This is sub-optimal, but simplified the simulations and helps emphasize how coordinated control of the MRS could improve the execution of tasks. Much of the previous research in coordinated control of MRS is on optimal task allocation. One of the next steps for mixed mode autonomy is to combine it with an optimal task assignment algorithm (or with several different algorithms). The overall effective robot usage rates should be higher with optimal allocation so the effects of the execution part of control will be more subtle.

6.4 Contributions

This section summarizes the unique contributions of this dissertation. There were three primary areas of contribution to the state of art – development of a new MRS control architecture, development of a system model for the dynamic multi-robot autonomous routing problem, and

identification of the tradeoffs for MRS design for the dynamic multi-robot autonomous routing problem.

MRS Control Architecture

Mixed mode autonomy is a novel MRS control architecture that reduces the communication requirements, improves the performance in many situations, and scales better for large numbers of robots in an MRS than the traditional controllers for the dynamic multi-robot autonomous routing problem. It was developed to trade off the benefits of centralized and distributed control of MRS. Unlike previous MRS control architectures, mixed mode autonomy is a distributed controller with a centralized controller. The robots can operate fully independently, but the centralized controller can override any robot decision if it finds a better solution.

The communication reduction is achieved by using the world models kept by individual robots to determine the usefulness of sending their status updates. The robots increase their status update rate when they are either far from the centralized controller or when they have neighboring robots. The rate is increased when robots are far from the centralized controller to ensure it is kept current because packet errors get more likely at longer distances. Robots are defined as neighbors when their information might impact the route selection for a robot.

The increase in the effective robot usage rate for mixed mode autonomy can be attributed to the independence from the centralized controller, the oversight from the centralized controller, and the reduced communication requirements. Unlike with centralized control the robots are not dependent on receiving commands from the centralized controller to proceed with the next step in a task. This reduces the idle time for the robots caused by failed command messages and latency in sending messages. Distributed control with communication also has this independence, but occasionally makes route decisions without information the centralized control would have. This offsets the gains made by pure independence from the centralized controller. With higher numbers of robots the gains in effective robot usage rate are because the communication network is not saturated as easily as centralized control or distributed control with communication.

The reduced communication requirements and independence from the centralized control improve the scalability of the system for mixed mode autonomy. The communication requirements are lower for both the robots and the central controller. The robots send messages at a lower rate most of the time. The highest update rate for mixed mode autonomy is the standard rate for the centralized control and distributed control with communication. The centralized controller only sends commands when that command is an improvement over the robot's plan. The lower number of messages means more robots can be in the MRS before the network saturates. The independence from the centralized controller means that even if the network were to saturate the effective robot usage rate would drop more slowly than for the other controllers.

Systems Model for Dynamic Multi-Robot Autonomous Routing

A model for dynamic multi-robot autonomous routing was developed. The biggest improvement over prior models was to include a realist simulation of wireless communication between the

robots and centralized controller. Most of the simulation work on MRS control makes optimistic or unrealistic assumptions about communication. Much of the work assumes ideal communication – none of the messages sent by either the robots or centralized controllers ever fail. This is unlikely to be the case in any real application, and has major consequences for the performance of any control architecture that depends on communication to share information throughout the MRS. The solution was to include a detailed model of wireless communication at the core of the system.

The communication model was based off of typical operating environments and protocols for the example applications. Ultimately, the packet error rate (PER) was calculated for every transmitter-receiver pair. The PER calculations included the path loss effects, reflection, scattering, the encoding method, and the length of the packet. The only simplifying assumptions were that the communication conditions were constant throughout the environment and that there were no elevation changes. This meant that simulations accurate reflect how the controllers would work in a real communication environment.

The remainder of the simulation model was fairly similar to most of the others used in this kind of research. The focus of the simulation was the high level functionality of the MRS, not the detailed operation of individual robots. The robots were homogenous even though none of the controllers required this – it reduced the number of iterations of the Monte Carlo simulation to converge. The road network is reduced to a uniform grid to reduce the time for the simulations to run. The robots were treated as a point.

Tradeoff Analysis for MRS Design with Dynamic Multi-Robot Autonomous Routing

Several clear trends were identified from the simulation results regarding control architecture selection and the effect of system parameters on the MRS performance for the dynamic multi-robot autonomous routing problem. These trends are useful for designing the configurations of the MRS, environment, and communication system for an application. Some careful planning based on these tradeoff relationships can improve the effective robot usage rate by a few percent. This is a large difference, especially for large scale MRS. The tradeoffs were discussed in detail in section 6.2. The primary trends were: mixed mode autonomy used less bandwidth and had higher effective robot usage rates; bottlenecks in the environment reduced the effective robot usage rate; less connected environments has lower effective robot usage rates; increasing the coupling of tasks reduced the effective robot usage rate; and effective robot usage rate dropped dramatically when the bandwidth limitations are reached.

References

[1] Brian Gerkey and Maja Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, September 2004, http://ai.stanford.edu/~gerkey/research/final_papers/mrta-taxonomy.pdf.

[2] Lynne Parker, "Multiple Mobile Robot Systems," in *Springer Handbook of Robotics*.: Springer, 2008, p. Chapter 40.

[3] K. Pruhs and B. Kalyanasundaram, "Online Weighted Matching," *Journal of Algorithms*, 1993.

[4] Alejandro R. Mosteo, Luis Montano, and Michail G. Lagoudakis, "Multi-Robot Routing under Limited Communication Range," in *IEEE International Conference on Robotics and Automation 2008*, 2008, http://www.mosteo.com/papers/mml08-icra.pdf.

[5] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, and Pinar Keskinocak, "Auction-Based Multi-Robot Routing," in *Robotics Science and Systems*, 2005, http://www.cs.ucla.edu/~awm/papers/robotauction.pdf.

[6] David Grossman, "Traffic Control of Multiple Robot Vehicles," *IEEE Journal of Robotics and Automation*, 1988, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=20433&isnumber=816.

[7] B.L. Brumitt and A. Stentz, "Dynamic mission planning for multiple mobile robots," in *International Conference on Robotics and Automation*, 1996, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=506522.

[8] Kiva Systems. Kiva Systems.

[9] Paul Kellet. (2011) Robotic Industries Association. [Online]. http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Feature-Article/Robotic-Warehousing:-A-New-Market-Opportunity-for-Robot-Manufacturers-and-Integrators/content_id/2574

[10] Lee, Nwana, Ndumu, and De Wilde, "The Stability, Scalabilty and Performance of Multi-Agents," *BT Technology Journal*, no. 16, 1998, http://citeseer.ist.psu.edu/cache/papers/cs/17902/http:zSzzSzwww.labs.bt.comzSzprojectszSzagentszSzpublishzSzpaperszSzbttj98-scalability.pdf/lee98stability.pdf.

[11] Xianbo Xiang, Xinhan Huang, Qin Zhang, and Guohua Xu, "Prospective Research on Coordinated Control of Multiple AUVS," in *Control and Decision Conference 2008*, China,

2008.

[12] EURON Special Interest Group on Cooperative Robotics, "Two "Hot Issues" in Cooperative Robotics: Network Robot Systems, and Formal Models and Methods for Cooperation," 2008.

[13] Farinell, Iocchi, and Nardi, "Multirobot Systems: A Classification Focused on Coordination," *IEEE Transactions on Systems, Man, and Cybernetics*, October 2004, http://ieeexplore.ieee.org/iel5/3477/29465/01335496.pdf?tp=&isnumber=&arnumber=1335496.

[14] Maja Mataric and Brian Gerkey, "A Framework for Studying Multi-Robot Task Allocation (MRTA)," *Multi-Robot Systems: From Swarms to Intelligent Automata*, 2003, http://robotics.stanford.edu/~gerkey/research/final_papers/mrs03.pdf.

[15] Matson Gustafson, "Taxonomy of Cooperative Robotic Systems," , 2003, http://ieeexplore.ieee.org/iel5/8811/27880/01244565.pdf?tp=&isnumber=&arnumber=1244565.

[16] Gancet and Lacroix, "Embedding Heterogeneous Levels of Decisional Autonomy in Multi-Robot Systems," in *Distributed Autonomous Robots and Systems (DARS)*, 2004, http://www.comets-uavs.org/papers/GANCET-DARS-2004.pdf.

[17] Gancet, Hattenberger, Alami, and Lacroix, "Task planning and control for a multi-UAV system: architecture and algorithms," in *IEEE IROS*, 2005, http://www.comets-uavs.org/papers/GANCET-IROS-2005.pdf.

[18] Rosenfeld, Kaminka, and Kraus, "Adaptive Robotic Communication Using Coordination Costs," in *Distributed Autonomous Robots and Systems (DARS)*, 2006, http://www.cs.biu.ac.il/~maverick/Publications/Papers/dars06avi.pdf.

[19] Rosenfeld, Kaminka, and Kraus, "A Study of Scalability Properties in Robotic Teams," in *Coordination of Large-Scale Multiagent Systems.*, 2005, http://www.cs.biu.ac.il/~galk/Publications/Papers/rosenfeld05.pdf.

[20] Maja Mataric and Brian Gerkey, "Sold!: Auction Methods for Multirobot Coordination," *IEEE Transactions on Robotics and Automation*, October 2002, http://ieeexplore.ieee.org/iel5/70/22928/01067996.pdf?tp=&arnumber=1067996&isnumber=22928.

[21] Maja Mataric, EH Ostergaard, and Guarav Sukhatme, "Multi-robot Task Allocation in the Light of Uncertainty," in *IEEE International Conference on Robotics and Automation*,

2002,
http://ieeexplore.ieee.org/iel5/7916/21827/01013688.pdf?tp=&arnumber=1013688&isnumber=21827.

[22] P Sujit, A Sinha, and D Ghose, "Multiple UAV task allocation using negotiation," in *AAMAS '06 Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, New York, 2006.

[23] Botelho Alami, "Robots that cooperatively enhance their plans," in *Distributed Autonomous Robots and Systems (DARS)*, 2000,
http://mack.ittc.ku.edu/cache/papers/cs/23304/http:zSzzSzwww.laas.frzSz~silviacbzSznew_dars.pdf/botelho00robots.pdf.

[24] Isik, Stulp, Mayer, and Utz, "Coordination without Negotiation in Teams of Heterogeneous Robots," in *Robocup 2006 Proceedings*, 2006,
http://www9.cs.tum.edu/papers/pdf/isik06coordination.pdf.

[25] Maayan Roth, Douglas Vail, and Manuela Veloso, "A World Model for Multi-Robot Teams with Communication," in *IEEE IROS*, 2003,
http://www.cs.cmu.edu/~robosoccer/cmrobobits/papers/03iros-model.pdf.

[26] Scerri, Vincent, and Mailler, "Comparing Three Approaches to Large-Scale Coordination," in *Coordination of Large-Scale Multiagent Systems*., 2005,
http://www.cs.cmu.edu/~pscerri/papers/SMVBook05.pdf.

[27] Steven Shafer and Douglas A. Reece, "A Computational Model of Driving for Autonomous Vehicles," Carnegie Mellon, 1991.

[28] Australian Government Department of Defense. (2010) MAGIC 2010. [Online].
http://www.dsto.defence.gov.au/MAGIC2010/

[29] M. Pasquier, H. C. Quek, B. T. Tan, and C. K. Chan, "Opportunistic Planning for a Fleet of Transportation Robots," in *Intelligent Transportation Systems*, 1999,
http://ieeexplore.ieee.org/iel5/6644/17743/00821141.pdf?arnumber=821141.

[30] Sanem Sariel-Talay, Tucker R. Balch, and and Nadia Erdogan, "Multiple Traveling Robot Problem: A Solution Based on Dynamic Task Selection and Robust Execution," *IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 14, NO. 2, APRIL 2009*, vol. 14, no. 2, April 2009.

[31] Andrew Lim and Fan Wang, "Multi-Depot Vehicle Routing Problem: A One-Stage Approach," *IEEE Transactions on Automation Science adn Engineering*, October 2005,

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1514459.

[32] D.K. Liu, X. Wu, A. K. Kulatunga, and G. Dissanayake, "Motion Coordination of Multiple Autonomous Vehicles in Dynamic and Strictly Constrained Environments," in *IEEE Cybernetics and Intelligent Systems*, 2006, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4017901&isnumber=4017783.

[33] A. K. Kulatunga, D. K. Liu, G. Dissanayake, and S.B. Siyambalapitiya, "Ant Colony Optimization based Simultaneous Task Allocation and Path Planning of Autonomous Vehicles," , 2006, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04017908.

[34] Clark, Rock, and Latombe, "Motion Planning for Multiple Mobile Robot Systems using Dynamic Networks," in *International Conference on Robotics and Automation*, 2003, http://ai.stanford.edu/~latombe/papers/icra03/multi/paper.pdf.

[35] Ivan Mezei, Veljko Malbasa, and Ivan Stojmenovic, *AD-HOC, MOBILE AND WIRELESS NETWORKS - Auction Aggregation Protocols for Wireless Robot-Robot Coordination*.: Springer, 2009, vol. 5793.

[36] Maja Mataric, Vaughan, Stoy, and G. Sukhatme, "Exploiting Task Regularities to Transform Between Reference Frames in Robot Teams," in *International Conference on Robotics and Automation*, 2002, http://ieeexplore.ieee.org/iel5/7916/21827/01013623.pdf?arnumber=1013623.

[37] Beslon, Biennier, and Hirsbrunner, "Multi-Robot Path-Planning Based on Implicit Cooperation in a Robotic Swarm," in *International conference on autonomous agents*, 1998, http://delivery.acm.org/10.1145/290000/280772/p39-beslon.pdf?key1=280772&key2=5220245021&coll=GUIDE&dl=ACM&CFID=59174424&CFTOKEN=56162734.

[38] R Walters and B Jennings, "Swarm-Based MRS Peformance Limits," in *International Joint Conference on Artificial Intelligence*, 2009.

[39] A. Ekic, P. Keskinocak, and S. Koenig, "Multi-robot routing with linear decreasing rewards over time," in *2009. ICRA '09. IEEE International Conference on Robotics and Automation*, 2009.

[40] Alami, "A fleet of autonomous and cooperative mobile robots," in *IROS*, 1996.

[41] Min, Zhe, Yin, Hiang, and Yong, "A Rules and Communication Based Multiple Robots Transportation System," in *Computational Intelligence in Robotics and Automation*, 1999,

http://ieeexplore.ieee.org/iel5/6589/17587/00810046.pdf?arnumber=810046.

[42] Nishi, Mori, Konishi, and Imai, "An Asynchronous Distributed Routing System for Multi-robot Cooperative Transportation," in *IROS*, 2005, http://ieeexplore.ieee.org/iel5/10375/32977/01545268.pdf?arnumber=1545268.

[43] S Shafiee, M Nehring, and E Topal, "Estimating Average Total Cost of Open Pit Coal Mines in Australia," in *Australian Mining Technology Conference*, 2009.

[44] Q Gu, C Lu, F Li, and C Wan, "Monitoring dispatch information system of trucks and shovels in an open pit based on GIS/GPS/GPRS," *Journal of China University of Mining and Technology*, June 2008.

[45] A Nieto and K Dagdelen, "Development and Testing of a Vehicle Collision Avoidance System Based on GPS and Wireless Networks for Open-Pit Mines," 2010.

[46] Qiang Wang, Ying Zhang, Chong Chen, and Wenli Xu, "Open-Pit Mine Truck Real-time Dispatching Principle under Macroscopic Control," in *Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on* , Beijing, 2006.

[47] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-Robot Cooperation in the MARTHA Project," *Robotics and Automation*, 1998, http://ieeexplore.ieee.org/iel1/100/14631/00667325.pdf?arnumber=667325.

[48] J. Zhang, P. Ioannou, and A. Chassiakos, "Automated container transport system between inland port and terminals," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, April 2006, http://portal.acm.org/citation.cfm?id=1138464.1138465.

[49] S. Hoshino and J. Ota, "Design of an automated transportation system in a seaport container terminal for the reliability of operating robots," in *IROS*, 2007, http://www.ieeexplore.ieee.org/iel5/4398943/4398944/04398972.pdf?tp=&isnumber=4398944&arnumber=4398972.

[50] P Baker and Z Halim, "An exploration of warehouse automation implementations: cost, service and flexibility issues," *Supply Chain Management: An International Journal*, 1999.

[51] J.I.U. Rubrico, J. Ota, T. Higashi, and H. Tamura, "Scheduling multiple agents for picking products in a warehouse," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006.

[52] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, pp. 269-271, 1959.

[53] D Gleich, Dijkstra Implementation in Matlab, 2009.

[54] Alexander Stapanov. Wikipedia. [Online].
http://en.wikipedia.org/wiki/File:Udachnaya_pipe.JPG

[55] Missy Schmidt. Wikipedia. [Online].
http://en.wikipedia.org/wiki/File:Norfolk_International_Terminal.jpg

[56] Green Logistics. (2011) Wikipedia. [Online].
http://en.wikipedia.org/wiki/File:Warehouse_md17.jpg

Appendix: Matlab Code for the Simulations

Distributed Control without Communication

```
tic
% Main simulation parameters

num_missions = 25;
totaltime = 0;
% load an environment

% Set of new environments

env_index = 101;
env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=4xupper=28yupper=25.mat';
DA = 50;
AP_location = [26 26];

% env_index = 102;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 103;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 104;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 105;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 106;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 201;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
```

```
%
% env_index = 202;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=75xupper=75yupper=2.mat';
% orig_cent = topnode;
% dest_cent = bottomnode;
% DA = 75;
% AP_location = [38 38];
%
% env_index = 203;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 204;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 205;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 206;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 301;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 302;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 303;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 304;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
```

```
% env_index = 305;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 306;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
load(env);
orig_cent = topnode;
dest_cent = bottomnode;

% Controller Information control_type is a flag set to record which
% controller is in use for the simulation
% type 1 is the baseline - fully decentralized, stigmergic approach
% type 2 is decentralized control with communication (DCC)
% type 3 is centralized control
% type 4 is ???
% type 5 is sliding autonomy/ETA Control
control_type = 1;
sim_length = 2000; %DA/vel * num_missions;
vel = DA*num_missions/sim_length;
scale = DA/100;

% sim_length = sim_length / 100;

% Sensor parameters
fp = 0.02; % Fraction of sensor readings that are false positive
fn = 0.02; % Fraction of sensor readings that are false negative

for N = [500 200 100 50 20 10 1] %1000
    low_t = tril(true(N),-1);
    pop = ones(N, 1, 'single');
    pop(1:N*fp) = 0;
    pop(N-N*fn+1:N) = 100;
    for thresh = [0.4 0.1] * scale
        %        for overhead = [48 240] % will remove for no comm cases
        % Communication Parameters - none, this is nocomm case
        % Simulation Timing
            [dt dt_type] = min([thresh/vel/2.4 1/vel/10]);
        delta = vel * dt;

        for coupling = [10000 10000 10000 10 10 5; 10000 10 5 10 5 5;] *
scale
            couple_orig = coupling(1);
            couple_dest = coupling(2); % task coupling - possible values of
10,000, 10, 5
            % Distribution about the center of origination
            scaled_cdf_orig = assign_matrix(orig_cent, couple_orig, xtotal,
ytotal, connected);

            % Distribution about the center of origination
```

```
            scaled_cdf_dest = assign_matrix(dest_cent, couple_dest, xtotal,
ytotal, connected);

            for iter = 1:1
                clearvars -except fp fn pop scale num_missions totaltime DA
sim_length iter N thresh vel couple_orig couple_dest WAM2 bottle connected d
dest_cent orig_cent nodes env_index env x_node xlower xtotal xupper y_node
ylower ytotal yupper low_t scaled_cdf_orig scaled_cdf_dest control_type
data_rate overhead total_length message_time AP_location tstep update_time dt
dt_type delta;

                % Simulation Variables
                tie = rand(N, 1);
                dist = zeros(N, N);
                dist_list = zeros(1, N^2/2-N/2);

                rand_fp_fn = zeros(N, N, 'single');
                seed_index = zeros(N, 1);
                sense_dist = dist;

                nearmiss = zeros(N, 1, 'int32');
                false_obstacle = zeros(N, 1, 'int32');
                bias = randn(N, N) * 0.1;
                sensitivity = randn(N, N)*.01 + 1;

                for j = 1:N
                    rand_fp_fn(j, :) = pop(ceil(N * rand(N,1)));
                end

                % Simulation Data (results logging)
                distance_traveled = zeros(1, N);
                paused = zeros(1, N, 'int32');
                nopause = zeros(1, N, 'int32');
                missions = ones(1, N, 'int32');
                % Assume maximum missions is 100 (for now)
                mission_dist = zeros(200, N, 'int32');
                mission_list = zeros(200, N, 'int32');
                mission_time = zeros(200, N);
                orig = zeros(1, N,'int32');
                dest = zeros(1, N, 'int32');
                next_vertex = zeros(1, N, 'int32');

                for n=1:N
                    raw_assign = rand(1);
                    orig(n) = find(scaled_cdf_orig > raw_assign, 1);

                    raw_assign = rand(1);
                    dest(n) = find(scaled_cdf_dest > raw_assign, 1);

                    while dest(n) == orig(n)
                        raw_assign = rand(1);
                        dest(n) = find(scaled_cdf_dest > raw_assign, 1);
                    end
```

```
                    candidates = find(WAM2(:, orig(n)))'; % faster to search
by column!
                    dlist = d(dest(n), candidates);
                    pathlength = min(dlist);
                    onpath = candidates(dlist == pathlength);
                    next_vertex(n) = onpath(ceil(rand * length(onpath)));
                end

                mission_start = orig;
                mission_dist(1, :) = abs(x_node(orig) - x_node(dest)) +
abs(y_node(orig) - y_node(dest));
                mission_list(1, :) = dest;

                dx = delta * (x_node(next_vertex) - x_node(orig));
                dy = delta * (y_node(next_vertex) - y_node(orig));

                x = x_node(orig);
                y = y_node(orig);

                path_travel = zeros(N, 1);

                for time = 0:dt:sim_length

                    % must include file in Path of C:\Program
                    dist_list = pdist([x y], 'euclidean');
                    dist = dist * 0;
                    dist(low_t) = dist_list;
                    dist = dist + dist';

                    sense_dist = dist .* sensitivity + bias;

                    % State update (in loop)
                    for n = 1:N
                        % Check if robot n is currently at a vertex
                        if abs(x_node(next_vertex(n)) - x(n)) +
abs(y_node(next_vertex(n)) - y(n)) <= delta/2
                            path_travel(n) = path_travel(n) + 1;
                            % Robot n has reached a vertex, check if vertex
the
                            % final goal for a mission
                            if next_vertex(n) == dest(n)
                                % End of mission reached, assign new mission,
                                % increment mission counter
                                orig(n) = dest(n);
                                if mod(missions(n),2) == 0
                                    while dest(n) == orig(n)
                                        raw_assign = rand(1);
                                        dest(n) = find(scaled_cdf_dest >
raw_assign, 1);
                                    end
                                else
                                    while dest(n) == orig(n)
                                        raw_assign = rand(1);
                                        dest(n) = find(scaled_cdf_orig >
raw_assign, 1);
```

```
                                    end
                                end
                                mission_time(missions(n), n) = time;
                                missions(n) = missions(n) + 1;
                                mission_dist(missions(n), n) =
abs(x_node(orig(n)) - x_node(dest(n))) + abs(y_node(orig(n)) -
y_node(dest(n)));
                                mission_list(missions(n), n) = dest(n);
                            end
                            candidates = find(WAM2(:, next_vertex(n)))'; %
faster to search by column!
                            dlist = d(dest(n), candidates);
                            pathlength = min(dlist);
                            onpath = candidates(dlist == pathlength);
                            % Set position to current next_vertex since we're
there
                            x(n) = x_node(next_vertex(n));
                            y(n) = y_node(next_vertex(n));
                            % set next step on path to dest
                            next_vertex(n) = onpath(ceil(rand *
length(onpath)));
                            % Set movement towards dest
                            dx(n) = delta * (x_node(next_vertex(n)) - x(n));
                            dy(n) = delta * (y_node(next_vertex(n)) - y(n));
                        end

                        % Check for conflicts
                        %conflicts = find(dist(:,n)<=thresh); % this is the
distance that should be replaced with noisy sensor data!
                        conflicts = find(sense_dist(:,n)<=thresh); % this is
the distance that should be replaced with noisy sensor data!
                        num_conflicts = length(conflicts);
                        stp = 1;
                        sensor_fp = rand(1);
                        if sensor_fp <= fp
                            false_obstacle(n) = false_obstacle(n) + 1;
                            stp = 0;
                        elseif num_conflicts > 1
                            sensor_fn = rand(1);
                            n_goal = next_vertex(n);
                            if sensor_fn <= fn
                                nearmiss(n) = nearmiss(n) + 1;
                            else
                                for k = conflicts'
                                    dist_n_to_next_pt = abs(x(n) -
x_node(n_goal)) + abs(y(n)- y_node(n_goal));
                                    if ( ( next_vertex(k) == n_goal ) && ( k
~= n ) )
                                        % check to see if k is closer to goal
than n
                                        dist_k_to_next_pt = abs(x(k) -
x_node(n_goal)) + abs(y(k)-y_node(n_goal));
                                        if ( dist_n_to_next_pt >=
dist_k_to_next_pt )
                                            % check for tie
                                            if ( dist_n_to_next_pt ==
dist_k_to_next_pt && tie(n) < tie(k) )
```

```
                                        else
                                            stp = 0;
                                            break
                                        end
                                    end
                                end
                            end
                        end
                        % Update x and y based on dx, dy, and conflict state
                        if stp == 1
                            x(n) = x(n) + dx(n);
                            y(n) = y(n) + dy(n);
                            distance_traveled(n) = distance_traveled(n) +
delta;
                            nopause(n) = nopause(n) + 1;
                        else
                            paused(n) = paused(n) + 1;
                        end
                    end
                end
                nam_var = ['control=' num2str(control_type) 'N=' num2str(N)
'env=' num2str(env_index) 'thresh=' num2str(thresh) 'len='
num2str(sim_length) 'couple_orig=' num2str(couple_orig) 'couple_dest='
num2str(couple_dest) 'iter=' num2str(iter)];
                looptime = toc - totaltime
                totaltime = toc;
                mission_time = mission_time - [zeros(1, N);
mission_time(1:199, :)];
                mission_time = max(mission_time, 0);
                save(['C:\multirobot model\Logs\sim-' nam_var '.mat'],
'bias', 'sensitivity', 'fp', 'fn', 'DA', 'sim_length', 'nearmiss',
'false_obstacle', 'num_missions', 'looptime', 'AP_location', 'N',
'candidates', 'conflicts', 'control_type', 'couple_dest', 'couple_orig',
'delta', 'dest', 'dest_cent', 'dist', 'distance_traveled', 'dlist', 'dt',
'dt_type', 'dx', 'dy', 'env', 'env_index', 'iter', 'missions', 'next_vertex',
'nopause', 'num_conflicts', 'onpath', 'orig', 'orig_cent', 'path_travel',
'pathlength', 'paused', 'raw_assign', 'scaled_cdf_dest', 'scaled_cdf_orig',
'sim_length', 'thresh', 'tie', 'time', 'vel', 'x', 'y', 'mission_dist',
'mission_list', 'mission_start', 'mission_time');
            end
        end
    end
end
toc
```

```
tic
% Main simulation parameters

num_missions = 25;
totaltime = 0;
% load an environment
% Set of new environments

env_index = 5555;
env = 'C:\multirobot
model\Environments\env_x=7y=7bottle=7xupper=7yupper=6.mat';
DA = 8;
topnode = 9;
bottomnode = 41;
AP_location = [4 4];

% env_index = 101;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];

% env_index = 102;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 103;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 104;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 105;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 106;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 201;
```

```
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 202;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 203;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 204;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 205;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 206;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 301;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 302;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 303;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 304;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
```

```
% AP_location = [51 51];
%
% env_index = 305;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 306;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];

load(env);
orig_cent = topnode;
dest_cent = bottomnode;

% Controller Information control_type is a flag set to record which
% controller is in use for the simulation
% type 1 is the baseline - fully decentralized, stigmergic approach
% type 2 is decentralized control with communication (DCC)
% type 3 is centralized control
% type 4 is ???
% type 5 is sliding autonomy/ETA Control
control_type = 2;
sim_length = 2000; %DA/vel * num_missions;
vel = DA*num_missions/sim_length;
scale = DA/100;

sim_length = sim_length/100; % Run shorter simulation to test stuff

% Sensor parameters
fp = 0.02; % Fraction of sensor readings that are false positive
fn = 0.02; % Fraction of sensor readings that are false negative

% Communication Model for 2.4 GHz
F = 6; % 6 dB
k = 1.38e-23; % J/K
T0 = 290; % K
B = 22e6; % Hz
Pn = 10 * log10 ((F + 1) * k * T0 * B * 1000); % dBm
% assume unity gains on antennas for now
PL_d0 = 40;  % dB
d0 = scale; % m
loss_exp = 3.0;
dev = 9; % dB  typical value for log-normal shadowing with open terrain
Bn = 22e6; % Hz
R = 1e6; % bps
Bn_R = Bn/R; % Bn/R

for N = 20 %[500 200 100 50 20 10 1]
    low_t = tril(true(N+1),-1);
    AP = N + 1;
    for thresh = 0.4 * scale %[0.4 0.1] * scale
```

```
        for comm_set = [240; -7.75] %[48 48 240 240; -1.25 -7.75 -0.5 -7]%
replaced overhead with comm parameters (overhead and Pt)
            overhead = comm_set(1);
            Pt = comm_set(2);
            % Communication Parameters
            data_rate = 1e6; % bits per second
            data_length = 48; % bits
            total_length = data_length + overhead;
            FL = total_length;
            message_time = total_length / data_rate; % seconds
            update_time = message_time * 1000;

            % Simulation Timing
            [dt dt_type] = min([thresh/vel/2.4 1/vel/10]);
            delta = vel * dt;

            for coupling = [12.5; 12.5;] * scale %[10000 10000 10000 12.5
12.5 6.25; 10000 12.5 6.25 12.5 6.25 6.25;] * scale
                couple_orig = coupling(1);
                couple_dest = coupling(2); % task coupling - possible values
of 10,000, 10, 5
                % Distribution about the center of origination
                scaled_cdf_orig = assign_matrix(orig_cent, couple_orig,
xtotal, ytotal, connected);

                % Distribution about the center of origination
                scaled_cdf_dest = assign_matrix(dest_cent, couple_dest,
xtotal, ytotal, connected);

                for iter = 1:1
                    clearvars -except comm_set Pt AP data_rate data_length
total_length message_time Pt F k T0 B Pn PL_d0 d0 loss_exp dev Bn_R FL fp fn
scale num_missions totaltime DA sim_length iter N thresh vel couple_orig
couple_dest WAM2 bottle connected d dest_cent orig_cent nodes env_index env
x_node xlower xtotal xupper y_node ylower ytotal yupper low_t scaled_cdf_orig
scaled_cdf_dest control_type data_rate overhead total_length message_time
AP_location tstep update_time dt dt_type delta;

                    % Simulation Variables
                    tdynamic = 0;
                    tupdate = 0;
                    tie = rand(N, 1);
                    dist = zeros(N+1, N+1);
                    %                     dist_list = zeros(1, N^2/2-N/2);

                    sense_dist = dist;
                    message_queue = zeros(N+1, 4); % Pt, x, y, vx, vy, row
corresponds to transmitter/robot
                    message_queue(:, 1) = 10; % Queue starts empty
                    central_model = zeros(N+1, 4); % Time updated,
robot_status
                    local_model = zeros(N+1, 4, N+1);

                    nearmiss = zeros(1, N, 'int32');
                    false_obstacle = zeros(1, N, 'int32');
```

```
bias = randn(N+1, N+1) * 0.1;
sensitivity = randn(N+1, N+1)*.01 + 1;

% Simulation Data (results logging)
msg_sent = zeros(N+1, N+1, 'int32');
bandwidth_used = 0;
messages_failed = zeros(N+1, N+1, 'int32');
missed = 0;

distance_traveled = zeros(1, N);
paused = zeros(1, N, 'int32');
pause_nocomm = zeros(1, N, 'int32');
nopause = zeros(1, N, 'int32');
missions = ones(1, N, 'int32');
mission_dist = zeros(200, N, 'int32');
mission_list = zeros(200, N, 'int32');
mission_time = zeros(200, N);
orig = zeros(1, N,'int32');
dest = zeros(1, N, 'int32');
next_vertex = zeros(1, N, 'int32');
sent_vertex = zeros(1, N, 'int32');
next_vertex_buffer = zeros(1, N, 'int32');
cmd_buffer = zeros(2, N, 'int32');
cmd_buffer_send = 1;
cmd_buffer_store = 1;
cmd_flag = zeros(1, N, 'int8');
TX_time = zeros(N, 1);
RX_time = zeros(N, 1);

% Initial task assignments
for n=1:N
    raw_assign = rand(1);
    orig(n) = find(scaled_cdf_orig > raw_assign, 1);

    raw_assign = rand(1);
    dest(n) = find(scaled_cdf_dest > raw_assign, 1);

    while dest(n) == orig(n)
        raw_assign = rand(1);
        dest(n) = find(scaled_cdf_dest > raw_assign, 1);
    end

    candidates = find(WAM2(:, orig(n)))'; % faster to
search by column!
    dlist = d(dest(n), candidates);
    pathlength = min(dlist);
    onpath = candidates(dlist == pathlength);
    next_vertex(n) = onpath(ceil(rand * length(onpath)));
end

sent_vertex = next_vertex;
next_vertex_buffer = next_vertex;
prev_vertex = orig;

mission_start = orig;
```

```
                    mission_dist(1, :) = abs(x_node(orig) - x_node(dest)) +
abs(y_node(orig) - y_node(dest));
                    mission_list(1, :) = dest;

                    dx = delta * (x_node(next_vertex) - x_node(orig));
                    dy = delta * (y_node(next_vertex) - y_node(orig));

                    x = [x_node(orig); AP_location(1)];
                    y = [y_node(orig); AP_location(2)];

                    dist_list = pdist([x y], 'euclidean');
                    dist = dist * 0;
                    dist(low_t) = dist_list;
                    dist = dist + dist';

                    % Initital PRR calculation for all robots and AP

                    PL = PL_d0 + 10 * loss_exp *
log2(dist/d0)/3.3219280948873622 + randn(N+1, N+1) * dev;
                    SNR = Pt - PL - Pn;
                    Pe = 0.5*erfc(sqrt(10.^(SNR/10)*Bn_R));
                    PRR = (1 - Pe).^FL;

                    PRR_AP = PRR(AP, 1:N);

                    path_travel = zeros(N, 1);

                    for time = 0:message_time:sim_length

                        % load cmd_buffer to message_queue

                        if (message_queue(AP, 1) == 10 && cmd_buffer_send ~=
cmd_buffer_store)
                            %                               test_cond = [time
single(cmd_buffer(1, cmd_buffer_send)) x(cmd_buffer(1, cmd_buffer_send))
y(cmd_buffer(1, cmd_buffer_send)) single(x_node(next_vertex(cmd_buffer(1,
cmd_buffer_send)))) single(y_node(next_vertex(cmd_buffer(1,
cmd_buffer_send)))) single(cmd_buffer(2, cmd_buffer_send))/100]
                            message_queue(AP, 1) = rand(1) * 0.1; % Give
central control messages higher priority on network
                            message_queue(AP, 2) = cmd_buffer(1,
cmd_buffer_send);
                            message_queue(AP, 3) = cmd_buffer(2,
cmd_buffer_send);
                            if cmd_buffer_send == N
                                cmd_buffer_send = 1;
                            else
                                cmd_buffer_send = cmd_buffer_send + 1;
                            end
                        end

                        % Message communication simulation
                        [val message_index] = min(message_queue(:,1));
                        % calculate the packet reception rate for the link
between the robot
```

```
                        % (message_index) and the access point
                        if ( val < 10)
                            bandwidth_used = bandwidth_used + total_length;
                            % Update local model and track communication
success/failure

                            condit = PRR(:, message_index) > rand(N+1, 1);
                            local_model(message_index, 1, condit) = time;
                            local_model(message_index, 2, condit) =
message_queue(message_index, 2);
                            local_model(message_index, 3, condit) =
message_queue(message_index, 3);
                            local_model(message_index, 4, condit) =
message_queue(message_index, 4);
                            % How should message success and failure be
tracked for multipoint
                            % communiction?
                            msg_sent(:, message_index) = msg_sent(:,
message_index) + int32(condit); % track percentage of bandwidth used
successfully
                            messages_failed(:, message_index) =
messages_failed(:, message_index) + int32(not(condit));
                            message_queue(message_index, 1) = 10;
                        end

                        if time >= tdynamic-message_time/2
                            tdynamic = tdynamic + dt; %time + dt;

                            dist_list = pdist([x y], 'euclidean'); % version
if not
                            %modified
                            dist = dist * 0;
                            dist(low_t) = dist_list;
                            dist = dist + dist';

                            % Communication propagation model

                            PL = PL_d0 + 10 * loss_exp *
log2(dist/d0)/3.3219280948873622 + randn(N+1, N+1) * dev;
                            SNR = Pt - PL - Pn;
                            Pe = 0.5*erfc(sqrt(10.^(SNR/10)*Bn_R));
                            PRR = (1 - Pe).^FL;

                            PRR_AP = PRR(AP, 1:N);

                            sense_dist = dist .* sensitivity + bias;

                            % State update (in loop)
                            for n = 1:N
                                stp = 1;
                                % Check if robot n is currently at a vertex
                                if abs(x_node(next_vertex(n)) - x(n)) +
abs(y_node(next_vertex(n)) - y(n)) <= delta/2
                                    path_travel(n) = path_travel(n) + 1;
```

```
                                % Robot n has reached a vertex, check if
vertex the
                                % final goal for a mission
                                if next_vertex(n) == dest(n)
                                    % End of mission reached, assign new
mission,
                                    % increment mission counter
                                    orig(n) = dest(n);
                                    if mod(missions(n),2) == 0
                                        while dest(n) == orig(n)
                                            raw_assign = rand(1);
                                            dest(n) =
find(scaled_cdf_dest > raw_assign, 1);
                                        end
                                    else
                                        while dest(n) == orig(n)
                                            raw_assign = rand(1);
                                            dest(n) =
find(scaled_cdf_orig > raw_assign, 1);
                                        end
                                    end
                                    mission_time(missions(n), n) = time;
                                    missions(n) = missions(n) + 1;
                                    mission_dist(missions(n), n) =
abs(x_node(orig(n)) - x_node(dest(n))) + abs(y_node(orig(n)) -
y_node(dest(n)));
                                    mission_list(missions(n), n) =
dest(n);
                                end
                                candidates = find(WAM2(:,
next_vertex(n)))'; % faster to search by column!
                                %                               dlist =
zeros(size(candidates));
                                %                               idx = 1;
                                %                               for i =
candidates
                                %
dlist(idx) = d(i, dest(n));
                                %                                   idx =
idx + 1;
                                %                               end
                                dlist = d(dest(n), candidates);
                                pathlength = min(dlist);
                                onpath = candidates(dlist == pathlength);
                                % Set position to current next_vertex
since we're there
                                x(n) = x_node(next_vertex(n));
                                y(n) = y_node(next_vertex(n));
                                % set next step on path to dest
                                prev_vertex(n) = next_vertex(n);
                                next_vertex(n) = onpath(ceil(rand *
length(onpath)));
                                % Set movement towards dest
                                dx(n) = delta * (x_node(next_vertex(n)) -
x(n));
                                dy(n) = delta * (y_node(next_vertex(n)) -
y(n));
```

```
                                end
                                % Check for conflicts
                                %conflicts = find(dist(:,n)<=thresh); % this
is the distance that should be replaced with noisy sensor data!
                                conflicts = find(sense_dist(1:N,n)<=thresh);
% this is the distance that should be replaced with noisy sensor data!
                                num_conflicts = length(conflicts);
                                sensor_fp = rand(1);
                                if sensor_fp <= fp
                                    false_obstacle(n) = false_obstacle(n) +
1;
                                    stp = 0;
                                elseif num_conflicts > 1
                                    sensor_fn = rand(1);
                                    n_goal = next_vertex(n);
                                    if sensor_fn <= fn
                                        nearmiss(n) = nearmiss(n) + 1;
                                    else
                                        for q = conflicts'
                                            dist_n_to_next_pt = abs(x(n) -
x_node(n_goal)) + abs(y(n)- y_node(n_goal));
                                            if ( ( next_vertex(q) == n_goal )
&& ( q ~= n ) )
                                                % check to see if q is closer
to goal than n
                                                dist_q_to_next_pt = abs(x(q)
- x_node(n_goal)) + abs(y(q)-y_node(n_goal));
                                                if ( dist_n_to_next_pt >=
dist_q_to_next_pt )
                                                    % check for tie
                                                    if ( dist_n_to_next_pt ==
dist_q_to_next_pt && tie(n) < tie(q) )
                                                    else
                                                        stp = 0;
                                                        break
                                                    end
                                                end
                                            end
                                        end
                                    end
                                end
                                % Update x and y based on dx, dy, and
conflict state
                                if stp == 1
                                    x(n) = x(n) + dx(n);
                                    y(n) = y(n) + dy(n);
                                    distance_traveled(n) =
distance_traveled(n) + delta;
                                    nopause(n) = nopause(n) + 1;
                                else
                                    paused(n) = paused(n) + 1;
                                end
                            end
                        end
                        if time >= tupdate-message_time/2
                            tupdate = tupdate + update_time;
                            missed = missed + sum(message_queue(:,1) ~= 10);
```

```
                            dist_to_next_vertex = abs(x_node(next_vertex) -
x(1:N)) + abs(y_node(next_vertex) - y(1:N));
                            message_queue(1:N, 1) = rand(N, 1);
                            message_queue(1:N, 2) = dist_to_next_vertex;
                            message_queue(1:N, 3) = prev_vertex';
                            message_queue(1:N, 4) = next_vertex';
                        end
                    end
                    nam_var = ['control=' num2str(control_type) 'N='
num2str(N) 'env=' num2str(env_index) 'thresh=' num2str(thresh) 'len='
num2str(sim_length) 'couple_orig=' num2str(couple_orig) 'couple_dest='
num2str(couple_dest) 'overhead=' num2str(overhead) 'Pt=' num2str(Pt) 'iter='
num2str(iter)];
                    looptime = toc - totaltime
                    totaltime = toc;
                    mission_time = mission_time - [zeros(1, N);
mission_time(1:199, :)];
                    mission_time = max(mission_time, 0);
                    save(['C:\multirobot model\Logs\sim-' nam_var '.mat'],
'overhead', 'Pt', 'message_time', 'missed', 'msg_sent_AP', 'msg_sent_robot',
'messages_failed_AP', 'messages_failed_robot', 'bias', 'sensitivity', 'fp',
'fn', 'DA', 'sim_length', 'nearmiss', 'false_obstacle', 'num_missions',
'looptime', 'AP_location', 'N', 'candidates', 'conflicts', 'control_type',
'couple_dest', 'couple_orig', 'delta', 'dest', 'dest_cent', 'dist',
'distance_traveled', 'dlist', 'dt', 'dt_type', 'dx', 'dy', 'env',
'env_index', 'iter', 'missions', 'next_vertex', 'nopause', 'num_conflicts',
'onpath', 'orig', 'orig_cent', 'path_travel', 'pathlength', 'paused',
'pause_nocomm', 'raw_assign', 'scaled_cdf_dest', 'scaled_cdf_orig',
'sim_length', 'thresh', 'vel', 'x', 'y', 'mission_dist', 'mission_list',
'mission_start', 'mission_time');
                end
            end
        end
    end
end
toc
```

Centralized Control

```
tic
% Main simulation parameters

% Changed 6-6-10 to do all communication on a per dt basis instead of by
increments of message time

num_missions = 25;
totaltime = 0;
% load an environment

% Set of new environments

env_index = 101;
env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=4xupper=28yupper=25.mat';
DA = 50;
AP_location = [26 26];

% env_index = 102;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 103;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 104;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 105;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 106;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 201;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
```

```
%
% env_index = 202;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 203;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 204;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 205;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 206;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 301;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 302;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 303;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 304;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 305;
```

```
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 306;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];

load(env);
orig_cent = topnode;
dest_cent = bottomnode;

% Controller Information control_type is a flag set to record which
% controller is in use for the simulation
% type 1 is the baseline - fully decentralized, stigmergic approach
% type 2 is decentralized control with communication (DCC)
% type 3 is centralized control
% type 4 is ???
% type 5 is sliding autonomy/ETA Control
control_type = 3;
sim_length = 2000; %DA/vel * num_missions;
vel = DA*num_missions/sim_length;
scale = DA/100;

sim_length = sim_length/50; % comment this line out for full simulation

% Sensor parameters
fp = 0.02; % Fraction of sensor readings that are false positive
fn = 0.02; % Fraction of sensor readings that are false negative

% Communication Model for 2.4 GHz
F = 6; % 6 dB
k = 1.38e-23; % J/K
T0 = 290; % K
B = 22e6; % Hz
Pn = 10 * log10 ((F + 1) * k * T0 * B * 1000); % dBm
% assume unity gains on antennas for now
PL_d0 = 40;  % dB
d0 = scale; % m
loss_exp = 3.0;
dev = 9; % dB  typical value for log-normal shadowing with open terrain
Bn = 22e6; % Hz
R = 1e6; % bps
Bn_R = Bn/R; % Bn/R

for N = 500 %[500 400 300 200 100 50 20 10 1]
    low_t = tril(true(N+1),-1);
    AP = N + 1;
    thresh = 0.4 * scale; %[0.4 0.1] * scale
    for comm_set = [48; 3.7] %[48 240; 3.7 5]% replaced overhead with comm
parameters (overhead and Pt)
        overhead = comm_set(1);
        Pt = comm_set(2);
```

```
        % Communication Parameters
        data_rate = 1e6; % bits per second
        data_length = 48; % bits
        total_length = data_length + overhead;
        FL = total_length;
        message_time = total_length / data_rate; % seconds

        % Simulation Timing
        [dt dt_type] = min([thresh/vel/2.4 1/vel/10]);
        delta = vel * dt;
        update_time = dt;

        for coupling = [10; 10] * scale %[10000 10000 10000 10 10 5; 10000 10
5 10 5 5;] * scale
            couple_orig = coupling(1);
            couple_dest = coupling(2); % task coupling - possible values of
10,000, 10, 5
                        % Distribution about the center of origination
                        scaled_cdf_orig = assign_matrix(orig_cent,
couple_orig, xtotal, ytotal, connected);

                        % Distribution about the center of origination
                        scaled_cdf_dest = assign_matrix(dest_cent,
couple_dest, xtotal, ytotal, connected);
            % Load distributions to prevent license problem with statistics
toolkit
%
%             if couple_orig == 10000 * scale
%                 scaled_cdf_orig = scaled_cdf_orig1;
%             elseif couple_orig == 10 * scale
%                 scaled_cdf_orig = scaled_cdf_orig2;
%             else
%                 scaled_cdf_orig = scaled_cdf_orig3;
%             end
%
%             if couple_dest == 10000 * scale
%                 scaled_cdf_dest = scaled_cdf_dest1;
%             elseif couple_dest == 10 * scale
%                 scaled_cdf_dest = scaled_cdf_dest2;
%             else
%                 scaled_cdf_dest = scaled_cdf_dest3;
%             end

            for iter = 1:1
                clearvars -except scaled_cdf_orig1 scaled_cdf_orig2
scaled_cdf_orig3 scaled_cdf_dest1 scaled_cdf_dest2 scaled_cdf_dest3 reach
comm_set Pt AP data_rate data_length total_length message_time Pt F k T0 B Pn
PL_d0 d0 loss_exp dev Bn_R FL fp fn scale num_missions totaltime DA
sim_length iter N thresh vel couple_orig couple_dest WAM2 bottle connected d
dest_cent orig_cent nodes env_index env x_node xlower xtotal xupper y_node
ylower ytotal yupper low_t scaled_cdf_orig scaled_cdf_dest control_type
data_rate overhead total_length message_time AP_location tstep update_time dt
dt_type delta;

                % Simulation Variables
                tdynamic = 0;
```

```
                tupdate = 0;
                tie = rand(N, 1);
                dist = zeros(N+1, N+1);

                sense_dist = dist;
                message_queue = zeros(N+1, 4); % Pt, x, y, vx, vy, row
corresponds to transmitter/robot
                message_queue(:, 1) = 10; % Queue starts empty
                central_model = zeros(N, 4); % Time updated, robot_status

                nearmiss = zeros(1, N, 'int32');
                false_obstacle = zeros(1, N, 'int32');
                bias = randn(N+1, N+1) * 0.1;
                sensitivity = randn(N+1, N+1)*.01 + 1;

                % Simulation Data (results logging)
                msg_sent_AP = 0;
                msg_sent_robot = zeros(1, N, 'int32');
                bandwidth_used = 0;
                messages_failed_AP = 0;
                messages_failed_robot = zeros(1, N, 'int32');
                missed = 0;

                distance_traveled = zeros(1, N);
                paused = zeros(1, N, 'int32');
                pause_nocomm = zeros(1, N, 'int32');
                nopause = zeros(1, N, 'int32');
                missions = ones(1, N, 'int32');
                mission_dist = zeros(200, N, 'int32');
                mission_list = zeros(200, N, 'int32');
                mission_time = zeros(200, N);
                orig = zeros(1, N,'int32');
                dest = zeros(1, N, 'int32');
                next_vertex = zeros(1, N, 'int32');
                cmd_buffer = zeros(2, N, 'int32');
                cmd_buffer_send = 1;
                cmd_buffer_store = 1;
                cmd_flag = zeros(1, N, 'int8');
                %                     TX_time = zeros(N, 1);
                %                     RX_time = zeros(N, 1);

                sent_time = zeros(N, 1);
                sent_route_flag = zeros(N, 1);
                n_list = (1:1:N)';

                central_model(:, 2) = 1; % all robots start full segment away
from next vertex

                % Initial task assignments
                for n=1:N
                    raw_assign = rand(1);
                    orig(n) = find(scaled_cdf_orig > raw_assign, 1);

                    raw_assign = rand(1);
                    dest(n) = find(scaled_cdf_dest > raw_assign, 1);
```

```
                    while dest(n) == orig(n)
                        raw_assign = rand(1);
                        dest(n) = find(scaled_cdf_dest > raw_assign, 1);
                    end

                    candidates = find(WAM2(:, orig(n)))'; % faster to search
by column!
                    dlist = d(dest(n), candidates);
                    pathlength = min(dlist);
                    onpath = candidates(dlist == pathlength);
                    next_vertex(n) = onpath(ceil(rand * length(onpath)));
                end

                cmd_vertex = next_vertex; % initialize command set as first
set of next vertices
                next_vertex_buffer = next_vertex;
                prev_vertex = orig;

                mission_start = orig;
                mission_dist(1, :) = abs(x_node(orig) - x_node(dest)) +
abs(y_node(orig) - y_node(dest));
                mission_list(1, :) = dest;

                dx = delta * (x_node(next_vertex) - x_node(orig));
                dy = delta * (y_node(next_vertex) - y_node(orig));

                x = [x_node(orig); AP_location(1)];
                y = [y_node(orig); AP_location(2)];

                dist_list = pdist([x y], 'euclidean');
                dist = dist * 0;
                dist(low_t) = dist_list;
                dist = dist + dist';

                PL_AP = PL_d0 + 10 * loss_exp * log2(dist(AP,
1:N)/d0)/3.3219280948873622 + randn(1, N)*dev;
                SNR_AP = Pt - PL_AP - Pn;
                %Pe_AP = Q(sqrt(2*10.^(SNR_AP/10)*Bn_R));
                Pe_AP = 0.5*erfc(sqrt(10.^(SNR_AP/10)*Bn_R));
                PRR_AP = (1 - Pe_AP).^FL;

                path_travel = zeros(N, 1);

                for time = 0:dt:sim_length
                    % calculate list of robots that need to be routed

                    % robot needs route if it is with distance of next vertex
and the next vertex buffer is still the
                    % next vertex
                    needs_route = (central_model(:, 2) < 3.25 * delta &
central_model(:, 3) == central_model(:, 4)); % 0.1 value is placeholder
                    num_to_route = sum(needs_route);
                    mess_per_dt = min(floor(dt/ message_time)-num_to_route,
N);
```

```
                    % route the robots

                    % resend failed route messages for sent_route_flag &
needs_route & central model has been updated since command was sent
                    resend_route = needs_route & (central_model(:,1) >=
sent_time & sent_route_flag); % central model

                    % calculate new routes

                    needs_new_list = n_list(needs_route & ~resend_route)';
%n_list(needs_route)';
                    if (~isempty(needs_new_list))
                        for curr_rob = needs_new_list
                            if next_vertex(curr_rob) == dest(curr_rob)
                                % End of mission reached, assign new mission,
                                % increment mission counter
                                orig(curr_rob) = dest(curr_rob);
                                if mod(missions(curr_rob),2) == 0
                                    while dest(curr_rob) == orig(curr_rob)
                                        raw_assign = rand(1);
                                        dest(curr_rob) = find(scaled_cdf_dest
> raw_assign, 1);
                                    end
                                else
                                    while dest(curr_rob) == orig(curr_rob)
                                        raw_assign = rand(1);
                                        dest(curr_rob) = find(scaled_cdf_orig
> raw_assign, 1);
                                    end
                                end
                                mission_time(missions(curr_rob), curr_rob) =
time;
                                missions(curr_rob) = missions(curr_rob) + 1;
                                mission_dist(missions(curr_rob), curr_rob) =
abs(x_node(orig(curr_rob)) - x_node(dest(curr_rob))) +
abs(y_node(orig(curr_rob)) - y_node(dest(curr_rob)));
                                mission_list(missions(curr_rob), curr_rob) =
dest(curr_rob);
                            end
                            % replaced unique with part of that code
                            curr_vertices = sort(central_model([1:curr_rob-1
curr_rob+1:N],4));
                            if ~isempty(curr_vertices)
                                cv_db = diff(curr_vertices);
                                numelcv = N-1;
                                cv_ind = cv_db ~= 0;
                                cv_ind(numelcv) = 1;
                                curr_vertices = curr_vertices(cv_ind);
                            end

                            candidates = find(WAM2(:,
next_vertex(curr_rob))); % faster to search by column!
                            % form path sets
                            cp_step1 = candidates(candidates ~=
prev_vertex(curr_rob));
```

```
                            if sum(cp_step1 == dest(curr_rob)) == 1 % if any
of the first steps in possible paths is the destination - go there and no
need to check paths
                                %cmd_buffer(:, cmd_buffer_store) = [curr_rob;
dest(curr_rob);];
                                cmd_vertex(curr_rob) = dest(curr_rob);
                            else
                                path_set = zeros(2, length(cp_step1)*3);
                                path_index = 0;
                                for j = 1:size(cp_step1)
                                    poss_step = find(WAM2(:,cp_step1(j)));
                                    cp_step2 = poss_step(poss_step ~=
next_vertex(curr_rob));
                                    for g = 1:length(cp_step2)
                                        path_index = path_index + 1;
                                        path_set(1, path_index) =
cp_step1(j);
                                        path_set(2, path_index) =
cp_step2(g);
                                    end
                                end
                                path_set = path_set(:, 1:path_index);
                                dlist_raw = d(dest(curr_rob),
path_set(2,:))+2;
                                dlist = dlist_raw;
                                for p = 1:size(path_set, 2)
                                    %[step_on_path can_reach_path] =
find(d(path_set(:, p), :) <= 2);
                                    can_reach_path = [reach(path_set(1,
p)).can_reach_path'; reach(path_set(2, p)).can_reach_path'];
                                    %potential_conflicts =
intersect(central_model([1:curr_rob-1 curr_rob+1:N],4), can_reach_path); %
replace curr_rob (z only) with N+1 for centralized router
                                    %potential_conflicts =
curr_vertices(ismember(curr_vertices, can_reach_path));
                                    % this method is much faster for low N,
about the same for N=500
                                    cv_indexed = false(length(curr_vertices),
1);
                                    for crp = 1:length(can_reach_path)
                                        cv_indexed = (curr_vertices ==
can_reach_path(crp)) | cv_indexed;
                                    end
                                    potential_conflicts =
curr_vertices(cv_indexed);

                                    off_base = central_model(curr_rob, 2) +
1; % replace with own location information - distance to reach first point on
possible next path
                                    if isempty(potential_conflicts) ~= 1
                                        for u = potential_conflicts'
                                            check_cond = central_model(:, 4)
== u; % index to the robot where the conflict is
                                            cm = central_model(check_cond,
2);
                                            d1 = d(u, path_set(1,p));
```

```
                                                offset_dist1 = off_base - (d1 +
cm);
                                                offset_dist2 = off_base + 1 -
(d(u, path_set(2,p)) + cm);
                                                %penalty = (1 ./ (d(u,
path_set(1,p)) + d(path_set(1,p), dest(check_cond)) - d(u, dest(check_cond))
+ 1)) * max([thresh - abs(offset_dist1) thresh - abs(offset_dist2)
zeros(size(offset_dist1))], [], 2); % penalty is reduced if the robot is
unlikely to visit that node on its path
                                                dest_check = dest(check_cond);
                                                d2 = d(path_set(1,p),
dest_check);
                                                d3 = d(u, dest_check);
                                                %
pen1 = thresh - abs(offset_dist1);
                                                %
pen2 = thresh - abs(offset_dist2);
                                                %
penalty = (1 ./ (d1 + d2 - d3 + 1)) * max(max(pen1, pen2),0); % penalty is
reduced if the robot is unlikely to visit that node on its path
                                                penalty = (1 ./ (d1 + d2 - d3 +
1)) * max(thresh - min(abs(offset_dist1), abs(offset_dist2)),0); % penalty is
reduced if the robot is unlikely to visit that node on its path
                                                %penalty = (1 ./ (d(u,
path_set(1,p)) + d(path_set(1,p), dest(check_cond)) - d(u, dest(check_cond))
+ 1)) * max([thresh - abs(offset_dist1) thresh - abs(offset_dist2)
zeros(size(offset_dist1))], [], 2); % penalty is reduced if the robot is
unlikely to visit that node on its path

                                                dlist(p) = dlist(p) + penalty;
                                            end
                                        end
                                    end
                                    % path selection
                                    pathlength = min(dlist);
                                    onpath = path_set(1, dlist == pathlength);
                                    % set next step on path to dest
                                    %cmd_buffer(:, cmd_buffer_store) = [curr_rob;
onpath(randi(length(onpath)));];
                                    cmd_vertex(curr_rob) =
onpath(randi(length(onpath)));

                                end
                            end
                        end

                        rx_cmd = PRR_AP > rand(1, N) & needs_route';
                        msg_sent_AP = msg_sent_AP + sum(rx_cmd);
                        fail_cmd = needs_route' & ~rx_cmd;
                        messages_failed_AP = messages_failed_AP + sum(fail_cmd);
                        next_vertex_buffer(rx_cmd) = cmd_vertex(rx_cmd);
                        sent_time(needs_route) = time;
                        sent_route_flag = needs_route;

                        % determine robots that transmit during this dt
                        [sorted rows] = sortrows(message_queue);
```

```
                    last_in_queue = find(sorted(:,1) == 10, 1) - 1;
                    if last_in_queue > 0
                        valid_queue = rows(sorted(:,1)<10);
                        valid_queue = valid_queue(1:min(mess_per_dt,
last_in_queue));
                        send_ok = PRR_AP(valid_queue)>rand(1,
min(mess_per_dt, last_in_queue));
                        sent_queue = valid_queue(send_ok);
                        failed_queue = valid_queue(~send_ok);
                        message_queue(valid_queue, 1) = 10; % clear queue -
message was sent
                        messages_failed_robot(failed_queue) =
messages_failed_robot(failed_queue) + 1;
                        msg_sent_robot(sent_queue) =
msg_sent_robot(sent_queue) + 1;

                        % update central model
                        central_model(sent_queue, 1) = time;
                        central_model(sent_queue, 2:4) =
message_queue(sent_queue, 2:4);

                    end

                    dist_list = pdist([x y], 'euclidean'); % version if not
                    %modified
                    dist = dist * 0;
                    dist(low_t) = dist_list;
                    dist = dist + dist';

                    % Communication propagation model

                    PL_AP = PL_d0 + 10 * loss_exp * log2(dist(AP,
1:N)/d0)/3.3219280948873622 + randn(1, N)*dev;
                    SNR_AP = Pt - PL_AP - Pn;
                    %Pe_AP = Q(sqrt(2*10.^(SNR_AP/10)*Bn_R));
                    Pe_AP = 0.5*erfc(sqrt(10.^(SNR_AP/10)*Bn_R));
                    PRR_AP = (1 - Pe_AP).^FL;

                    sense_dist = dist .* sensitivity + bias;

                    % State update (in loop)
                    for n = 1:N
                        stp = 1;
                        % Check if robot n is currently at a vertex
                        if (abs(x_node(next_vertex(n)) - x(n)) +
abs(y_node(next_vertex(n)) - y(n))) <= delta/2
                            % Check if next command has been received
                            if next_vertex_buffer(n) ~= next_vertex(n) % was
-1
                                % Set position to current next_vertex since
we're there
                                x(n) = x_node(next_vertex(n));
                                y(n) = y_node(next_vertex(n));
                                % Increment segment count
                                path_travel(n) = path_travel(n) + 1;
```

```
                                % Set vertex
                                prev_vertex(n) = next_vertex(n);
                                next_vertex(n) = next_vertex_buffer(n);
                                %
next_vertex_buffer(n) = -1;
                                dx(n) = delta * (x_node(next_vertex(n)) -
x(n));
                                dy(n) = delta * (y_node(next_vertex(n)) -
y(n));
                            else
                                stp = -1;
                            end
                        end

                        % Check for conflicts
                        %conflicts = find(dist(:,n)<=thresh); % this is the
distance that should be replaced with noisy sensor data!
                        conflicts = find(sense_dist(1:N,n)<=thresh); % this
is the distance that should be replaced with noisy sensor data!
                        num_conflicts = length(conflicts);
                        sensor_fp = rand(1);
                        if sensor_fp <= fp
                            false_obstacle(n) = false_obstacle(n) + 1;
                            stp = 0;
                        elseif num_conflicts > 1
                            sensor_fn = rand(1);
                            n_goal = next_vertex(n);
                            if sensor_fn <= fn
                                nearmiss(n) = nearmiss(n) + 1;
                            else
                                for q = conflicts'
                                    dist_n_to_next_pt = abs(x(n) -
x_node(n_goal)) + abs(y(n)- y_node(n_goal));
                                    if ( ( next_vertex(q) == n_goal ) && ( q
~= n ) )
                                        % check to see if q is closer to goal
than n
                                        dist_q_to_next_pt = abs(x(q) -
x_node(n_goal)) + abs(y(q)-y_node(n_goal));
                                        if ( dist_n_to_next_pt >=
dist_q_to_next_pt )
                                            % check for tie
                                            if ( dist_n_to_next_pt ==
dist_q_to_next_pt && tie(n) < tie(q) )
                                            else
                                                stp = 0;
                                                break
                                            end
                                        end
                                    end
                                end
                            end
                        end
                        % Update x and y based on dx, dy, and conflict state
                        if stp == 1
                            x(n) = x(n) + dx(n);
                            y(n) = y(n) + dy(n);
```

```
                            distance_traveled(n) = distance_traveled(n) +
delta;
                            nopause(n) = nopause(n) + 1;
                        elseif stp == 0
                            paused(n) = paused(n) + 1;
                        else
                            pause_nocomm(n) = pause_nocomm(n) + 1;
                        end
                    end
                    %                     if time >= tupdate-message_time/2
                    %                         tupdate = tupdate +
update_time;
                    % update time is set to be the same as dt = 12 messages/
per segment for DA = 50
                    missed = missed + sum(message_queue(:,1) ~= 10);
                    dist_to_next_vertex = abs(x_node(next_vertex) - x(1:N)) +
abs(y_node(next_vertex) - y(1:N));
                    message_queue(1:N, 1) = rand(N, 1);
                    message_queue(1:N, 2) = dist_to_next_vertex;
                    message_queue(1:N, 3) = next_vertex_buffer';
                    message_queue(1:N, 4) = next_vertex';
                    %                     end
                end
                nam_var = ['control=' num2str(control_type) 'N=' num2str(N)
'env=' num2str(env_index) 'thresh=' num2str(thresh) 'len='
num2str(sim_length) 'couple_orig=' num2str(couple_orig) 'couple_dest='
num2str(couple_dest) 'overhead=' num2str(overhead) 'Pt=' num2str(Pt) 'iter='
num2str(iter)];
                looptime = toc - totaltime
                totaltime = toc;
                mission_time = mission_time - [zeros(1, N);
mission_time(1:199, :)];
                mission_time = max(mission_time, 0);
                save(['C:\multirobot model\Logs\sim-' nam_var '.mat'],
'data_length', 'bandwidth_used', 'overhead', 'Pt', 'message_time', 'missed',
'msg_sent_AP', 'msg_sent_robot', 'messages_failed_AP',
'messages_failed_robot', 'bias', 'sensitivity', 'fp', 'fn', 'DA',
'sim_length', 'nearmiss', 'false_obstacle', 'num_missions', 'looptime',
'AP_location', 'N', 'candidates', 'conflicts', 'control_type', 'couple_dest',
'couple_orig', 'delta', 'dest', 'dest_cent', 'dist', 'distance_traveled',
'dlist', 'dt', 'dt_type', 'dx', 'dy', 'env', 'env_index', 'iter', 'missions',
'next_vertex', 'nopause', 'num_conflicts', 'onpath', 'orig', 'orig_cent',
'path_travel', 'pathlength', 'paused', 'pause_nocomm', 'raw_assign',
'sim_length', 'thresh', 'vel', 'x', 'y', 'mission_dist', 'mission_list',
'mission_start', 'mission_time', 'tupdate');
            end
        end
    end
end
toc
```

Mixed Mode Autonomy

```
tic
% Main simulation parameters

% Changed 6-6-10 to do all communication on a per dt basis instead of by
increments of message time

num_missions = 25;
totaltime = 0;
% load an environment

% Set of new environments

% env_index = 101;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 102;
% env = 'C:\multirobot
model\Environments\env_1x1_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 103;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=4xupper=28yupper=25.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 104;
% env = 'C:\multirobot
model\Environments\env_1x2_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
env_index = 105;
env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=4xupper=28yupper=25.mat';
DA = 50;
AP_location = [26 26];
%
% env_index = 106;
% env = 'C:\multirobot
model\Environments\env_1x4_x=51y=49bottle=51xupper=51yupper=2.mat';
% DA = 50;
% AP_location = [26 26];
%
% env_index = 201;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
```

```
%
% env_index = 202;
% env = 'C:\multirobot
model\Environments\env_1x1_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 203;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 204;
% env = 'C:\multirobot
model\Environments\env_1x2_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 205;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=4xupper=40yupper=37.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 206;
% env = 'C:\multirobot
model\Environments\env_1x4_x=75y=75bottle=75xupper=75yupper=2.mat';
% DA = 75;
% AP_location = [38 38];
%
% env_index = 301;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 302;
% env = 'C:\multirobot
model\Environments\env_1x1_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 303;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 304;
% env = 'C:\multirobot
model\Environments\env_1x2_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 305;
```

```
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=4xupper=53yupper=49.mat';
% DA = 100;
% AP_location = [51 51];
%
% env_index = 306;
% env = 'C:\multirobot
model\Environments\env_1x4_x=101y=101bottle=101xupper=101yupper=2.mat';
% DA = 100;
% AP_location = [51 51];

load(env);
orig_cent = topnode;
dest_cent = bottomnode;

% Controller Information control_type is a flag set to record which
% controller is in use for the simulation
% type 1 is the baseline - fully decentralized, stigmergic approach
% type 2 is decentralized control with communication (DCC)
% type 3 is centralized control
% type 4 is ???
% type 5 is sliding autonomy/ETA Control
control_type = 4;
sim_length = 2000; %DA/vel * num_missions;
vel = DA*num_missions/sim_length;
scale = DA/100;

% sim_length = sim_length/10; % comment this line out for full simulation

% Sensor parameters
fp = 0.02; % Fraction of sensor readings that are false positive
fn = 0.02; % Fraction of sensor readings that are false negative

% Communication Model for 2.4 GHz
F = 6; % 6 dB
k = 1.38e-23; % J/K
T0 = 290; % K
B = 22e6; % Hz
Pn = 10 * log10 ((F + 1) * k * T0 * B * 1000); % dBm
% assume unity gains on antennas for now
PL_d0 = 40;  % dB
d0 = scale; % m
loss_exp = 3.0;
dev = 9; % dB  typical value for log-normal shadowing with open terrain
Bn = 22e6; % Hz
R = 1e6; % bps
Bn_R = Bn/R; % Bn/R

for N = [400 500] %[1 10 20 50 100 200 300 400 500] %[500 400 300 200 100 50
20 10 1]
    low_t = tril(true(N+1),-1);
    AP = N + 1;
    blank_diag = ~eye(N, 'uint8'); % used to eliminate decision to
communicate based on own proximity to decision
    thresh = 0.4 * scale; %[0.4 0.1] * scale
```

```
    for comm_set = [48 48 240 240; 3.7 -10.5 5 -9.2]% replaced overhead with
comm parameters (overhead and Pt)
        overhead = comm_set(1);
        Pt = comm_set(2);
        % Communication Parameters
        data_rate = 1e6; % bits per second
        data_length = 48; % bits
        total_length = data_length + overhead;
        FL = total_length;
        message_time = total_length / data_rate; % seconds

        % Simulation Timing
        [dt dt_type] = min([thresh/vel/2.4 1/vel/10]);
        delta = vel * dt;
        update_time = dt;

        for coupling = [10000 10000 10000 10 10 5; 10000 10 5 10 5 5;] *
scale
            couple_orig = coupling(1);
            couple_dest = coupling(2); % task coupling - possible values of
10,000, 10, 5
            % Distribution about the center of origination
            scaled_cdf_orig = assign_matrix(orig_cent, couple_orig, xtotal,
ytotal, connected);

            % Distribution about the center of origination
            scaled_cdf_dest = assign_matrix(dest_cent, couple_dest, xtotal,
ytotal, connected);

            for iter = 13:13
                clearvars -except blank_diag reach comm_set Pt AP data_rate
data_length total_length message_time Pt F k T0 B Pn PL_d0 d0 loss_exp dev
Bn_R FL fp fn scale num_missions totaltime DA sim_length iter N thresh vel
couple_orig couple_dest WAM2 bottle connected d dest_cent orig_cent nodes
env_index env x_node xlower xtotal xupper y_node ylower ytotal yupper low_t
scaled_cdf_orig scaled_cdf_dest control_type data_rate overhead total_length
message_time AP_location tstep update_time dt dt_type delta;

                % Simulation Variables
                tdynamic = 0;
                tupdate = 0;
                tie = rand(N, 1);
                dist = zeros(N+1, N+1);

                sense_dist = dist;
                message_queue = zeros(N+1, 4); % Pt, x, y, vx, vy, row
corresponds to transmitter/robot
                message_queue(:, 1) = 10; % Queue starts empty
                %central_model = zeros(N, 4); % Time updated, robot_status
                local_model = zeros(N, 4, N+1);

                nearmiss = zeros(1, N, 'int32');
                false_obstacle = zeros(1, N, 'int32');
                bias = randn(N+1, N+1) * 0.1;
                sensitivity = randn(N+1, N+1)*.01 + 1;
```

```
                % Simulation Data (results logging)

                msg_sent = zeros(N+1, N+1, 'int32');
                bandwidth_used = 0;
                msg_sent_AP = 0;
                messages_failed_AP = 0;
                messages_failed = zeros(N+1, N+1, 'int32');
                missed = 0;

                skip = zeros(N, 1, 'uint8'); % initially every robot sends

                distance_traveled = zeros(1, N);
                paused = zeros(1, N, 'int32');
                pause_nocomm = zeros(1, N, 'int32');
                nopause = zeros(1, N, 'int32');
                missions = ones(1, N, 'int32');
                mission_dist = zeros(200, N, 'int32');
                mission_list = zeros(200, N, 'int32');
                mission_time = zeros(200, N);
                orig = zeros(1, N,'int32');
                dest = zeros(1, N, 'int32');
                next_vertex = zeros(1, N, 'int32');

                n_list = (1:1:N)';

                local_model(:, 2, :) = 1; % all robots start full segment
away from next vertex

                % Initial task assignments
                for n=1:N
                    raw_assign = rand(1);
                    orig(n) = find(scaled_cdf_orig > raw_assign, 1);

                    raw_assign = rand(1);
                    dest(n) = find(scaled_cdf_dest > raw_assign, 1);

                    while dest(n) == orig(n)
                        raw_assign = rand(1);
                        dest(n) = find(scaled_cdf_dest > raw_assign, 1);
                    end

                    candidates = find(WAM2(:, orig(n)))'; % faster to search
by column!
                    dlist = d(dest(n), candidates);
                    pathlength = min(dlist);
                    onpath = candidates(dlist == pathlength);
                    next_vertex(n) = onpath(ceil(rand * length(onpath)));
                end

                next_vertex_buffer = next_vertex;
                prev_vertex = orig;
                dist_to_next_vertex = ones(N, 1);
```

```
                local_model(1:N, 4, :) = repmat(orig, N+1, 1)';

                mission_start = orig;
                mission_dist(1, :) = abs(x_node(orig) - x_node(dest)) +
abs(y_node(orig) - y_node(dest));
                mission_list(1, :) = dest;

                dx = delta * (x_node(next_vertex) - x_node(orig));
                dy = delta * (y_node(next_vertex) - y_node(orig));

                x = [x_node(orig); AP_location(1)];
                y = [y_node(orig); AP_location(2)];

                dist_list = pdist([x y], 'euclidean');
                dist = dist * 0;
                dist(low_t) = dist_list;
                dist = dist + dist';

                PL = PL_d0 + 10 * loss_exp * log2(dist/d0)/3.3219280948873622
+ randn(N+1, N+1) * dev;
                SNR = Pt - PL - Pn;
                Pe = 0.5*erfc(sqrt(10.^(SNR/10)*Bn_R));
                PRR = (1 - Pe).^FL;

                path_travel = zeros(N, 1);

                PRRcount = 0;

                opt_route = zeros(1, N, 'int32');
                new_route = zeros(1, N, 'int32');
                old_route = zeros(1, N, 'int32');
                old_data = zeros(1, N, 'int32');
                know_more_route = zeros(1, N, 'int32');
                know_intent_route = zeros(1, N, 'int32');
                no_neighbor = zeros(1, N, 'int32');

                for time = 0:dt:sim_length
                    % Central control check of robot routes
                    % Check when central model next_vertex ~=
next_vertex_buffer
                    % central model is N+1 model of local model
                    % set flag after first check - no need to recalculate
                    % reset flag when central model next_vertex ==
next_vertex_buffer

                    % use central communication model information to send
central route information
                    % number of improved routes should be calculated

                    % calculate list of robots that need to be routed

                    % robot needs route if it is with distance of next vertex
and the next vertex buffer is still the
                    % next vertex
```

```
                    needs_route = (dist_to_next_vertex' < 3.25 * delta &
next_vertex_buffer == next_vertex);
                    num_to_route = sum(needs_route);
                    central_routes = sum(opt_route ~= 0); % determine the
number of robots that need to get a new route from central controller

                    mess_per_dt = min(floor(dt/ message_time) -
central_routes, N);


                    % update robot (n_list(opt_route ~=0)) routes based on
central commands

                    get_from_central = n_list(opt_route ~= 0);

                    if central_routes ~=0

                        rx_cmd = PRR(AP, get_from_central)  > rand(1,
central_routes);
                        msg_sent_AP = msg_sent_AP + sum(rx_cmd);
                        messages_failed_AP = messages_failed_AP +
sum(~rx_cmd);
                        next_vertex_buffer(get_from_central(rx_cmd)) =
opt_route(get_from_central(rx_cmd));
                        opt_route(get_from_central(rx_cmd)) = 0;

                    end

                    % route the robots
                    % calculate alternate central control route information
during this step as well
                    % - avoids duplicate calculations - route will not be
used until time + dt
                    % difference between two routes is difference in local
and central model and
                    % knowledge of goal in central model
                    needs_new_list = n_list(needs_route)';
                    if (~isempty(needs_new_list))
                        for curr_rob = needs_new_list
                            if next_vertex(curr_rob) == dest(curr_rob)
                                % End of mission reached, assign new mission,
                                % increment mission counter
                                orig(curr_rob) = dest(curr_rob);
                                if mod(missions(curr_rob),2) == 0
                                    while dest(curr_rob) == orig(curr_rob)
                                        raw_assign = rand(1);
                                        dest(curr_rob) = find(scaled_cdf_dest
> raw_assign, 1);
                                    end
                                else
                                    while dest(curr_rob) == orig(curr_rob)
                                        raw_assign = rand(1);
                                        dest(curr_rob) = find(scaled_cdf_orig
> raw_assign, 1);
                                    end
```

```
                                end
                                mission_time(missions(curr_rob), curr_rob) =
time;
                                missions(curr_rob) = missions(curr_rob) + 1;
                                mission_dist(missions(curr_rob), curr_rob) =
abs(x_node(orig(curr_rob)) - x_node(dest(curr_rob))) +
abs(y_node(orig(curr_rob)) - y_node(dest(curr_rob)));
                                mission_list(missions(curr_rob), curr_rob) =
dest(curr_rob);
                            end
                            % replaced unique with part of that code
                            if local_model(:, 4, curr_rob) == local_model(:,
4, AP)
                                match4 = 1;
                            else
                                match4 = 0;
                            end
                            if local_model(:, 2, curr_rob) == local_model(:,
2, AP)
                                match2 = 1;
                            else
                                match2 = 0;
                            end
                            curr_vertices = sort(local_model([1:curr_rob-1
curr_rob+1:N],4, curr_rob));
                            % eliminate redundant values to check
                            if ~isempty(curr_vertices)
                                cv_db = diff(curr_vertices);
                                numelcv = N-1;
                                cv_ind = cv_db ~= 0;
                                cv_ind(numelcv) = 1;
                                curr_vertices = curr_vertices(cv_ind);
                            end
                            if match4 == 1
                                curr_vertices_cr = curr_vertices;
                            else
                                curr_vertices_cr =
sort(local_model([1:curr_rob-1 curr_rob+1:N],4, AP));
                                % eliminate redundant values to check
                                if ~isempty(curr_vertices_cr)
                                    cv_db = diff(curr_vertices_cr);
                                    numelcv = N-1;
                                    cv_ind = cv_db ~= 0;
                                    cv_ind(numelcv) = 1;
                                    curr_vertices_cr =
curr_vertices_cr(cv_ind);
                                end
                            end


                            candidates = find(WAM2(:,
next_vertex(curr_rob))); % faster to search by column!
                            % form path sets
                            cp_step1 = candidates(candidates ~=
prev_vertex(curr_rob));
```

```
                            if sum(cp_step1 == dest(curr_rob)) == 1 % if any
of the first steps in possible paths is the destination - go there and no
need to check paths
                                next_vertex_buffer(curr_rob) =
dest(curr_rob);
                            else
                                path_set = zeros(2, length(cp_step1)*3);
                                path_index = 0;
                                for j = 1:size(cp_step1)
                                    poss_step = find(WAM2(:,cp_step1(j)));
                                    cp_step2 = poss_step(poss_step ~=
next_vertex(curr_rob));
                                    for g = 1:length(cp_step2)
                                        path_index = path_index + 1;
                                        path_set(1, path_index) =
cp_step1(j);
                                        path_set(2, path_index) =
cp_step2(g);
                                    end
                                end
                                path_set = path_set(:, 1:path_index);
                                dlist_raw = d(dest(curr_rob),
path_set(2,:))+2;
                                dlist = dlist_raw;
                                dlist_central_no_intent = dlist_raw;
                                dlist_central = dlist_raw;
                                for p = 1:size(path_set, 2)
                                    can_reach_path = [reach(path_set(1,
p)).can_reach_path'; reach(path_set(2, p)).can_reach_path'];
                                    potential_conflicts =
curr_vertices(ismember(curr_vertices, can_reach_path));
                                    off_base = dist_to_next_vertex(curr_rob)
+ 1; %distance to reach first point on possible next path

                                    if match4 == 1 && match2 == 1
                                        if isempty(potential_conflicts) ~= 1
                                            for u = potential_conflicts'
                                                check_cond = local_model(:,
4, curr_rob) == u; % index to the robot where the conflict is
                                                cm = local_model(check_cond,
2, curr_rob);
                                                d1 = d(u, path_set(1,p));
                                                dest_check =
dest(check_cond);
                                                d2 = d(path_set(1,p),
dest_check);
                                                d3 = d(u, dest_check);
                                                offset_dist1 = off_base - (d1
+ cm);
                                                offset_dist2 = off_base + 1 -
(d(u, path_set(2,p)) + cm);
                                                penalty = max(thresh -
min(abs(offset_dist1), abs(offset_dist2)),0);
                                                penalty_cr = (1 ./ (d1 + d2 -
d3 + 1)) * penalty; % penalty is reduced if the robot is unlikely to visit
that node on its path
```

```
                                                dlist(p) = dlist(p) +
sum(penalty);
                                                dlist_central(p) =
dlist_central(p) + sum(penalty_cr);
                                            end
                                        end
                                        dlist_central_no_intent = dlist;

                                    elseif match4 == 1 && match2 == 0
                                        if isempty(potential_conflicts) ~= 1
                                            for u = potential_conflicts'
                                                check_cond = local_model(:,
4, curr_rob) == u; % index to the robot where the conflict is
                                                cm = local_model(check_cond,
2, curr_rob);
                                                cm_cr =
local_model(check_cond, 2, AP);
                                                d1 = d(u, path_set(1,p));
                                                dest_check =
dest(check_cond);
                                                d2 = d(path_set(1,p),
dest_check);
                                                d3 = d(u, dest_check);
                                                offset_dist1 = off_base - (d1
+ cm);
                                                offset_dist2 = off_base + 1 -
(d(u, path_set(2,p)) + cm);
                                                offset_dist1_cr = off_base -
(d1 + cm_cr);
                                                offset_dist2_cr = off_base +
1 - (d(u, path_set(2,p)) + cm_cr);

                                                penalty = max(thresh -
min(abs(offset_dist1), abs(offset_dist2)),0);
                                                penalty_no_intent =
max(thresh - min(abs(offset_dist1_cr), abs(offset_dist2_cr)),0);
                                                penalty_cr = (1 ./ (d1 + d2 -
d3 + 1)) * penalty_no_intent; % penalty is reduced if the robot is unlikely
to visit that node on its path
                                                dlist(p) = dlist(p) +
sum(penalty);
                                                dlist_central_no_intent(p) =
dlist_central_no_intent(p) + sum(penalty_no_intent);
                                                dlist_central(p) =
dlist_central(p) + sum(penalty_cr);
                                            end
                                        end

                                    else
                                        if isempty(potential_conflicts) ~= 1
                                            for u = potential_conflicts'
                                                check_cond = local_model(:,
4, curr_rob) == u; % index to the robot where the conflict is
                                                cm = local_model(check_cond,
2, curr_rob);
                                                d1 = d(u, path_set(1,p));
```

```
                                                offset_dist1 = off_base - (d1
+ cm);
                                                offset_dist2 = off_base + 1 -
(d(u, path_set(2,p)) + cm);
                                                penalty = max(thresh -
min(abs(offset_dist1), abs(offset_dist2)),0);
                                                dlist(p) = dlist(p) +
sum(penalty);
                                            end
                                        end

                                        potential_conflicts_cr =
curr_vertices_cr(ismember(curr_vertices_cr, can_reach_path));
                                        if isempty(potential_conflicts_cr) ~=
1
                                            for u = potential_conflicts_cr'
                                                check_cond = local_model(:,
4, AP) == u; % index to the robot where the conflict is
                                                cm = local_model(check_cond,
2, AP);
                                                d1 = d(u, path_set(1,p));
                                                dest_check =
dest(check_cond);
                                                d2 = d(path_set(1,p),
dest_check);
                                                d3 = d(u, dest_check);
                                                offset_dist1 = off_base - (d1
+ cm);
                                                offset_dist2 = off_base + 1 -
(d(u, path_set(2,p)) + cm);
                                                penalty = max(thresh -
min(abs(offset_dist1), abs(offset_dist2)),0);
                                                penalty_cr = (1 ./ (d1 + d2 -
d3 + 1)) * penalty; % penalty is reduced if the robot is unlikely to visit
that node on its path
                                                dlist_central_no_intent(p) =
dlist_central_no_intent(p) + sum(penalty);
                                                dlist_central(p) =
dlist_central(p) + sum(penalty_cr);
                                            end
                                        end
                                    end
                                end
                                % path selection
                                pathlength = min(dlist);
                                pathlength_cr = min(dlist_central);
                                onpath = path_set(1, dlist == pathlength);
                                onpath_cr = path_set(1, dlist_central ==
pathlength_cr);
                                % set next step on path to dest
                                next_vertex_buffer(curr_rob) =
onpath(randi(length(onpath)));
                                if sum(next_vertex_buffer(curr_rob) ==
onpath_cr) == 0
                                    opt_route(curr_rob) =
onpath_cr(randi(length(onpath_cr)));
```

```
                                    new_route(curr_rob) = new_route(curr_rob)
+ 1;

                                    pathlength_cr_no_intent =
min(dlist_central_no_intent);
                                    onpath_cr_no_intent = path_set(1,
dlist_central_no_intent == pathlength_cr_no_intent);

                                    neighbors = d(next_vertex,
next_vertex(curr_rob))<4;
                                    neighbors(curr_rob) = 0;
                                    if sum(neighbors) > 0
                                        AP_newer = local_model(neighbors, 1,
AP) >= local_model(neighbors, 1, curr_rob);
                                        if sum(AP_newer) < sum(neighbors)
                                            old_route(curr_rob) =
old_route(curr_rob) + 1;
                                        end
                                        AP_current = time <=
local_model(neighbors, 1, AP) + local_model(neighbors, 2, AP)/vel + dt;
                                        if sum(AP_current) < sum(neighbors)
                                            old_data(curr_rob) =
old_data(curr_rob) + 1;
                                            opt_route(curr_rob) = 0; % can't
use this route - central model info is too old
                                        end
                                    else
                                        no_neighbor(curr_rob) =
no_neighbor(curr_rob) + 1;
                                    end



                                    if sum(next_vertex_buffer(curr_rob) ==
onpath_cr_no_intent) == 0
                                        % Different info caused route change
instead of intent
                                        know_more_route(curr_rob) =
know_more_route(curr_rob) + 1;

                                    else
                                        % path difference is due to central
knowing intent
                                        know_intent_route(curr_rob) =
know_intent_route(curr_rob) + 1;
                                    end
                                else
                                    opt_route(curr_rob) = 0;
                                end

                            end
                        end
                    end

                    % determine robots that transmit during this dt
                    [sorted rows] = sortrows(message_queue);
```

```
                    last_in_queue = find(sorted(:,1) == 10, 1) - 1;
                    if last_in_queue > 0
                        valid_queue = rows(sorted(:,1)<10);
                        valid_queue = valid_queue(1:min(mess_per_dt,
last_in_queue));
                        for msg = valid_queue'
                            send_ok = PRR(:, msg)>rand(N+1, 1);
                            msg_sent(:, msg) = msg_sent(:, msg) +
int32(send_ok); % track percentage of bandwidth used successfully
                            messages_failed(:, msg) = messages_failed(:, msg)
+ int32(~send_ok);
                            message_queue(msg, 1) = 10;
                            % update local model
                            local_model(msg, 1, send_ok) = time;
                            local_model(msg, 2:4, send_ok) =
repmat(message_queue(msg, 2:4), sum(send_ok), 1)';
                        end
                    end
                    for lm = 1:N
                        local_model(lm, 2, lm) = dist_to_next_vertex(lm);
                        local_model(lm, 1, lm) = time;
                    end

                    dist_list = pdist([x y], 'euclidean');
                    dist = dist * 0;
                    dist(low_t) = dist_list;
                    dist = dist + dist';

                    % Communication propagation model

                    PRRcount = PRRcount + 1;

                    if PRRcount == 2

                        PL_list = PL_d0 + 10 * loss_exp *
log2(dist_list/d0)/3.3219280948873622;
                        PL = PL * 0;
                        PL(low_t) = PL_list;
                        PL = PL + PL' + randn(N+1, N+1) * dev;
                        SNR = Pt - PL - Pn;
                        Pe = 0.5*erfc(sqrt(10.^(SNR/10)*Bn_R));
                        PRR = (1 - Pe).^FL;
                        PRRcount = 0;

                    end

                    sense_dist = dist .* sensitivity + bias;

                    % State update (in loop)
                    for n = 1:N
                        stp = 1;
                        % Check if robot n is currently at a vertex
                        if (abs(x_node(next_vertex(n)) - x(n)) +
abs(y_node(next_vertex(n)) - y(n))) <= delta/2
                            % Check if next command has been generated
```

```
                            if next_vertex_buffer(n) ~= next_vertex(n) % was
-1
                                % Set position to current next_vertex since
we're there
                                x(n) = x_node(next_vertex(n));
                                y(n) = y_node(next_vertex(n));
                                % Increment segment count
                                path_travel(n) = path_travel(n) + 1;
                                % Set vertex
                                prev_vertex(n) = next_vertex(n);
                                next_vertex(n) = next_vertex_buffer(n);
                                local_model(n,4,n) = next_vertex(n);

                                %
next_vertex_buffer(n) = -1;
                                dx(n) = delta * (x_node(next_vertex(n)) -
x(n));
                                dy(n) = delta * (y_node(next_vertex(n)) -
y(n));
                            else
                                stp = -1;
                            end
                        end

                        % Check for conflicts
                        %conflicts = find(dist(:,n)<=thresh); % this is the
distance that should be replaced with noisy sensor data!
                        conflicts = find(sense_dist(n_list,n)<=thresh); %
this is the distance that should be replaced with noisy sensor data!
                        num_conflicts = length(conflicts);
                        sensor_fp = rand(1);
                        if sensor_fp <= fp
                            false_obstacle(n) = false_obstacle(n) + 1;
                            stp = 0;
                        elseif num_conflicts > 1
                            sensor_fn = rand(1);
                            n_goal = next_vertex(n);
                            if sensor_fn <= fn
                                nearmiss(n) = nearmiss(n) + 1;
                            else
                                for q = conflicts'
                                    dist_n_to_next_pt = abs(x(n) -
x_node(n_goal)) + abs(y(n)- y_node(n_goal));
                                    if ( ( next_vertex(q) == n_goal ) && ( q
~= n ) )
                                        % check to see if q is closer to goal
than n
                                        dist_q_to_next_pt = abs(x(q) -
x_node(n_goal)) + abs(y(q)-y_node(n_goal));
                                        if ( dist_n_to_next_pt >=
dist_q_to_next_pt )
                                            % check for tie
                                            if ( dist_n_to_next_pt ==
dist_q_to_next_pt && tie(n) < tie(q) )
                                            else
                                                stp = 0;
```

```
                                        break
                                    end
                                end
                            end
                        end
                    end
                    % Update x and y based on dx, dy, and conflict state
                    if stp == 1
                        x(n) = x(n) + dx(n);
                        y(n) = y(n) + dy(n);
                        distance_traveled(n) = distance_traveled(n) +
delta;
                        nopause(n) = nopause(n) + 1;
                    elseif stp == 0
                        paused(n) = paused(n) + 1;
                    else
                        pause_nocomm(n) = pause_nocomm(n) + 1;
                    end
                end

                % Load message queue based on the number of neighbors
that may need information
                % (and how close they are to deciding next step in route)
                % Transmit more frequently when far from Access Point -
more likely to fail
                % use dist(AP, :) for this weighting


                % update time was set to be the same as dt => 12
messages/ per segment for DA = 50
                % with adaptive communication rate will send 1/3 as often
unless it appears
                % other robot needs the information or the robot is far
from the access point
                missed = missed + sum(message_queue(:,1) ~= 10);
                dist_to_next_vertex = abs(x_node(next_vertex) -
x(n_list)) + abs(y_node(next_vertex) - y(n_list));

                send_set = (skip == 0); % add messages to queue when skip
is 0


                skip = skip - 1; % decrement the skip counter

                new_in_queue = sum(send_set); % length of new part of
queue

                if new_in_queue ~= 0

                    message_queue(send_set, 1) = rand(new_in_queue, 1);
                    message_queue(send_set, 2) =
dist_to_next_vertex(send_set);
                    message_queue(send_set, 3) =
next_vertex_buffer(send_set)';
                    message_queue(send_set, 4) = next_vertex(send_set)';
```

```
                        % calculate skip for the messages that transmitted
                        far_AP = uint8(dist(send_set, AP) >= 20);
                        cp = zeros(new_in_queue, 1, 'uint8');
                        cc_count = 1;
                        for cc = n_list(send_set)'
                            cp(cc_count) = sum(d(local_model(:,4, cc),
next_vertex(cc))<4 & local_model(:,2, cc) <= 5 *delta & blank_diag(:, cc)) +
local_model(cc, 2, cc) >= 11 * delta;
                            cc_count = cc_count + 1;
                        end
                        % send more often if far from AP (4 messages per
segment => 6 per segment)
                        % reduce weight on message by one round for each
robot that might need decision
                        % need to set thresholds in comparisons (currently 4,
5*delta, and 20)
                        skip(send_set) = max(2 - far_AP - cp, 0);

                    end

%                                       message_queue(1:N, 1) = rand(N, 1);
%                                       message_queue(1:N, 2) =
dist_to_next_vertex;
%                                       message_queue(1:N, 3) =
next_vertex_buffer';
%                                       message_queue(1:N, 4) =
next_vertex';
                end
                nam_var = ['control=' num2str(control_type) 'N=' num2str(N)
'env=' num2str(env_index) 'thresh=' num2str(thresh) 'len='
num2str(sim_length) 'couple_orig=' num2str(couple_orig) 'couple_dest='
num2str(couple_dest) 'overhead=' num2str(overhead) 'Pt=' num2str(Pt) 'iter='
num2str(iter)];
                looptime = toc - totaltime
                totaltime = toc;
                mission_time = mission_time - [zeros(1, N);
mission_time(1:199, :)];
                mission_time = max(mission_time, 0);
                save(['C:\multirobot model\Logs\sim-' nam_var '.mat'],
'msg_sent_AP', 'messages_failed_AP', 'new_route', 'old_route',
'know_more_route', 'know_intent_route', 'no_neighbor', 'old_data',
'data_length', 'bandwidth_used', 'overhead', 'Pt', 'message_time', 'missed',
'msg_sent', 'messages_failed', 'bias', 'sensitivity', 'fp', 'fn', 'DA',
'sim_length', 'nearmiss', 'false_obstacle', 'num_missions', 'looptime',
'AP_location', 'N', 'candidates', 'conflicts', 'control_type', 'couple_dest',
'couple_orig', 'delta', 'dest', 'dest_cent', 'dist', 'distance_traveled',
'dlist', 'dt', 'dt_type', 'dx', 'dy', 'env', 'env_index', 'iter', 'missions',
'next_vertex', 'nopause', 'num_conflicts', 'onpath', 'orig', 'orig_cent',
'path_travel', 'pathlength', 'paused', 'pause_nocomm', 'raw_assign',
'sim_length', 'thresh', 'vel', 'x', 'y', 'mission_dist', 'mission_list',
'mission_start', 'mission_time', 'tupdate');
            end
        end
    end
end
```