

7-2015

Efficient Simultaneous Task and Motion Planning for Multiple Mobile Robots Using Task Reachability Graphs

Brad Woosley

University of Nebraska at Omaha

Follow this and additional works at: <https://digitalcommons.unomaha.edu/studentwork>

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Woosley, Brad, "Efficient Simultaneous Task and Motion Planning for Multiple Mobile Robots Using Task Reachability Graphs" (2015). *Student Work*. 2904.

<https://digitalcommons.unomaha.edu/studentwork/2904>

This Thesis is brought to you for free and open access by DigitalCommons@UNO. It has been accepted for inclusion in Student Work by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.



Efficient Simultaneous Task and Motion Planning for Multiple Mobile Robots Using Task Reachability Graphs

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

University of Nebraska at Omaha

by

Brad Woosley

July 2015

Supervisory Committee:

Raj Dasgupta

Victor Winter

Vyacheslav Rykov

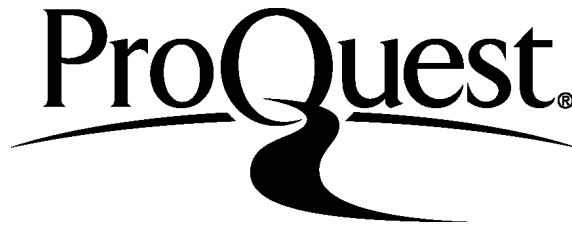
ProQuest Number: 1597579

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 1597579

Published by ProQuest LLC (2015). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Efficient Simultaneous Task and Motion Planning for Multiple Mobile Robots Using Task Reachability Graphs

Brad Woosley, MSc

University of Nebraska, 2015

Advisor: Raj Dasgupta

In this thesis, we consider the problem of efficient navigation by robots in initially unknown environments while performing tasks at certain locations. In initially unknown environments, the path plans might change dynamically as the robot discovers obstacles along its route. Because robots have limited energy, adaptations to the task schedule of the robot in conjunction with updates to its path plan are required so that the robot can perform its tasks while reducing time and energy expended. However, most existing techniques consider robot path planning and task planning as separate problems. This thesis plans to bridge this gap by developing a unified approach for navigating multiple robots in uncertain environments. We first formalize this as a problem called task ordering with path uncertainty (TOP-U) where robots are provided with a set of task locations to visit in a bounded environment, but the length of the path between a pair of task locations is initially known only coarsely by the robots. The robots must find the order of tasks that reduces the path length to visit the task locations. We then propose an abstraction called a task reachability graph (TRG) that integrates the robots task ordering and path planning. The TRG is updated dynamically based on inter-task path costs calculated by the path planner. A Hidden Markov Model-based technique calculates the belief in the current path costs based on the environment perceived by the robot's sensors. We then describe a Markov Decision Process-based algorithm used by each robot in a distributed manner to reason about the path lengths between tasks and select the paths that reduce the overall path length to visit the task locations. We have evaluated our algorithm in simulated and hardware robots. Our results show that the TRG-based approach performs up to 60% better in planning and locomotion times with 44% fewer replans, while traveling almost-similar distances as compared to a greedy, nearest task-first selection algorithm.

Grant Acknowledgement

This work was funded in part by the Office of Naval Research as part of the COMRADES project, and a GRACA Grant from the University of Nebraska at Omaha.

Contents

1	Introduction	1
2	Related Work	5
2.1	Task Allocation	5
2.2	Path planning	8
2.2.1	Single robot path planning	8
2.2.2	Multi-robot path planning and collision avoidance	10
3	STAMP Problem Formation	13
4	Techniques for Solving STAMP Problem	18
4.1	Updating Edge Costs	19
4.2	Updating Edge Availabilities	21
4.3	TOP-U Solution using Markov Decision Process	24
4.4	Robot Navigation and Task Selection	26
4.5	Coordinating paths between robots to avoid collisions	28
4.6	Example	31
5	Experimental Results	36
5.1	Simulated Experiments	36
5.1.1	Setup	36
5.1.2	Results	39

5.2	Physical Robots	47
5.2.1	Setup	47
5.2.2	Results	48
6	Conclusions and Future Work	49
6.1	Lessons Learned	49
6.2	Future Works	50

List of Figures

1.1	Paths generated	2
3.1	Example TRG	14
4.1	Structure of proposed technique	18
4.2	Bayesian Network	21
4.3	Example collision avoidance	30
4.4	Example part 1, tasks are shown by red circles	33
4.5	Example part 2, tasks are shown by red circles	34
4.6	Example part 3, tasks are shown by red circles	34
4.7	Example part 4, tasks are shown by red circles	35
5.1	Simulation setup	38
5.2	Robots used	40
5.3	Simulation result $\Gamma = 1000$	41
5.4	Replans results as Γ changes	42
5.5	Distance results as Γ changes	43
5.6	Timing results as Γ changes	45
5.7	Overhead and side view of environment used for testing with physical robots, white dots represent the task locations	47

List of Tables

4.1	Table of PLL observations for each edge of TRG	32
5.1	Change in Distance, Non-switching replans, Switching replans, Plan time, and Navigation time as Γ increases	46
5.2	Physical robot experiment results	48

List of Algorithms

1	Select Task	26
2	Update TRG	27
3	Collision Avoidance	29

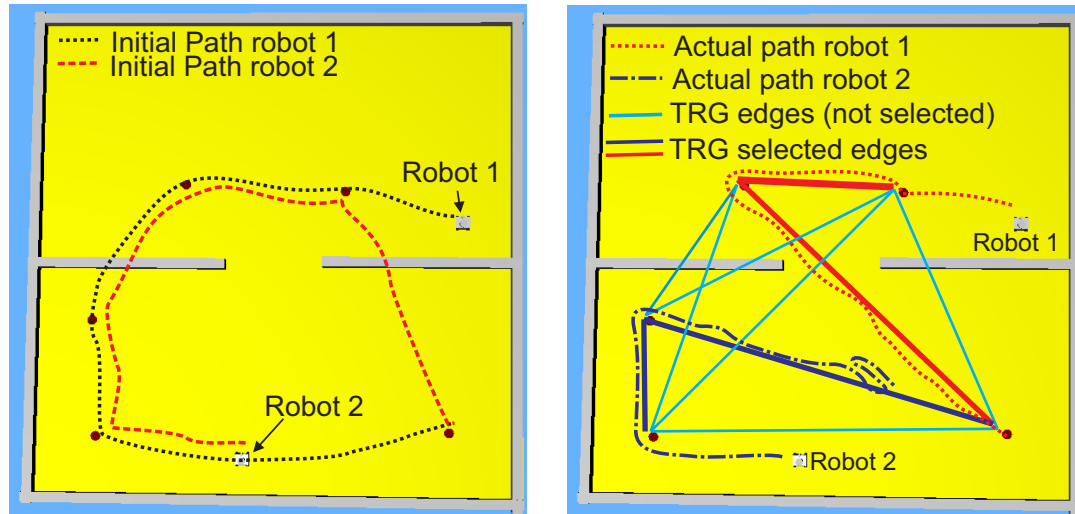
Chapter 1

Introduction

In this thesis, we investigate the scenario where there exists a bounded environment, and in this environment, there are a group of spatially distributed tasks that must be completed by a group of robots. Due to the limited resources that robots have, especially their power source, it is very important to minimize the total cost expended by the robots in accomplishing all of the tasks.

This problem is encountered in many applications of multi-robot systems such as automated surveillance [35], robotic demining [25], and automated inspection of engineering structures [27]. As a motivating example, for performing standoff detection of explosives or landmines using autonomous robots, multiple robots with different types of sensors are provided with a coarse map containing locations of objects of interest. The robots are required to autonomously plan their paths to get in proximity of each object of interest so that they can analyze the object with their detection sensors. Due to the likely remote area that the robots are operating in, it is very important that the robot's conserve their resources and perform as much of the given tasks as possible with their resources.

In this scenario, there are two important problems that must be solved—task planning and motion planning. The goal of task planning is to find an ordering over the set of tasks that the robots must perform. The found ordering must minimize some cost metric,



(a) Path planned by robots without knowledge of obstacles using task locations only

(b) Paths planned by robots using TRG

Figure 1.1: Paths generated

such as the distance traveled or energy expended by robots to visit the task locations, and be feasible. A feasible task ordering is one that maintains any constraints imposed on the tasks or dependencies between tasks. For example, if there are two tasks, collecting a soil sample, and delivering it to scientists for analysis, there is a dependency between the two tasks, the soil sample must be collected before it can be delivered for analysis. Certain tasks may require visits from multiple robots, where a single robot can only perform a fraction of the task and a second robot is required to. On the other hand, the goal of motion planning is to find a path for the robot to follow through the environment while avoiding collisions with obstacles in the environment and minimizing some cost metric. Conventionally, these two problems have been solved separately, with the task planning methods assuming that cost to reach a task is fixed and known to all robots as soon as the robot becomes aware of the task [4]. However, this assumption does not hold in initially unknown, or partially unknown, environments. As a robot follows along its initial path to a task, it may discover the locations of new obstacles that force the path to be updated, thus increasing the length of the path. An example of this is shown in Figure 1.1(a) where

two robots are tasked with visiting 5 task locations, where each task location only requires a visit from one robot. However, initially the robots have no knowledge of the wall-like obstacles in the environment, or the paths that the other robot will take. As can be seen, both robot's paths and will intersect with obstacles, which once they are detected will cause the path lengths to increase. The goal is to incorporate data from the path planner into the task planner to adjust the task plan as there are changes to the robots perception of the environment

Computing a task plan is a computationally challenging problem, and quickly becomes infeasible as the number of tasks and robots increase, and has been shown to be an NP-hard problem [12]. In order to handle this problem, researches have proposed approximations to task allocation to provide close to optimal solutions in polynomial time.

To address this problem, we propose an approach that dynamically recalculates the task plan based on the length of the current best path between tasks, and a belief in the availability of the tasks for completion. To facilitate this, we propose an abstraction called a task reachability graph (TRG) that represents the reachability between tasks. Each edge in the TRG represents a path between two task locations or the robot and a single task location, and has associated with it the cost of the path returned by a path planner. Each edge also has a probability associated with it that represents the robot's belief in the current edge cost. The beliefs are calculated and updated using a Temporal Bayesian Network (TBN) that encodes the interactions between the state of the world and what the robot can observe about the length of the paths between all of the tasks. Solving of the TBN is then handled using a Hidden Markov Model (HMM). The beliefs are then used inside a Markov Decision Process (MDP) model to calculate the best task for the robot to travel to, this induces a schedule over the tasks for the robot to visit. This causes only select edges from the TRG to be selected for the robot to move along, which can be seen in Figure 1.1(b), which shows all the edges of the TRG, and the edges that each robot selects to navigate until all tasks have been visited by exactly one robot.

To verify our approach, we tested it in both simulation and on physical robots while varying the environment layout, number of tasks, and how many robots are needed to complete a task. Our results show that the TRG-based approach performs up to 60% better in planning and locomotion times with 44% fewer replans, while traveling almost-similar distances as compared to a greedy, nearest task-first selection algorithm.

The rest of this document has the following structure. In Chapter 2 we discuss the related works on this topic. Chapter 3 presents a model for Simultaneous Task and Motion Planning, then in Chapter 4 we present our proposed approach in that model. Then in Chapter 5 we discuss the experiments we performed to validate this approach and the results of the various experiments. And finally in Chapter 6 we summarize our work, discuss the conclusions we can draw, and provide a discussion on the future work for this topic.

Chapter 2

Related Work

2.1 Task Allocation

To properly discuss the problem of task allocation, it is important to provide a method for defining the various parameters that are apparent in the various types of task allocation. This was what Gerkey and Mataric did in [12]. They proposed a three axis based system. The first axis describes how many tasks a robot can be executing simultaneously, a single task (ST) system means that every robot in the environment can only execute a single task at a time, where as a multiple task (MT) environment means that some of the robots in the environment can execute multiple tasks at the exact same time. The second axis describes how many robots are required to execute a task. A single robot (SR) environment has every task in the environment require only a single robot to complete it, where as a multiple robot (MR) environment has tasks that require more than one robot to complete the task. The third, and final, axis describes how the robots learn about the environment and allocate the tasks among the robots. An instantaneous allocation (IA) environment has all of the information available at the start, thus meaning that the initial allocation of tasks is the final one as there is no more new information gained to later update the allocation of tasks. A time-extended allocation (TA) environment, on the other hand, has

more information available over time to allow for updates to the current task allocation. Using the aforementioned taxonomy, the system we are proposing a solution to would be described as a ST-MR-TA system. A robot can only execute, or attempt to execute a single task at a time, multiple robots are required for each task to be completed, and more information is available over time about the environment and tasks, requiring updates to the initial allocation, or task plan.

In [11], Gerkey and Mataric proposed a task allocation method using auctions. The auction runs in a distributed manner with each robot placing bids on how well they can complete the task. An auctioneer – which could be the user, a pre-scheduled task coming available or any other source – announces to the robots when a new task is available, including the details of the task needed for the robot to make a decision about it. The robots then evaluate how well they can accomplish that task based on the metrics for the task given. The robots publish how well they can accomplish the task, and the one with the best bid is assigned the task. The robot is also only allowed to work on the task for a certain amount of time before it is revoked, unless there was sufficient progress on completing the task during that time budget.

In [32], Wicke, Freelan, and Luke proposed a method in contrast to the allocation-based methods, like mentioned above. Their proposed solution is modeled after the idea of bail bondsmen and bounty hunters. The bondsmen issue a reward for completion of a task by the robots, the reward offered increases over time as task goes without being completed. The robots then select, based on the reward and a perception of how well the task could be completed, a task to complete. Unlike auction methods, where only robot can commit to a task, their approach allows multiple robots to commit to the same task, which provides robustness in-case the initial robot assigned to the task is unable to complete it in a reasonable time, because another robot will select it. This approach is similar to the one we propose in that multiple robots can attempt to complete a single task at any given time. However, our approach does not provide a reward for completing a task, instead, the task

is selected based on the distance to the task and the likelihood of being able to complete it. Also, in our approach, the cost (analogous to their reward) does not change over time, instead it always represents the robot’s perception of the environment and cost to reach that particular task from the previous replanning point.

There have also been several symbolic task allocation planner that have been proposed in the past [26]. For example, in [33], Wolfe, Marthi, and Russell propose a symbolic AI to integrate information about the tasks into the path planning of a robot while handling uncertainty in the robot’s motion in the environment. The proposed method provides a means to encode mobile manipulator problems as vertically integrated hierarchical task networks (HTNs), where the low level motions handled using rapidly-exploring random trees (RRT). The authors also provide an algorithm called State-Abstracted Hierarchical Task Network (SAHTN) which provides a mechanism for speeding up the search by determining what information for a particular subtask is irrelevant. This mechanism allows for re-use of previous plans that have the same parameters by discarding irrelevant information.

In [34], we proposed a solution to the Multi-Robot Task Allocation (MRTA) problem. In this approach, robots only considered the straight line distance between their current location and all other tasks that needed to still be done. The only time a robot was told to switch what task they were going to was when the straight line distance to another task became less than the straight line distance to the task it was currently going to. Our current approach, however, adds information about the actual path that the robot will need to follow between tasks, allowing it to much sooner determine that another task is better. Also our current approach calculates a belief in the current path length that represents the certainty that the robot has in each path being available for following.

Task and motion planning has also been studied in the context of mobile manipulator robots. For example, Sucan and Kavraki proposed a data structure called a Task Motion Multigraph (TMM) to encode the task and motion dependencies between the tasks in [29].

This approach exploits a property of mobile manipulator robots. Namely that there are multiple abstract motions that can accomplish the same task. For example, to pick up an object, the robot can move its base, left arm, right arm, or any combination of them to accomplish this task. To find the order of motions needed to accomplish the sequence of tasks, the authors used Dijkstra’s algorithm to determine the least cost sequence of actions to perform. The authors then extended this approach to use Markov Decision Processes (MDPs) in [30]. These works provide a good system for the Mobile Manipulator problem, however, our approach is an approach for mobile robots, where the only mechanism to complete a task is to navigate to it. Due to this difference, a direct mapping between the TMM and this problem domain does not provide the same optimizations as the TMM takes advantage of. For a mobile robot, the ways to accomplish each task are too large to enumerate in the TMM, and instead mapping the goal of reaching all tasks, causes the graph to grow very large and become impractical. In our approach, we keep the same idea of using a task graph to maintain the relationships between tasks, but instead of encoding all the ways to accomplish a task, we instead find the optimal path between two tasks given the current knowledge of the environment, and select the task to go to that minimizes our expected cost overall.

2.2 Path planning

2.2.1 Single robot path planning

Single robot motion planning is an important component of task and motion planning, as each robot requires a path to follow to reach their specific goal location. A simple path planner is called the Bug Algorithm [23]. In this algorithm, the robot travels towards the goal until it finds an obstacle, then similar to how a bug navigates, follows the boundary of the obstacle until it can find a way towards the goal. However, this approach does not take into account the location or geometry of obstacles until the robot encounters it. Another

method is potential fields [15] where two functions are defined that provide attraction towards the goal and repulsion away from obstacles. However, concave obstacles can cause a problem with local minima, where the robot gets stuck in a location that any motion from that point causes the robot to move to a worse location before it can make progress towards the goal.

In visibility graph roadmaps [21], the corners of the obstacles are connected by straight lines if that line does not intersect with another obstacle, the robot then navigates on this graph between the closest point to the robot, and leaves the roadmap at the closest point to the goal. Another roadmap based path planner is Silhouette roadmap [3], where the robot plans paths using straight lines to the edges of the obstacles that the robot can see while moving in the direction of the goal. In Voronoi roadmaps [1], the Generalized Voronoi Diagram is created, which are the points that are equal distances from the closest two obstacles.

Extending the idea of roadmaps, is Probabilistic Roadmap Planner (PRM) [14], where random collision free configurations are sampled from the configuration space and a simple path is planned between the samples, for example, by seeing if a collision free straight line path between the samples is possible. Over time, this planner has been extended to handle various special cases. For example, Missiuro and Roy in [24] proposed an extension to PRM that accounts for uncertainty in the location of obstacles in the environment. This was done by having the weight of each edge in the roadmap be a function of not only euclidean distance, but also the probability of colliding with an obstacle in the environment. The obstacles are defined by a sequence of vertices with associated Gaussian distributions that represent the positions where the obstacle vertex really could be. In our work, we have used this planner as the underlying path planner, because of it's ability to account for the noise inherent in the robot's localization, obstacle detection, and representation of obstacles in memory. This helps ensure that the robot does not collide with any obstacles in the environment.

In the same vein as PRM, are Rapidly-Exploring Random Trees (RRT) [13, 18, 17, 22, 9]. Like PRM, RRTs build a roadmap by sampling collision free configurations of the robot from the configuration space, and connects samples with collision free straight line paths. However, instead of building a graph that can be used for multiple path planning queries, RRT builds a single tree that is useful only for finding a path from the start to goal location. This helps speed up planning in cases where only one path is needed. However, in our work, we need to be able to generate multiple paths, therefore, PRM was a better choice.

There have also been methods proposed that use Markov Decision Processes (MDP) [26] to find paths in dynamic environments. One such planner was proposed by Loibl, Meyer-Delius, and Pfaff in [20]. They proposed using an MDP where the states were a tuple of the location in the environment, and the arrival time at that state. The actions are moving to any adjacent state, with a probability of successfully moving between the states, and a cost function based on the time needed to move between the states. We have modeled our task planner on this idea. However, instead of planning in the configuration space, we use the MDP to plan in the task space to help find an ordering over the tasks.

Another path planning method is D* [8, 19, 16], which focuses on decomposing the environment into grid squares, where each grid square is a configuration of the robot, then uses an informed search algorithm, with modifications to account for obstacles discovered during runtime, to find a path from the start grid square to the goal grid square.

2.2.2 Multi-robot path planning and collision avoidance

A common problem in multi-robot environments is handling potential collisions between robots. Single robot path planners do not take into account the locations of other robots in the environment or their paths through the environment. This leads to a high likelihood of collisions between robots. To solve this problem, it is possible to plan in the joint configuration space, accounting for every possible action of each robot in the environment. However planning in the joint configuration space can become infeasible as the number

of robots increase. To handle this problem, Wagner and Choset proposed a decoupled planning system in [31], which allowed for planning in multi-robot systems, while avoiding robot-robot collisions and without requiring a search of the entire joint configuration space. They do this through use of an algorithm named M^* , which is a modified A^* search that limits the number of states that are expanded at from each state of the search. They have also proven that M^* is both complete and optimal.

Another approach to handling multi-robot coordination was proposed by Desaraju and How in [6]. The proposed method is the Decentralized Multi-Agent Rapidly-exploring Random Tree (DMA-RRT) technique. It is an approach to planning paths for multiple agents operating in the same environment, while still being able to plan quickly. The coordination strategy that they proposed was to have the robots plan their paths using the paths of the other robots in the environment as obstacles to avoid. To resolve the problem of what order the robots replan and update their paths, the robot that has the highest potential gain from replanning is allowed to update its path.

Another approach to planning with multiple robots working in the environment was proposed by Saha and Isto in [28]. They proposed a decoupled path planning method for multi-robot environments. They propose an algorithm called MRP-IC which sequentially creates a path for each robot in the environment. As the next robot's path is generated, the previous robot's paths are treated as known mobile obstacles, which the planner then avoids like stationary obstacles. The planner also adjusts the speed of the previous robot's motion along their path to help find a collision free path.

Our approach to collision avoidance between robots differs from the previously mentioned techniques, because instead of coordinating the the interactions between the robots at the time that the robot generates the path from its current location to the goal location, we instead only perform the coordination when the robot's are within a certain distance of each other, where there is a high likelihood of collisions. At that point we use the bully algorithm [10] to determine which robot is allowed to move out of the collision

range first.

In [7], Dou, *et. al*, proposed a method for multi-robot path planning and collision avoidance through an extension of artificial bee colony (ABC), which generates paths for the robots through use of three phases for each robot. Scout robots which explore for better food sources, worker robots which travel to the food sources, and onlooker robots waiting for better sources of food to be found. Their algorithm, called improved artificial bee colony (IABC), extends ABC to simplify the setting of parameters and improve its performance. In our approach, the path planning between the robots does not have exploration phase that ABC and IABC contain. Our approach also handles collisions due to robots being near each other by stopping one all but one robot from moving until the moving robot is free of collisions.

Chapter 3

Simultaneous Task and Motion Planning

Problem Formation

We consider a set of wheeled mobile robots, R , deployed within a bounded environment. Robots are capable of localizing themselves within the environment and can also communicate wirelessly with each other. The environment contains a set of tasks, T , which correspond to a set of spatially distributed, distinct locations in the environment. Robots have to visit the locations of tasks to perform operations required to complete the tasks. Each task can require visits by one or more robots to get completed; the number of robots required to complete a task is provided *a priori* to the robots. We consider the case where task execution time is negligible, such that visiting the location of the task corresponds to completing the task. We also consider tasks that are loosely coupled and all robots required to complete a task do not necessarily need to visit the task's location at the same time.

Each robot is initially aware of the locations of the tasks, but does not know the exact paths between the tasks¹ nor the obstacles along those paths. The robots contain sensors necessary to perceive the portion of the environment near to the robot and determine where obstacles are located.

¹In the rest of this thesis, we have referred to task locations as tasks for legibility.

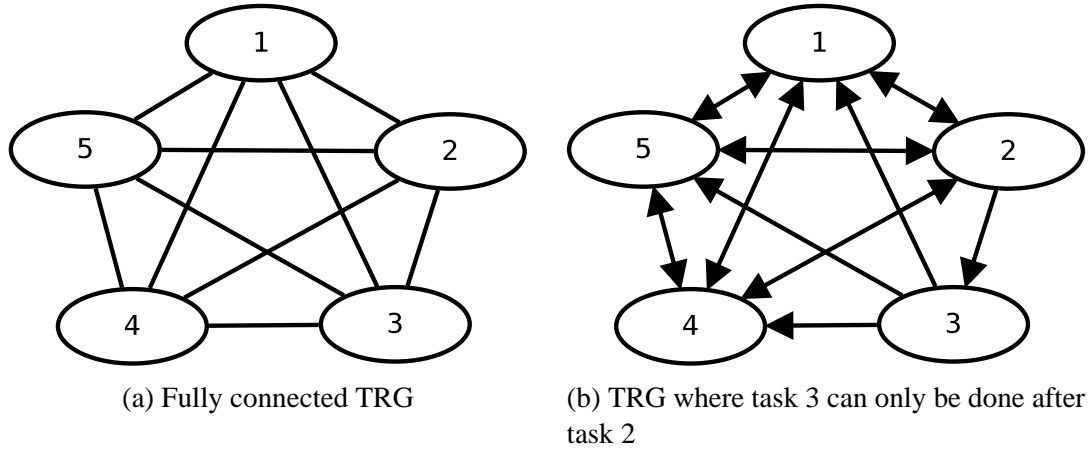


Figure 3.1: Example TRG

We first define the concept of the availability of a task for a robot to perform it. A task is available for performing if it is a good choice for the robot to perform that task next. A task can either be available for performing, or not available. It is not possible for the robot to directly observe the location of other robots, the paths they will take, and what tasks they plan to accomplish, this makes it difficult, if not impossible for the robot to directly observe the availability of a task; instead, we use a probability that reflects the current belief in the task being available for performing. The main cause of a task being unavailable is that other robots have, or will shortly, complete the task before this robot could complete it. This is related to the length of the path, and the uncertainty in the path costs due to changes in the paths as static obstacles are detected, or mobile obstacles move into the path of the robot.

We define an abstraction called a task reachability graph (TRG) to represent the tasks' spatio-temporal distribution. The TRG is a graph where the vertices correspond to the tasks that the robot needs to visit along with the robot's current location. The TRG edges correspond to the reachability between vertices such that there is an edge between two vertices if it is possible to travel between their corresponding tasks. Each edge in the TRG is weighted with a tuple of two values. The first value represents the the path cost or energy expended by the robot to travel between tasks. Because the environment is unknown, the edge weights in the TRG (path costs) are only approximate, and they get updated as the

robot discovers obstacles while traveling between tasks. To represent the uncertainty in path costs and in the robots ability to choose the best task to go to, the second value is the probability of the edge being available to complete. These values will also change as the robot gathers more information about the environment. For example, Figure 3.1(a) shows a TRG where all five tasks can be reached from one another. However, this does not strictly need to be the case, as can be seen in Figure 3.1(b), where task three can only be completed after task two has been completed, because that is the only edge that enters task three.

Due to the dynamic nature of the environment, it is important to define the TRG to be variable over time. Vertices can be added to the TRG due to the discovery of other tasks that must be completed by the robots. Likewise, vertices can also be removed when the task no longer needs to be done by that robot. As vertices are added or removed, edges will also need to be added or removed from the TRG. The initial estimates of the path lengths are likely to be inaccurate due to the limited information that the robot has about the location and geometry of obstacles in the environment. As the robot moves through the environment following the edges in the TRG, it is likely to update its map of the environment and become more certain of the layout of the obstacles in the environment. Because of this, the path length between tasks can change over time. Likewise, the availability of an edge can change as the robot gets more information about the environment.

Formally, let $TRG = (V, E, C, P, t)$ denote a fully connected graph where $V^{(t)} = \{v_i^{(t)} \cup v_{curr}\}$ is the vertex set and v_{curr} is the robot's current location, $E^{(t)} = \{e_{ij}^{(t)} : e_{ij}^{(t)} = (v_i^{(t)}, v_j^{(t)})\}$ is the edge set, $C^{(t)} = \{c_{ij}^{(t)}\}$ is the cost expended by a robot to traverse the path underlying edge $e_{ij}^{(t)}$, $P^{(t)} = \{p_{ij}^{(t)}\}$ is the probability corresponding to the availability of edge $e_{ij}^{(t)}$ and t is a time parameter. Let $\omega = (v^1, v^2, \dots)$ denote a single possible path through the TRG, and let Ω be the set of all possible paths ω through the TRG. We can then define the schedule $S : V \rightarrow \Omega$ as a function that returns an ordering over the set of tasks starting at a given node in the TRG. Each robot maintains its own copy of the TRG and plans its path using its local TRG. The problem facing each robot is specified by the Task

Ordering under Path Uncertainty (TOP-U) problem given below.

TOP-U Problem. Given $TRG = (V, E, C, P, t)$ representing the set of tasks, task reachabilities and inter-task costs at time t , determine a schedule $S^*(V)^{(t)}$ that induces an ordering $(v^1, v^2, v^3 \dots)$ over the tasks, given by:

$$S^*(V)^{(t)} = \arg \min_{\omega \in \Omega} \sum_{(v_i^{(t)}, v_{i+1}^{(t)}) \in \omega} (1 - p_{i,i+1}^{(t)}) c_{i,i+1}^{(t)} \quad (3.1)$$

$S^*(V)^{(t)}$ represents the path through the TRG with the best cost weighted with availability. Because maximum availability of an edge is identical to its minimum unavailability, we have considered the latter to solve for $S^*(V)^{(t)}$ as a minimization problem. An instance of the TOP-U problem corresponds to the well-known traveling salesman problem (TSP) [5]. However, finding an optimal solution to conventional TSP a known NP-hard problem [12]; also, an optimal solution to the corresponding TSP may not guarantee an optimal path for the robot as edge availabilities and costs (p_{ij} -s and c_{ij} -s) can change dynamically while the robot is traversing an edge.

In this thesis, we only consider cases where the cost to navigate between tasks is symmetric and does not depend on which direction between tasks the robot has to navigate. This allows us to consider cases where the TRG is undirected. We also only consider a connected environment where every task is reachable from all other tasks, and

there are no temporal dependencies between tasks, meaning that the TRG is fully connected. However, due to the dynamic nature of the TRG, the robots require methods to determine estimates of edge costs and availabilities as the robot gathers more information about the reachability between tasks as it explores the environment. Then the robot requires a method to utilise this information to determine the best task to navigate towards. To do this, we first propose a method to update the edge costs and probabilities based on sensor data from the robot. Then we use the most recent estimates for the cost and availabilities within an MDP-based framework to determine what the next task the robot should travel to

will be. In the rest of this thesis, for legibility, we have omitted the time notation from the TRG parameters, assuming it to be understood from context.

Chapter 4

Techniques for Solving Simultaneous Task and Motion Planning Problem

In the previous chapter, we proposed the Task Reachability Graph (TRG) as a means of encoding the data about the distribution of tasks and their availability. In this chapter, we propose a technique to use the TRG in finding a task plan that minimizes the expected cost of the robot to travel between all tasks. Figure 4.1 shows the general schematic of our proposed technique. Our technique is composed of two layers, the top layer is a task planner, which generates the order in which the tasks should be performed by the robot. This layer communicates with the bottom layer, which is a motion planner. The motion planner generates paths for the robot to get between the tasks and controls the robot's motion in the environment.

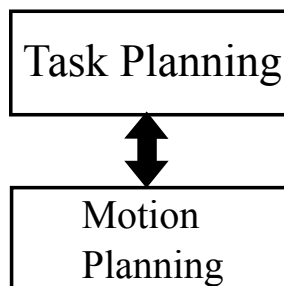


Figure 4.1: Structure of proposed technique

Each of the two layers shares information that allows the robot to always make movements towards the task that best minimizes the expected cost to complete all tasks. The task planner shares with the motion planner the task to move towards, and the set of all tasks, which allows the motion planner to monitor the paths between the tasks. In return, the motion planner shares with the task planner updates to the path lengths so that the task planner can always evaluate the best task to work towards completing.

4.1 Updating Edge Costs

The motion planning layer handles the updating of the path cost portion of the edge weights in the TRG. The motion planner uses a state-of-the-art probabilistic roadmap planner (PRM) that can handle uncertainty in paths [24]. Obstacles are provided to the planner in the form of a sequence of points that correspond to an estimate of the obstacles' bounding polygon's vertices. This sequence of points defines what the authors refer to as the "nominal" bounds of an obstacle. Each vertex is then assigned a probability distribution over which the obstacle's vertex may actually lie in the environment. The planner builds a roadmap by first generating a set of sampled points from the configuration space. Any samples that have a 50% or greater probability of colliding with an obstacle are discarded; this is equivalent to any point that lies within the "nominal" bounds of the obstacle. The remaining samples are kept based on the probability that they are located inside an obstacle, with a higher probability of collision resulting in a lower chance that the planner will keep that sample. The next step is to build a roadmap based on the retained sampled points. For each vertex in the roadmap, the nearest k samples, less than some distance d are selected, then the planner attempts to connect the two points with a straight line path. This path is rejected if it is not possible to connect the two points without intersecting the "nominal" bounds of an obstacle.

For each edge connecting two points (ρ_1 and ρ_2) in the roadmap, a probability $p_{\rho_1, \rho_2}^{coll}$

is assigned that represents the likelihood of colliding with an obstacle while navigating that edge. This probability is based on the distance to obstacles, and the certainty that the obstacle location is known. Based on this, the cost of a single edge in the roadmap can be calculated by:

$$cost_{\rho_1, \rho_2} = p_{\rho_1, \rho_2}^{coll} penalty + (1 - p_{\rho_1, \rho_2}^{coll}) dist(\rho_1, \rho_2)$$

where *penalty* is an arbitrary large number, used to discourage paths that are likely to collide with an obstacle and $dist(\rho_1, \rho_2)$ is the Euclidean distance between ρ_1 and ρ_2 .

A path ρ_{ij} in the roadmap between two points ρ_i and ρ_j is a sequence of points $\rho_{ij} = (\rho_i^{(0)}, \rho_a^{(1)}, \dots, \rho_j^{(n)})$ where there is an edge in the roadmap between any two adjacent points in the sequence. We denote the set of all possible paths in the roadmap as Φ . Given this, we can define the best path between ρ_i and ρ_j as:

$$\rho_{ij} = \arg \min_{\rho_{ij} \in \Phi} \sum_{\rho^{(a)} \in \rho_{ij}} cost_{\rho^{(a)}, \rho^{(a+1)}} \quad (4.1)$$

which determines the minimum cost path in the roadmap between points ρ_i and ρ_j [24].

To find the path corresponding to edge e_{ij} in the TRG, we add the points corresponding to vertices v_i and v_j into the roadmap as points ρ_i and ρ_j respectively. The path returned by Equation 4.1 is saved for future use and used to calculate the cost of edge e_{ij} . The cost is given by:

$$c_{ij} = \sum_{\rho^{(a)} \in \rho_{ij}} dist(\rho^a, \rho^{a+1}) \quad (4.2)$$

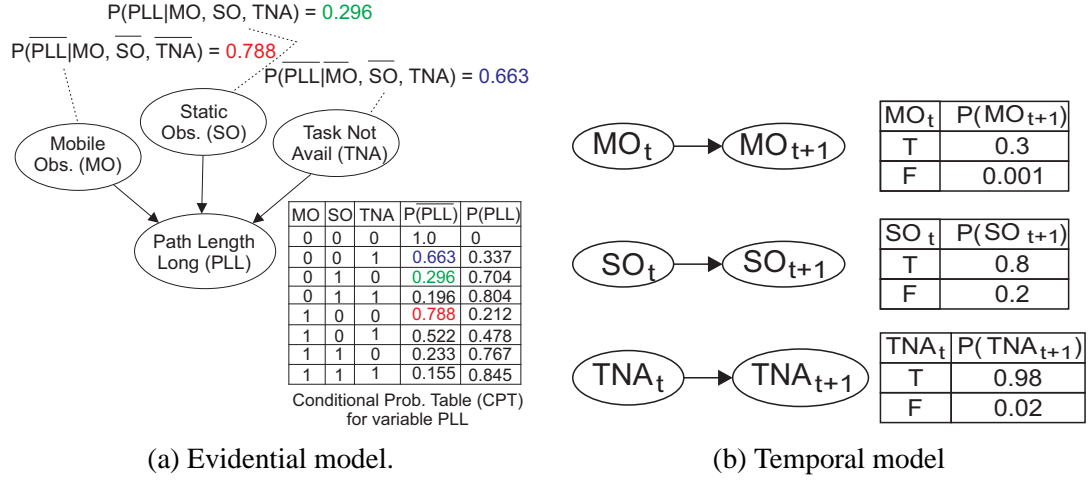


Figure 4.2: Bayesian Network used in the HMM for determining the suitability of path length to a task.

4.2 Updating Edge Availabilities

Using the updated edge costs from the motion planning layer, the task planning layer can calculate and update the probability that an edge in the TRG is available. It is not possible to directly observe the availability of an edge; however, it is possible to observe the path lengths of each edge in the TRG and compare it to other edges which share a node. There are multiple things which can effect the path length, which are:

- Mobile obstacles (MO)
- Static obstacles (SO)
- Task not available (TNA)

A mobile obstacle can either be along the path, or it can not be, however this fact cannot be directly observed, this means that we can best represent a mobile obstacle as a Boolean random variable. Also a mobile obstacle can be there at one time and not at the next time step. A similar argument can be made for both static obstacles and task not available. For this reason, we have modeled each of these as time dependent Boolean random variables. These variables all effect the path length, which we have decided to

discretize into a Boolean random variable. A path is considered too long if its length is above some threshold, and not too long if it is below the same threshold.

To reflect the dependencies between these events, and their influence on the path length, we have defined a Temporal Bayesian Network (TBN) [26] for each edge in the TRG, shown in Figure 4.2. Let TBN_{ij} denote the Temporal Bayesian Network corresponding to edge $e_{ij} \in E$. A TBN is a network of random variables which effect each other, an edge between two nodes represents that one node has an effect on the other. In Figure 4.2(a), mobile obstacle, static obstacle, and task not available, all effect the path length becoming too long, however the effect of these states are not deterministic, but are instead probabilistic. In other words, there is a probability of the path length being too long given the state of the world, e.g. $p(PLL|\overline{MO}, \overline{SO}, TNA)$ denotes the probability that the path length is too long given that there are no mobile obstacles, no static obstacles, but the task is unavailable. Given the probabilities associated with each edge in the TBN, it is possible to calculate the probability of TNA given the sequence of observations seen thus far.

To determine the effects on PLL given the state of the world, we first assumed that the inhibition probabilities of each state affecting PLL is independent. In other words, what ever prevents a mobile obstacle from causing the path length to be too long is independent of what ever prevents the task not being available from causing the path length to become too long. With this assumption we can use the Noisy-OR relationship [26] to build the probability tables shown in Figure 4.2(a). The three probabilities shown in red, green, and blue in Figure 4.2(a) are the inhibition probabilities, which were determined through experience of navigating robots through mobile and static obstacles in different environments. From these three probabilities it is possible to calculate the rest of the table.

Because mobile obstacles, static obstacles, and the availability of tasks evolve over time, a second set of probabilities are required, that capture this change. These probabilities are shown in Figure 4.2(b) and were determined through experience with robotic systems.

Each variable is only dependent on its previous state, but not on any state previous to that. For a mobile obstacle, its position in the next time step can be anywhere within a circle whose center is at the mobile obstacle's current location, with radius related to the speed it travels at and the amount of time between time steps. The detection of a static obstacle can change over time as the robot is able to get a better view of the area that the obstacle may be located at. Also a task that was unavailable for completion may become available for completion due to the changing perception of the environment. For example, due to the independence of the inhibition probabilities, we can write: $P(\overline{PLL}|MO, SO, \overline{TNA}) = P(\overline{PLL}|MO, \overline{SO}, \overline{TNA}) + P(\overline{PLL}|\overline{MO}, SO, \overline{TNA})$ and in a similar manner, the rest of the table can be calculated.

To determine the value of the observation PLL_{ij} , the path length of the current edge e_{ij} is compared to a threshold PLL_{thr} . Due to the wide array of environments that the robots could be deployed in, PLL_{thr} can not be set to a single value for all environments. Also, due to the variability inside the environment, a single threshold for the full environment may not work well either. However, it can be set in relation to the other information that the robot has available. To handle this, we define a separate threshold, $PLL_{thr_{ij}}$, for each edge e_{ij} . This threshold is defined as the minimum cost edge that shares the same starting vertex. This is shown below in Equation 4.3 below.

$$PLL_{thr_{ij}} = \min(\{c_{ik} : \forall k \in V\}) \quad (4.3)$$

To then determine the value of the observation PLL_{ij} , the following equation is used:

$$PLL_{ij} = \begin{cases} \text{FALSE} & \text{if } c_{ij} \leq \Gamma PLL_{thr_{ij}} \\ \text{TRUE} & \text{otherwise} \end{cases} \quad (4.4)$$

where Γ is a user defined constant that describes how much larger c_{ij} must be before we consider it to be too large.

In other words, the threshold used to determine the value of PLL_{ij} is based on the shortest path currently known leaving the vertex in question. The user-defined parameter Γ is used to determine how much longer than the current shortest path we allow before we decide that the path length is long enough that the task should no longer be considered available for immediate completion. Γ helps to prevent the robot from always selecting the closest task to go to, like a greedy approach would.

The node in the TBN that corresponds to the edge availability is TNA , which represents the probability that the edge is not available. To determine this probability, the robot generates observations of each edge e_{ij} in the environment, denoted $PLL_{ij}^{(1...t)}$. These observations are then used in the TBN to determine the probability of TNA_{ij} given the current observations. Stated mathematically, $p_{ij} = P(TNA_{ij} | PLL_{ij}^{(1...t)})$. The equation is solved using the Forward-Backward algorithm [26].

Directly solving a TBN can be a difficult task, to reduce this difficulty, we use an evaluation method called a Hidden Markov Model (HMM) [26]. The HMM improves computation time by modeling the TBN as a first order Markov model, where the states of the Markov model are combination of the state variables (MO, SO, TNA) of the TBN. As time progresses, the world transitions between theses states based on a transition probability matrix. When at each state, the world probabilistically emits an observation with is the combination of the evidence variables (PLL) of the TBN.

4.3 TOP-U Solution using Markov Decision Process

The environment that the robot is working in changes stochastically, and the robots have to make decisions inside this environment. Due to these properties, a Markov Decision Process (MDP) [26] is a good fit. In general, a MDP consists of a sequence of potential states that the decision maker (in our case a robot), can be in. At each state, the robot has a set of decisions that it can make. Each potential decision will change the state that the

robot is in, however, the exact change that is made is not deterministic. This means that if the robot is in a state A and performs an action b , it is not guaranteed to always arrive in state B , it may instead, sometimes, arrive in some other state C . The exact method that the states change is described by a set of probabilities that gives the likelihood of reaching a specified state given that robot starts in one state and performs an action. Each state also has an associated reward that is given to the robot for reaching that state. The robot's goal is to maximize its reward by performing actions that maximize the expected reward.

More formally, an MDP is a stochastically evolving process defined by four parameters $\langle S, A, T, R \rangle$. Where S is the set of states that the agent (robot) making decisions can be in. A is the set of actions, or decisions, that the robot can make at each state. Based on the action performed, the robot will transition from its current state to a new state, the transition of the robot is controlled by the transition function $T : S \times A \times S \rightarrow [0, 1]$, which gives the probability of transitioning from state s to a state s' by performing action a . When the robot reaches a state, it is provided a reward given by $R : S \rightarrow \mathbb{R}$. Based on these parameters, a policy $\pi^* : S \rightarrow A$, is found which prescribes the best action to perform at each state which maximizes the expected cumulative reward to the robot.

There are many similarities between the general MDP and the problem we are investigating. As MDPs can be used for path planning, we can draw the parallel between planning a path in the configuration space and planning a path in the task space. To do this, we convert our TRG into an MDP, where the MDP's state are the vertices of the TRG graph. At each vertex, the robot has to make a decision to follow any of the edges leaving that vertex. The probability of transitioning between states is analogous to the probability that an edge is available, where the robot is less likely to successfully navigate between the two vertices in the TRG if it is very unlikely that the task is available. However, because the edge availabilities are coming from separate TBNs, the raw edge availabilities do not sum to one, but since the robot must transition from the state to another state in the TRG, all probabilities have to sum to one. To solve this, the edge availabilities are normalized

using the equation, $p_{ij} = \frac{p(TNA_{ij}|PLL(1...t))}{\sum_k p(TNA_{ik}|PLL(1...t))}$. In our TRG, we do not have the concept of reward, but we do have a similar concept, in the edge cost. Because maximizing reward and minimizing cost are similar, we have decided to solve the MDP as a cost minimization problem. When we find a policy in the MDP, we are now finding the best task to go to after completing our current task that will minimize the expected cost to the robot.

4.4 Robot Navigation and Task Selection

```

1 selectTask(TRG = < V, E, P, C >)
   Input: TRG: task reachability graph
2 Initialize MDP with current TRG information
3 Determine paths in robots configuration space using PRM planner between all TRG
  edges  $e_{ij} = (v_i, v_j) \in E$ 
4 while true do
5    $v' \leftarrow \pi^*(v_{curr})$ 
6    $path \leftarrow$  PRM path between  $v_{curr}$  and  $v'$ 
7   while  $v'$  not reached do
8      $v_{curr} \leftarrow$  current position of robot
9     Broadcast  $v_{curr}$  to other robots
10     $coordinatePath(v')$  //avoid collisions with nearby robots, if any (Alg. 3)
11     $(v', path) \leftarrow$  updateTRG(TRG,  $v'$ )
12    if  $v' = null$  then
13      return
14    end
15    Move along current segment of  $path$ 
16  end
17  Remove  $v'$  from  $V$  //reached  $v'$ 
18 end

```

Algorithm 1: Algorithm to select a task in the TRG using an MDP-based policy.

The technique used by a robot for selecting tasks to visit using our TRG and MDP-based framework is shown in Algorithm 1. The MDP is initialized with the parameters from the TRG and the navigation paths in the environment between every pair of TRG vertices are calculated using the PRM planner (line 2 – 3). While the robot is aware of tasks that it needs to visit, it calculates the next task (TRG vertex), v' , to visit using the MDP policy and gets the PRM path to v' (line 5 – 6). Because each robot calculates its

```

1 updateTRG( $TRG = \langle V, E, P, C \rangle, v'$ )
   Input:  $TRG$ : task reachability graph;  $v'$ : destination TRG vertex
   Output:  $v$ : destination TRG vertex,  $path$ : path to destination TRG vertex
2 if any task got completed by other robots (received via communication) OR (new
   obstacle in collision with robot) then
3   update  $V$ 
4    $path \leftarrow$  replan path from  $v_{curr}$  to  $v'$  using PRM-planner
5    $\forall v \in V$ , update  $c_{v_{curr},v}$  from path planner data
6    $\forall v \in V$ , Update  $p_{v_{curr},v}$  from HMM using sensor data
7   Update MDP, TRG with new values of  $V$  and  $p_{v_{curr},v}$  and  $c_{v_{curr},v} \forall v \in V$ 
8    $v_{new} \leftarrow \pi^*(v_{curr})$  //calculate policy given by updated MDP (task replan)
9   if  $v_{new} = \{\emptyset\}$  then
10    return null; // No more tasks
11  end
12  if  $v_{new} \neq v'$  then
13     $v' \leftarrow v_{new}$ ; // Switch tasks
14     $path \leftarrow$  PRM path between  $v_{curr}$  and  $v'$ 
15  end
16 end
17 return  $v', path$ 

```

Algorithm 2: Algorithm to update TRG and path when TRG vertices are removed (task completer) or a new obstacle is detected that triggers a path re-calculation.

navigation plan independently in its own local configuration space, multiple robots might calculate paths that intersect with each other and might lead to a collision, especially when the robots are in close proximity. To address this problem, each robot broadcasts its current location to other robots and coordinates its path to avoid collisions with nearby robots using the *coordinatePath* algorithm (Algorithm 3) (lines 9 – 10). The robot then checks to see if the TRG needs to be updated using Algorithm 2 (line 11). If the updated TRG returns a null destination vertex indicating that there are no more tasks for the robot to visit (all other tasks have been completed by other robots), the robot stops navigation (lines 12 – 14). Otherwise, it moves towards its current destination TRG vertex v' along the next path segment of the PRM planner prescribed navigation path. When the robot reaches v' , it removes v' and its associated edges from the TRG (line 17) and proceeds to select the next TRG vertex to visit.

The algorithm used to update the TRG using the HMM updates and resulting MDP

policy updates is shown in Algorithm 2. If a robot receives communication from another robot informing that it has completed a task corresponding to a TRG vertex, the vertex is removed from the robot’s TRG. When a robot’s TRG vertex set changes in this manner or when the robot perceives a new obstacle on its sensor that is in collision with its navigation path, it updates the TRG vertex set, and calculates a new navigation path to its destination vertex v' (lines 2 – 4). The new navigation path from the PRM planner is used to update the TRG edge costs, while the HMM updates the TRG edge availabilities (line 5 – 6). These updated values are incorporated into the MDP representing the TRG and the MDP’s policy is recalculated to yield the new destination vertex (line 7 – 8). If the recalculated policy prescribes a new target vertex, $v_{new} \neq v'$, due to increased path costs of reaching v' , then the robot performs a task switch and changes its destination from v' to v_{new} (line 12 – 14). Note that due to the unknown nature of the obstacles between tasks, a robot might receive communication that another robot completed the task that it was heading towards. In the extreme case, all tasks in a robot’s TRG might be completed by other robots before its reaches those tasks (lines 9 – 11) (as in Figure 1.1(b) for robot 2); the `updateTRG` method returns a null vertex for this case.

4.5 Coordinating paths between robots to avoid collisions

If robots determine their paths individually using the PRM-based planner, it could lead to robot collisions when the planned paths of two or more robots intersect with each other. To avoid this scenario, we have used a collision avoidance algorithm shown in Algorithm 3. Each robot uses the locations broadcast by other robots to check if there are other robots within a radius of r_{coll} , called the collision circle, of itself (lines 2). When a set of robots are within the collision circle of each other, all the robots stop and the robots exchange their identifiers, representing their priorities, with each other. A leader election algorithm called the bully algorithm [10] is then used to select the robot with the highest priority as

```

1 coordinatePath
   Input:  $v'$ : destination TRG vertex
2 while another robot within  $r_{coll}$  do
3   stop
4   send priority to all other robots within  $r_{coll}$ 
5   //priority is either robot id or  $\infty$ 
6   calculate winner using bully algorithm [10]
7   winner robot holds winner token
8   if I am winner then
9      $path \leftarrow$  replan path from  $v_{curr}$  to  $v'$  using PRM planner
10    //other robots considered as static obstacles in PRM
11    if path to  $v'$  not found (another robot from collision circle stopped at  $v'$ ) then
12      set prio  $\leftarrow \infty$ 
13    else
14       $(v', path) \leftarrow$  updateTRG( $TRG, v'$ ) if  $v' = null$  then
15        return
16      end
17      Move along current segment of  $path$  until outside collision circle
18    end
19    release winner token
20  end
21 end
22 else
23   while winner token not released do
24     stop
25   end
26 end
27 end

```

Algorithm 3: Algorithm to avoid collisions between robots in close proximity of each other.

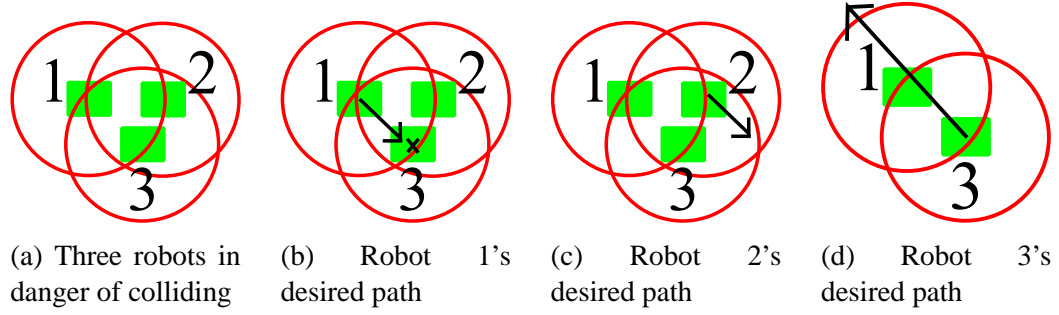


Figure 4.3: Example collision avoidance

the winner. The winner robot holds the winner token, which gives it the right to move (lines 3 – 7). All other robots in the collision circle, which do not hold the winner token, remain stationary (line 24 – 26). The winner robot uses the PRM planner in conjunction with updating the TRG using Algorithm 2 to find a path to its destination vertex v' . The path returned by the PRM planner is executed and the moving robot releases the winner token once it is outside its collision circle (lines 9, 14 – 22). If the PRM planner is not able to find a path to the goal, e.g., if the goal is unreachable because there is another robot within the collision circle that is stopped right at the goal location, the moving robot relinquishes its right to move by setting its priority to a high value (∞) (lines 11 – 12). Another robot from within the set of stopped robots gets a chance to run the bully algorithm and attempts to move. This protocol ensures that at least one robot exits the collision circle with each execution of the bully algorithm, and finally there is only one robot left inside the collision circle. This robot then reverts to using its PRM-based planner to plan its path as part of Algorithm 1

To help further explain how the collision avoidance works, we will use an example, shown in Figure 4.3. Consider the case shown in Figure 4.3(a), where the green squares are robots, the numbers represent the robot's id number, and the red circles are each robot's collision circle of radius r_{coll} . As can be seen, we have three robots that are all within each other's collision circles, and as such the robot's have stopped and shared their robot id's with each other. Robot 1 is the robot with the lowest robot id, and thus it is allowed to

move first. It takes a reading of the environment and replans its paths assuming that the robots in front of it are static obstacles. However, its goal, as shown by the arrow and x in Figure 4.3(b), is the same location where robot 3 is located. This causes the path length returned from the path planner to be returned as infinity, because a path does not exist. As such, robot 1 notifies the other robot's that it is unavailable to move by setting its temporary priority to ∞ , which hands over the winners token to the other robots to determine which is the winner. Robot 2 is the robot with the next lowest id, as can be seen in Figure 4.3(c), its desired path leads away from the robots in collision. Thus after replanning, it is able to navigate away from the other robots and exit the other robot's collision circles. Once robot 2 is outside of the last robot's collision circle, it leaves the collision avoidance algorithm and resumes operations like normal.

After robot 2 has left robot 1 and robot 3's collision circles, they re-evaluate the bully algorithm and determine that robot 3 is the robot with the lowest id; its desired path can be seen in Figure 4.3(d). As it can plan around robot 1, it moves and leaves robot 1's collision circle, allowing all robots to resume following their original plans.

4.6 Example

To help further explain how our proposed solution works, we look at a simple example with two robots, named R1 and R2, and five tasks; each task requires only one robot to visit it to consider it completed. The starting locations of robots R1 and R2, location of tasks, and location of all obstacles in the environment is shown in Figure 4.4(a). As can be seen, there is a wall separating the two robots with a small doorway in the center. When the robots start, they are unaware of the location of other robots or obstacles in the environment, but are aware of the boundary of the environment. At the beginning of execution, the information that robot R1 knows about the environment is shown in Figure 4.4(b). At this point the robots generate their TRGs. Robot R1's TRG is shown in Figure 4.5(a).

Vertex 1	Vertex 2	Path length	Observation	Probability	Normalized
R	1	2.999	FALSE	0.530	0.200
R	2	4.067	FALSE	0.530	0.200
R	3	5.548	FALSE	0.530	0.200
R	4	5.654	FALSE	0.530	0.200
R	5	4.142	FALSE	0.530	0.200
1	2	2.342	FALSE	0.530	0.434
1	3	5.414	FALSE	0.530	0.434
1	4	7.073	TRUE	0.081	0.066
1	5	7.108	TRUE	0.081	0.066
2	3	3.355	FALSE	0.530	0.317
2	4	5.776	FALSE	0.530	0.317
2	5	7.554	TRUE	0.081	0.048
3	4	3.260	FALSE	0.530	0.317
3	5	7.314	TRUE	0.081	0.048
4	5	5.287	FALSE	0.530	0.434

Table 4.1: Table of PLL observations for each edge of TRG

Once the TRG has been generated, a path is planned for each edge in the TRG. Since the environment is empty, the planned path will closely follow the TRG edges. Once the paths are planned, each edge of the TRG is updated with the length of the resulting paths. Once the path lengths are found, it is possible to construct the observations for each of the edges based on Equation 4.4. These observations are used in the HMM shown in Figure 4.2 to generate a probability of availability for each TRG edge. The path lengths, observations, and resulting probabilities for the TRG constructed by robot R1 is shown in Table 4.1. As can be seen, the cases where the path length was more than three times that of the shortest path connected to that node, PLL was observed to be FALSE. For example, the shortest path leaving v_1 is to v_2 with a path length of 2.342, thus that is the value that $PLL_{thr_{12}}$ takes. The path between v_1 and v_4 is 7.073 which is greater than $3 * 2.342 = 7.026$, so the observation for PLL is set to TRUE. As can also be seen, the observation of TRUE causes the probability returned from the HMM to drop from 0.530 to 0.081 reflecting the change in belief of those paths being the best and likely candidates to be available for execution.

Once the TRG is completed, it is converted to an MDP, and a policy is found, from this

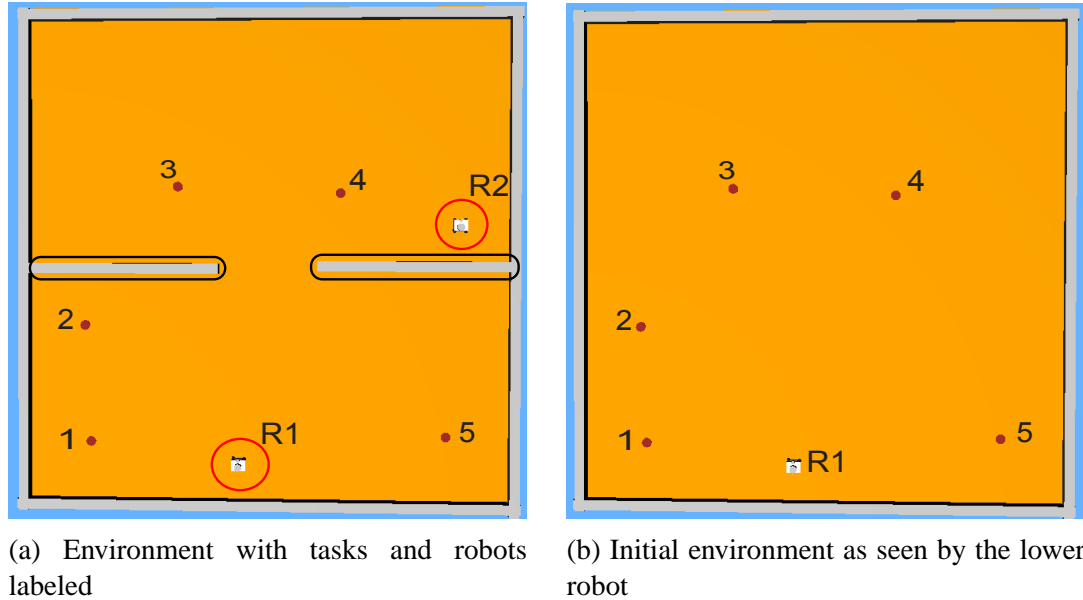


Figure 4.4: Example part 1, tasks are shown by red circles

the best task to go to is selected for each robot. Figure 4.5(b) shows the paths that the two robots will take.

Figure 4.6(a) shows the resulting TRG once both robots have completed their first task. As can be seen, the robots have communicated with each other, as the lower robot also knows that the upper robot has completed its task. At this time, the paths for each edge in the TRG is recomputed to account for any detected obstacles, there are none at this time and the edge weights of the TRG are updated. Figure 4.6(b), shows the paths that the robots will take to their second tasks. In this case, the robot R1 reaches its task before robot R2 does, this causes the task to be removed from robot R2's TRG. Robot R2 then replans due to the change in its TRG, it selects the same task because the removal of the task robot R1 completed did not affect robot R2's task. Robot R1 then replans, to select its new task, at this time it can see a portion of the wall that divides the two halves of the environment. So, the path planner plans a path around the obstacle, and the MDP selects the path and TRG edge shown in Figure 4.7(a).

As the robot R1 is following the path, the upper robot completes its task, causing the lower robot to have its path execution interrupted. This triggers a replan in both robots,

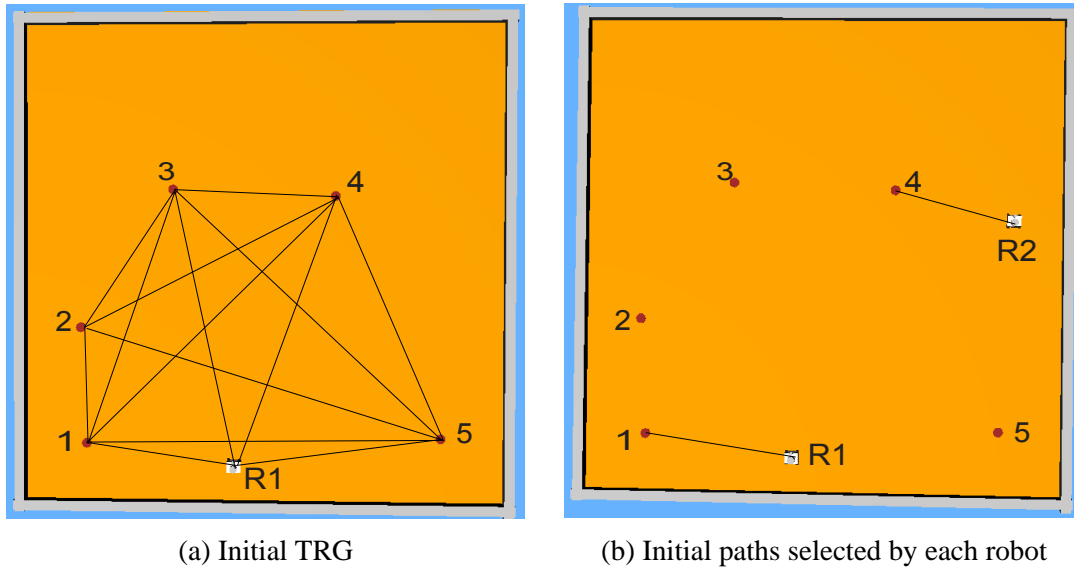


Figure 4.5: Example part 2, tasks are shown by red circles

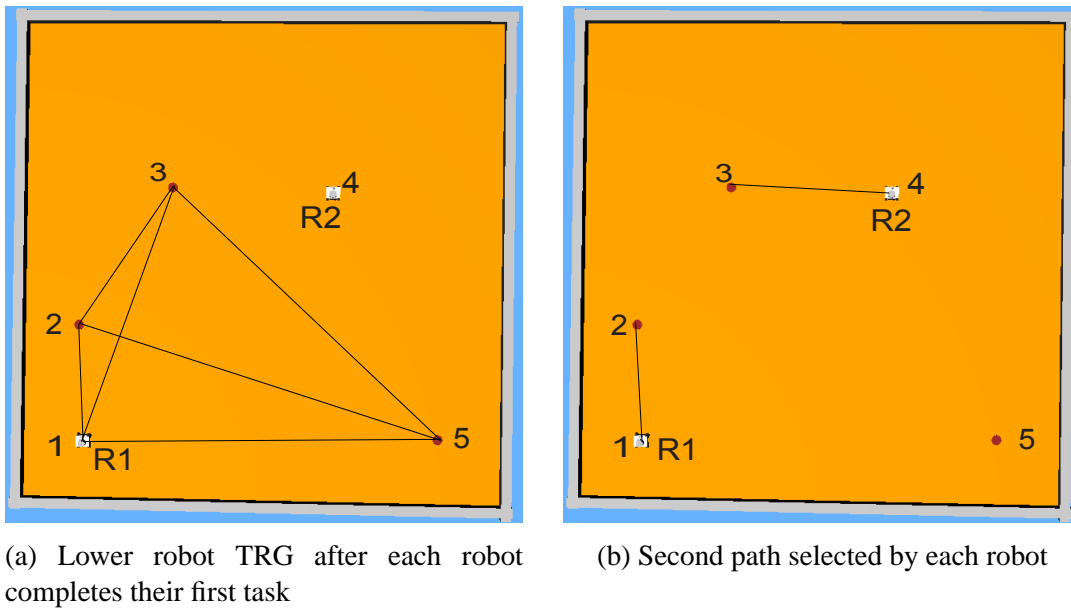


Figure 4.6: Example part 3, tasks are shown by red circles

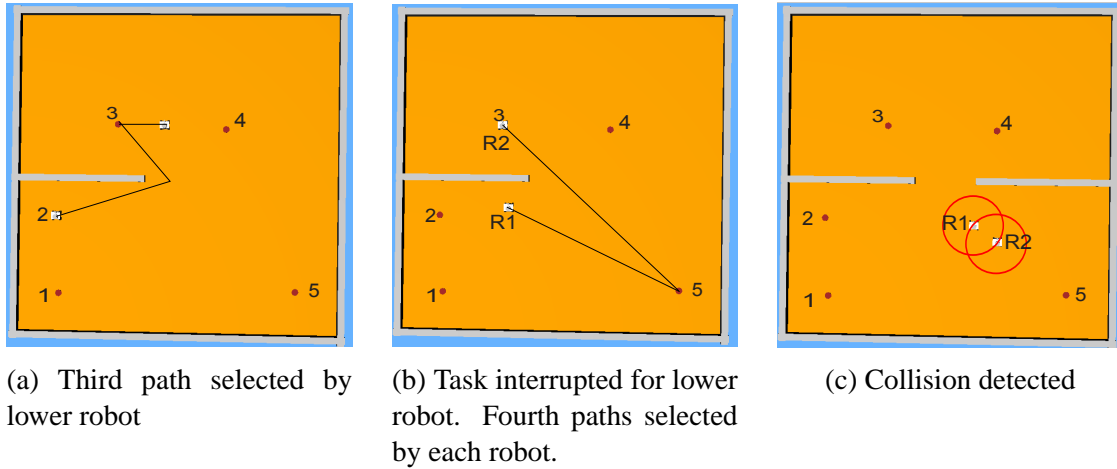


Figure 4.7: Example part 4, tasks are shown by red circles

where they both select the remaining task to complete, as seen in Figure 4.7(b).

As both robots navigate to the final task, their paths take them close together to the point where they enter each other's collision circles, as shown in Figure 4.7(c). At this time, the robots perform the collision avoidance algorithm which allows one of the two robots to resume navigation towards the goal, once they are outside each other's collision circles, both resume navigation towards the task, until one of the robots reaches it, at which time both robots will terminate their execution of Algorithm 1.

Chapter 5

Experimental Results

To verify our proposed approach, we performed two groups of experiments, the first was a set of simulations to determine the scalability and performance of the algorithm under different algorithm parameters, different numbers of robots, tasks and obstacles as well as different environment settings. The second set was using a set of hardware Coroware Corobot robots to test how well the approach performs in a real world setting. The following two sections describe in detail how the experiments were constructed, performed, and summarize the main experimental results.

5.1 Simulated Experiments

5.1.1 Setup

The simulated experiments were conducted using a simulated Coroware Corobot robot using the Webots robotics simulator. Webots is a commercial robotics simulator that allows for modeling robots, their sensors, and the environment they are in. It also provides a three dimensional view of the system as the experiments run. The simulator was run on a Intel Core 8-core i7 CPU running at 3.2 GHz machine running Ubuntu 12.04. The robot is equipped with:

- GPS
- Compass
- Laser distance sensor
- Wireless communications

The GPS and compass sensors together allows the robot to know its current location and orientation inside the environment. The laser distance sensor provides the robot with distance information to obstacles located in it's field of view, 270° centered in front of the robot. This information is used to plan around obstacles in the environment. And finally, the wireless communications allows the robots to exchange information with other robots in the environment.

The simulated robot can be seen in Figure 5.2(a). The environment was $10 \times 10 \text{ m}^2$ with obstacles placed at strategic locations in the environment. The obstacles were structured in such a way that in the single robot case, the robot would have to switch tasks 50% of the time if it was using a greedy closest-task-first task allocation method. The environment layouts used can be seen in Figures 5.1(a), 5.1(b), and 5.1(c). We setup the PRM planner to perform one sample for every 250cm^2 , with the penalty from Equation 4.1 $penalty = 1000$. It was also setup to connect the nearest ten neighbors. We also set $\Gamma = 1000$ in Equation 4.4.

The approach was compared to a greedy closest-task-first algorithm [34] modified to use the same PRM based path planner and coordination technique. These modifications were made to allow for a more direct comparison of the task allocation approaches without influences from the coordination or underlying path planner. Each robot selects the task that has the least cost on its TRG from its current location, without modeling uncertainties in the inter-task paths or updating TRG edge availabilities. At each time step, the greedy algorithm checks if its current location is still closest to the task it has selected, if it determines that the straight line distance between it's current location and another task

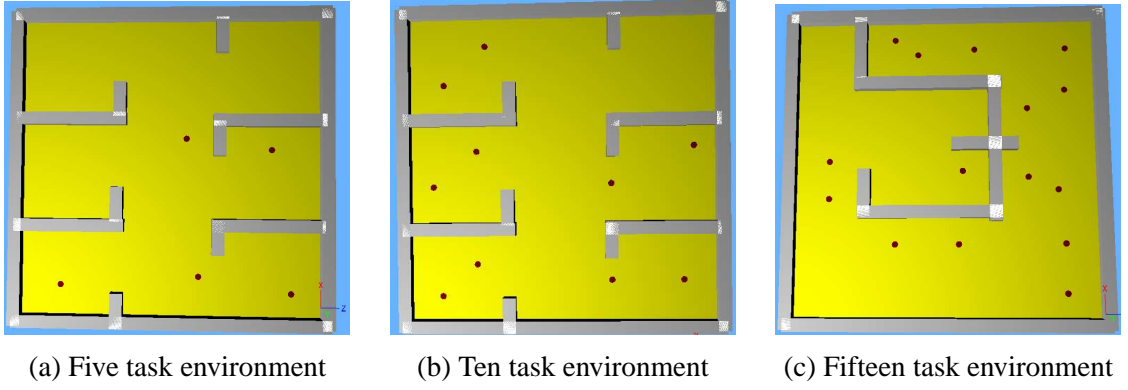


Figure 5.1: Simulation setup

is smaller than the straight line distance from the robot's current location to another task, it will switch and go to the closer task.

To test how the approach responds to a variety of environments we tested with number of robots $|R| = \{1, 3, 8\}$, number of tasks $|T| = \{5, 10, 15\}$, and number of visits that each task requires $Vis = \{1, 2, 3\}$. We also tested the effects of the user defined parameter Γ between 1.5 and 6 in steps of 0.5, and between 6 and 10 in steps of 1. To enable comparisons between each environment, we define a parameter L , called the task load. The task load is defined as:

$$L = \frac{T * Vis}{R}$$

Intuitively, the task load is a parameter that represents how much work the average robot needs to do. For example, if there are five tasks and five robots, and each task needs to be visited only once, then on average, each robot will visit only one task, or we can say that the setting of 5 tasks, 5 robots, and 1 visit has a load, $L = 1$.

From the experiments, we collected the following data:

- Distance traveled
- Number of replans resulting in a switch of tasks
- Number of replans not resulting in a switch of tasks

- Time taken to plan the paths
- Time taken to navigate selected paths

Each metric reveals a different aspect of the performance of our proposed approach. Distance traveled and navigation time are both common metrics that are used for optimization. Distance traveled is a good analog for energy used because the most energy intensive portion of robotics is normally moving the wheels. The time taken to navigate also provides an indication of how much energy the approach requires, but, unlike distance traveled, the time taken to navigate also gives us a measure of how much time was taken stopped in the collision circles of higher priority robots (Algorithm 3). The number of replans that result in a switching of tasks tells us how often the robot was going to what it considered to be the wrong task when one of the updateTRG conditions were triggered (Algorithm 2 line 2), while similarly, the number of replans that do not result in a switching of tasks tells us how often the same conditions were met but the robot was still going to the best task. This corresponds to cases where the robot found an obstacle along the path, but the obstacle was not large enough or in the correct position to make going to the previous task no longer the best task to go to. The time taken to plan paths provides information about how much of the total time was taken to find the paths between tasks and to select the best task to navigate towards.

5.1.2 Results

Figure 5.3(a) shows the number of replans made by each robot, resulting in task switches for the two algorithms. We observe that, on average, the TRG-based approach results in 61% less task switching than the greedy approach. Likewise, Figure 5.3(b) shows the number of replans made by each robot that did not result in a task switch. We can see that the TRG-based approach also results in 40% less replanning than the greedy approach. The reduced planning and task switching by the TRG-based algorithm can be attributed to its



(a) Simulated Corobot robot in Webots

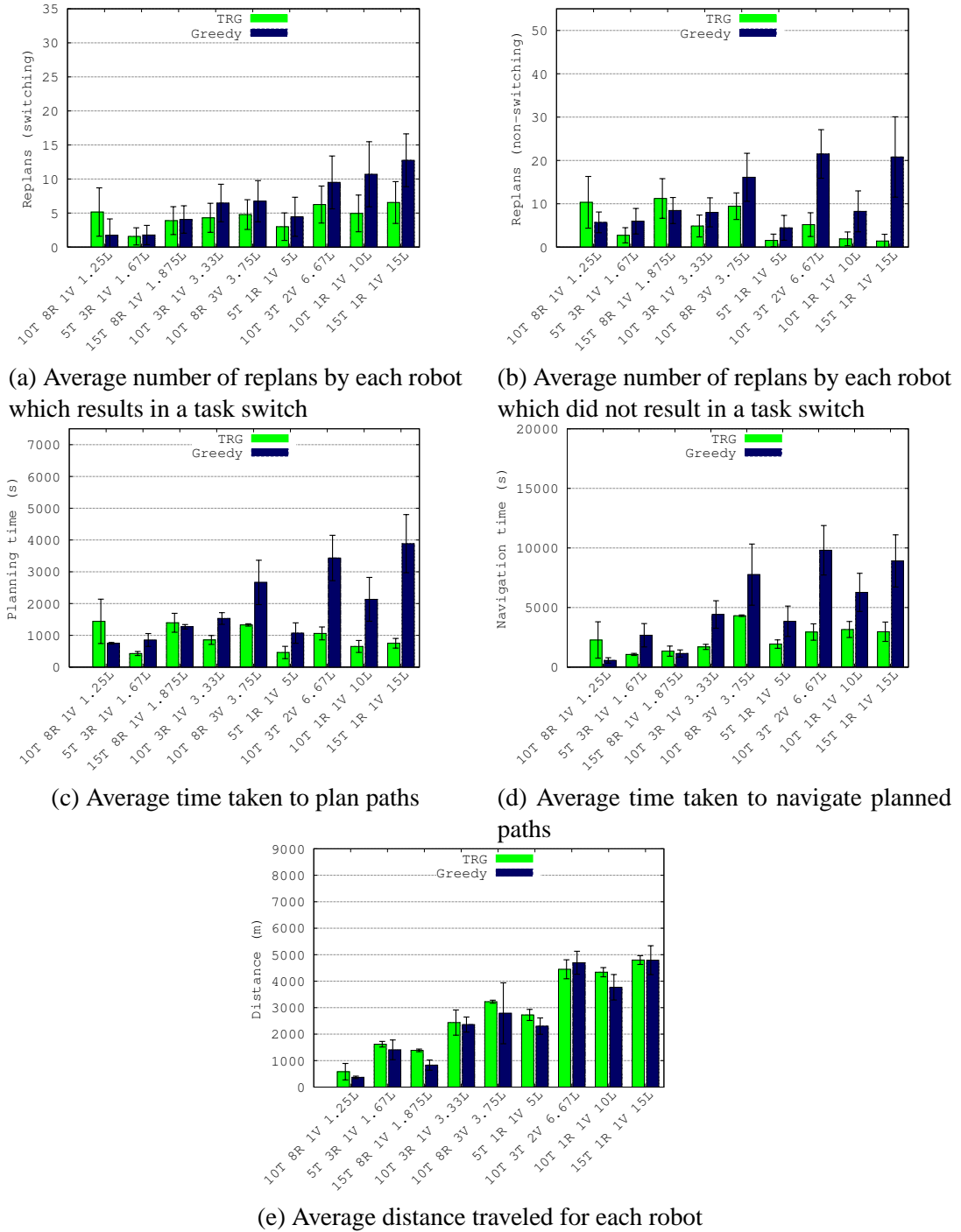


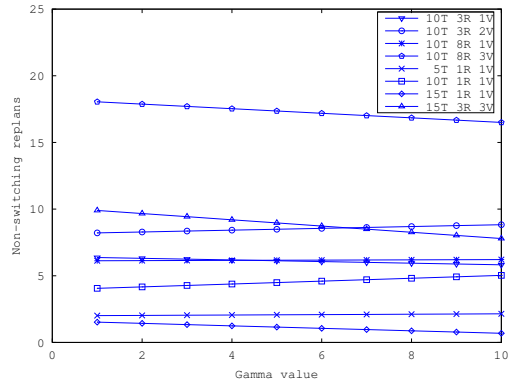
(b) Physical corobot robot

Figure 5.2: Robots used

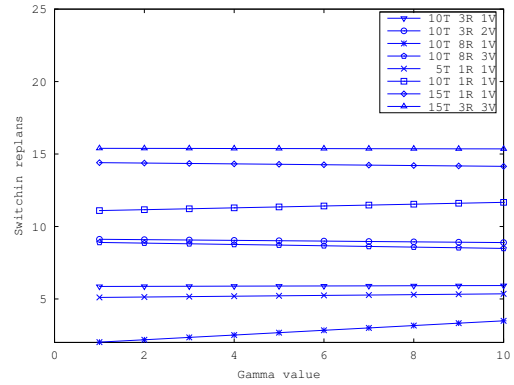
ability to reason more efficiently about task availabilities using its costs and beliefs about paths in the MDP based approach, along with real-time sensor data incorporated into its decisions using the HMM. In contrast, the greedy approach uses only Euclidean distances to select tasks and consequently performs poorly.

In Figure 5.3(c), we show the average time taken to plan the paths for both approaches. Figure 5.3(d) shows the time taken in locomotion, which includes time taken to handle collisions using Algorithm 3. The TRG-based approach takes much less time for both planning and navigation compared to the greedy approach. In this case, the TRG-based approach requires 60% less planning time, and, 58% lower locomotion and coordination times than the greedy approach. This is because the TRG-based approach accounts for both the known obstacles between tasks and the likelihood that the task will become unavailable. The greedy approach behaves myopically and selects the closest task to visit, which could be on the other side of a large obstacle and require considerable planning and locomotion times to reach. In contrast, the TRG-based approach uses the robots perception of the environment to weight the path costs to tasks with the corresponding path belief to reduce the overall path costs. Note that when the number of tasks is small, or the average task load per robot is close to 1, both algorithms have comparable performance for all three metrics as each robot has to visit only one task and there is no task ordering required.

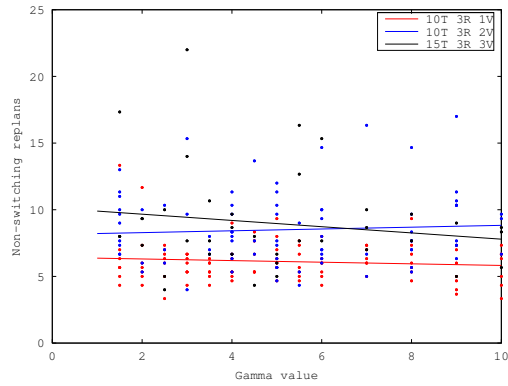
Figure 5.3: Simulation result $\Gamma = 1000$



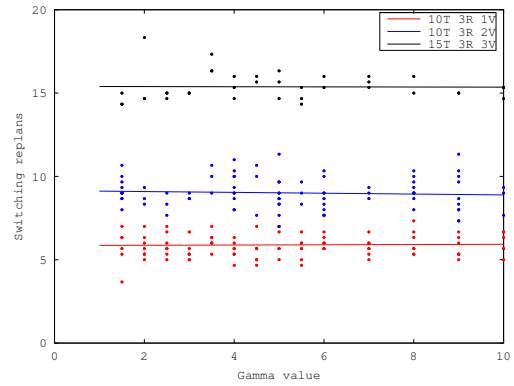
(a) Non-switching replan trend lines as Γ changes



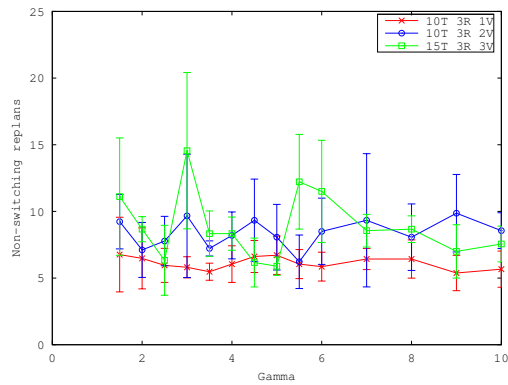
(b) Switching replans trend lines as Γ changes



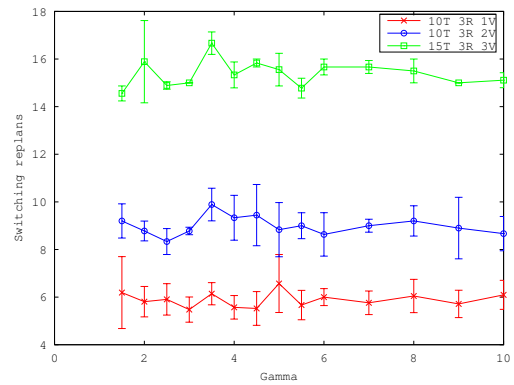
(c) Non-switching replans trend line and scatter plot as Γ changes for $Vis \geq 1$



(d) Switching replans trend line and scatter plot as Γ changes for $Vis \geq 1$

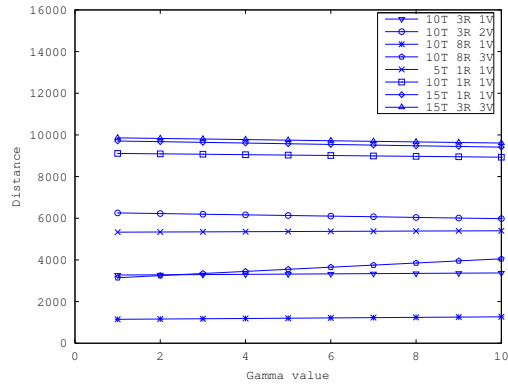
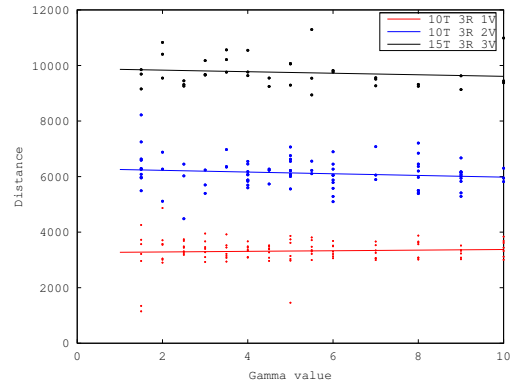
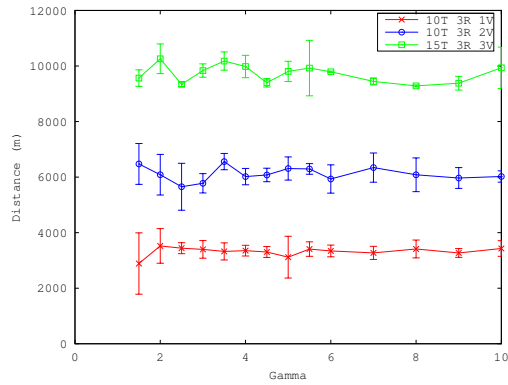
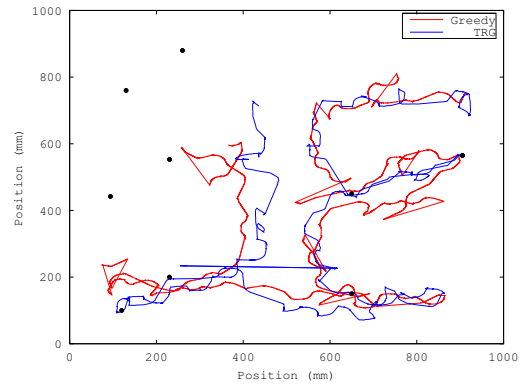


(e) Non-switching replans average values with error bars as Γ changes



(f) Switching replans average values with error bars as Γ changes

Figure 5.4: Replans results as Γ changes

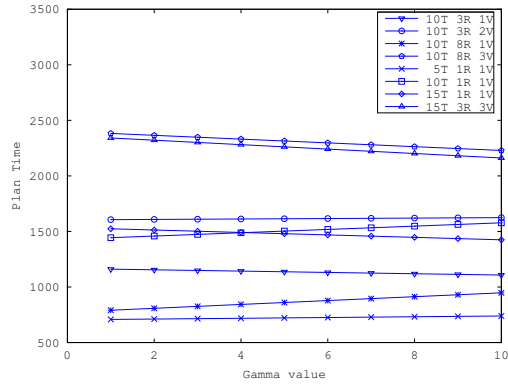
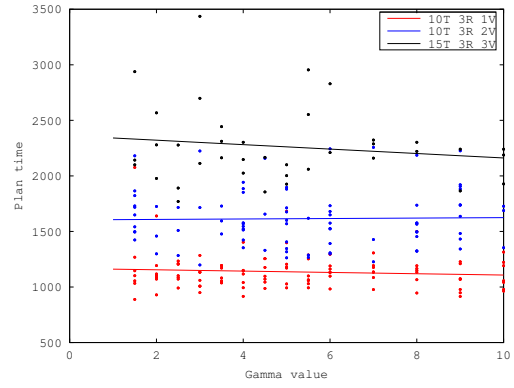
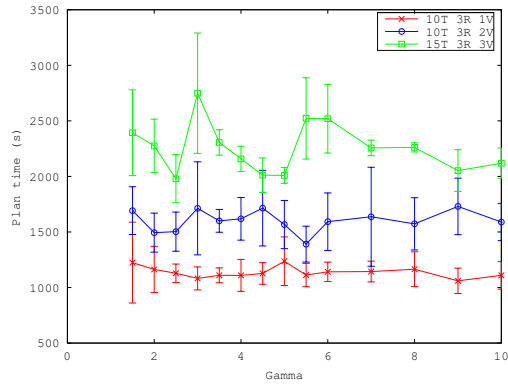
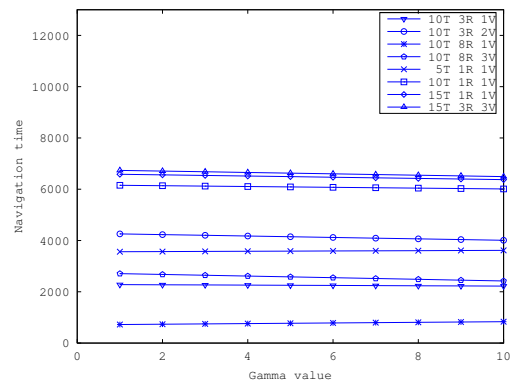
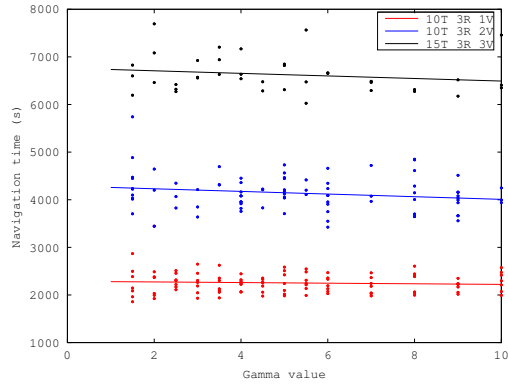
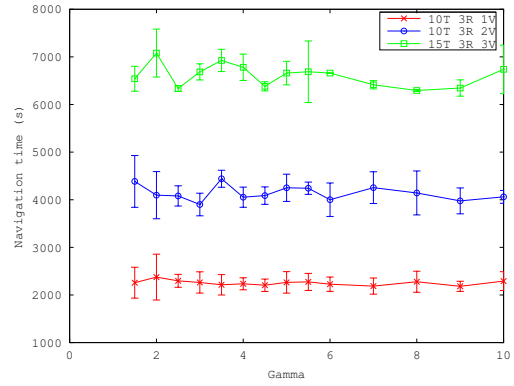
(a) Distance trends as Γ changes(b) Distance as Γ changes for $Vis \geq 1$ (c) Distance as Γ changes

(d) TRG and Greedy robot paths for a single robot in environment 10T 3R 2V, task locations are shown by black circles

Figure 5.5: Distance results as Γ changes

Figure 5.3(e) shows the average distances traveled by each robot for each of the approaches. We observe that as the average task load L of the robots increase (from left to right on the x-axis), the distances traveled by the robots increases. The robots using the TRG-based approach travel similar distances to those using the greedy approach. This is because both approaches use the same path planner. However, the TRG-based approach does travel a small amount more than the greedy approach (approximately 6% longer). This is due to when the robot decides to abandon its current task for another task, the greedy approach will switch as soon as the other task becomes closer, which in some cases is the best decision, whereas the TRG-based approach will continue to follow its previous task even though another task is closer. In some cases this is the best thing to do because the closer task might be on the other side of a wall that the robot has yet to explore with its laser and actually require more distance to explore enough of the wall to realize that the task is no longer the closest. In some of the environments with a larger load value, the TRG-based approach starts to perform better. Figure 5.5(d) shows one such case for the ten task, three robot, two visit environment. The red line is the path followed by a robot executing the greedy nearest task first method, whereas the blue line is the path of the robot in the same starting location using the TRG-based method. As can be seen in the figure, the robot using the Greedy approach tends to change its direction much more often compared to the TRG-based approach, this is due to the larger number of times that the robot switched between tasks to execute.

Figures 5.4, 5.5, and 5.6 show the effect varying Γ has on the various parameters collected. In general, it can be seen that Γ has a minor effect on the behaviour of the approach. This can also be seen in Table 5.1 which shows the precise rate of change in the tested parameters as Γ increases. For example, the largest change in distance is $100.76 \frac{\text{m}}{\Gamma}$ for environment 10T 8R 3V. In this approach the robot traveled approximately 4,000m which is only a change of 2.5% for each change in Γ . One will also notice that in the environments with a large load ($L > 5$), a larger value of Γ causes the average distance to

(a) Plan time trends as Γ changes(b) Plan time as Γ changes for $Vis \geq 1$ (c) Plan time as Γ changes(d) Navigation time trends as Γ changes(e) Navigation time as Γ changes for $Vis \geq 1$ (f) Navigation time as Γ changesFigure 5.6: Timing results as Γ changes

Environment	Distance	N.S. replan	S. replan	P. Time	N. Time
10T 3R 1V	11.21	0.01	-0.06	-5.92	-6.45
10T 3R 2V	-30.66	-0.03	0.07	1.99	-27.75
10T 8R 1V	13.25	0.16	0.01	17.42	12.19
10T 8R 3V	100.76	-0.05	-0.17	-16.97	-31.66
5T 1R 1V	7.54	0.03	0.02	3.44	5.90
10T 1R 1V	-20.23	0.06	0.11	14.88	-15.82
15T 1R 1V	-32.51	-0.03	-0.09	-11.02	-23.20
15T 3R 3V	-27.80	0.004	-0.23	-19.99	-26.97

Table 5.1: Change in Distance, Non-switching replans, Switching replans, Plan time, and Navigation time as Γ increases

decrease, whereas a smaller load causes the average distance to increase. This is due to the robot following the path to the wrong path a longer distance before it decides to switch to the task the ends up being better. This is more prominent in environments with larger loads because the robot has to cover a larger portion of the environment compared to the lower load values.

The effect of increasing Γ on switching replans (Figures 5.4(b), 5.4(d), and 5.4(f)) depends on the number of visits that each task requires. In general, the environments where $V_{is} > 1$, a larger Γ value resulted in fewer switching replans. This is because a robot will only become more likely to switch tasks after the path length to its current task becomes more the Γ times larger then the closest task by path length (Equations 4.4 and 4.3). As Γ becomes larger, it becomes more difficult for this condition to be satisfied, which causes fewer TRUE observations for PLL_{ij} , which in turn makes it less likely for the replan to be a switching replan. On the other hand, the effect on non-switching replans is distributed across all of the environments, meaning that there are likely other properties of the environments that are effecting this parameter, such as the density of obstacles, the density of robots, or the amount of collisions between robots.

Navigation time, unlike the other parameters, has a larger value of Γ resulting in less navigation time. In all, except two, environments a larger value of Γ caused the average navigation time to decrease (see Figure 5.6(d)). This is due to the robot not switching tasks



Figure 5.7: Overhead and side view of environment used for testing with physical robots, white dots represent the task locations

too often. Each time the robot switches a task, it has to rotate to follow the newly selected path. Because this rotation takes time, it can add up as the robot switches between tasks. Planning time, like non-switching replans, are scattered across various visit levels and ranges for load (Figure 5.6(a)). However, an interesting thing to note is that planning time increases and decreases in the same environments where non-switching replans increased and decreased. This is because more replans result in more planning time, and the effects of the switching replans was not enough to cancel out the change from non-switching replans.

5.2 Physical Robots

5.2.1 Setup

For the physical robot experiments, we converted the environment shown in Figure 5.1(a) to fit in the limited $3 \times 4 \text{ m}^2$ available. This new environment is shown in Figure 5.7, where the white dots represent the tasks that the robot must visit. Like the simulation environments, this environment was designed such that a greedy closest-task-first algorithm would switch tasks 50% of the time. In this environment we tested with a fixed value of $\Gamma = 5.0$. We also tested with $|R| = \{1, 2\}$ and $Vis = 1$. Results were averaged over three runs.

Robots	Visits	Distance Traveled (cm)	# Switches	# Non-switches	Plan Time (s)	Navigation Time (s)
1	1	2102.88 \pm (349.94)	4 \pm (0)	0.67 \pm (1.15)	613.97 \pm (144.54)	1415.96 \pm (238.87)
2	1	1130.45 \pm (86.92)	2.83 \pm (0.58)	2.5 \pm (1.32)	835.6 \pm (269.11)	746.86 \pm (48.66)
2	2	6118.95 \pm (2689.47)	4.5 \pm (1.5)	3.17 \pm (2.52)	760.86 \pm (372.83)	3921.43 \pm (1704.98)

Table 5.2: Physical robot experiment results

5.2.2 Results

Table 5.2 shows the results of the physical robot experiments for the same metrics as the simulation experiments. In both environmental, the robots were able to navigate and visit the tasks with the required number of visits. In comparing these results to those shown in Figure 5.3, we can see that the hardware performed fairly similar. For example, Table 5.2 shows that the five tasks, one robot, one visit environment had four switching replans, which is within the margin of error for the switching replans shown in Figure 5.3(a).

During experiments, one issue was noted, that of size of the passageways. In the limited space afforded for the hardware tests, a single robot could block one of the rooms in the environment and prevent the robot from entering. This also revealed an issue for future investigation for the coordination strategy, namely that it was possible for both robots to become stopped by the algorithm and become unable to move due to their need to swap locations. This usually meant that at least one of the robots would need to move away from the selected goal location to allow the other robot to be freed, which is not currently possible under our coordination strategy. However, it performs well in all other tested scenarios.

Chapter 6

Conclusions and Future Work

In this thesis, we introduced the TOP-U problem where robots have to determine the order to visit a set of task locations when the path costs between each pair of tasks can vary dynamically as the robot discovers obstacles while navigating between tasks. As a solution to this problem, we proposed a data structure called a task reachability graph (TRG). The TRG provides a mechanism to encode inter-task dependencies, the path length between the task locations, and the belief in the task path length being correct. We then proposed techniques to use the TRG to integrate task planning and motion planning using a sampling-based path planner to find current estimates for path length and a HMM-based technique to find the belief in those path costs based on the path length between tasks. Finally, an MDP-based algorithm was proposed that uses information from the TRG to select a suitable task order to reduce the cost in terms of time taken or distance traveled to visit the tasks.

6.1 Lessons Learned

We tested our proposed approach in both simulated and physical robots. The simulation was done using a simulator that provides an accurate simulation of the real-world dynamics of robots. When compared to a greedy closest-task-first task allocation method, our TRG-based approach performed 60% fewer task switching replans, 40% fewer replans overall.

This means that using the TRG-based approach reduced the amount of times that the robot has to replan a considerable amount, which reduces the time taken and reduces the amount of resources the robot has to use. It also took 60% less planning time and 58% less navigation time. In general, using the TRG-based approach reduces the planning and navigation times considerably. The TRG-based approach also traveled almost similar distances to the greedy approach, only taking about 6% longer of a path. In general, the TRG-based approach travels and uses a similar amount of energy in the locomotion of the robot compared to the greedy closest-task-first approach. We also tested an algorithm parameter Γ , which was shown to have minimal effect on the performance of the algorithm. Though it was shown that for an environment with a larger load of tasks that each robot has to complete, a larger Γ resulted in a decrease in distance traveled. If the number of visits per task was greater than one, a larger Γ also resulted in a decreased number of switching replans. And in most cases, a larger Γ resulted in a decrease in navigation time. In general, the Γ parameter had a very minor effect on the performance of the proposed approach, however it can be used to do fine tuning of the performance, though not required for the approach to perform better than the greedy closest-task-first approach tested against. The approach was also tested on physical robots using three environmental settings, which produced results similar to those generated in the simulation.

6.2 Future Works

As future work, we would like to look into the following topics to understand the problem of simultaneous task and motion planning more effectively:

- Partial task ordering, where certain tasks must be performed before other tasks can be done.
- Robots completing portions of task at same time: Multiple robots have to work together simultaneously to complete the task.

- Multi-robot coordination techniques: Finding better techniques that do not require all robots to stop moving.
- Environment parameter investigation: Finding what parameters of environment effects the results of our proposed technique.
- Improved planning time: Finding ways to not use as much time in updating path plans

To handle partial task ordering, it will be important to account for the changes in the TRG due to completion of tasks. The algorithms would need to account for the directed edge nature of a TRG with dependencies between tasks. This could be handled through online manipulation of the TRG edges as tasks are completed. For example, if task 3 is dependent on task 2, then until task 2 is completed, the only edge into task 3 will be from task 2, but once task 2 is completed, all other tasks can gain an edge into task 3 because the dependency has been completed.

Some tasks may require multiple robots to be present at the task location at the same time in order to perform the task. An example would be the movement of a large object that one robot does not have the power to move, but with multiple robots, they can provide collectively enough power to move it. To do this, the robots would need to communicate and take into account the task plans of other robots, because accepting and navigating to a joint robot task is only useful if another robot can meet the robot at the task location. To accomplish this, an extra term can be added to the computation of a tasks cost, which is conditional on another robot going to the task at the same time. The robot could announce to the other robots its intent to visit a task, and if another robot finds this task to also be reasonable to go to, can also select it and reply. However, if no other robot responds, the cost of going to the task increases to discourage the robot from going to a useless task if no other robot will join it there.

In our current approach, robots use a basic approach to perform multi-robot collision avoidance and coordination which requires all robots involved to stop movement towards their current task while the collision prevention is handled. In order to prevent the stopping of robots and be able to provide guarantees that the robots will successfully avoid collisions and continue to their tasks, planning will likely be required to be done in the joint configuration space, where the possible motions of at least a subset of the robots can be considered together. To prevent the high costs that planning in the joint configuration space usually requires, the joint planning can be limited to the area that the robots are in collision to provide short paths that can be found quickly amount the robots in danger of collision.

In our experiments, we noticed that the changes in the metrics as environments changed could not be directly mapped to one of the parameters that we tested. In order to best recommend the values of Γ for an environment or which planning methods are the best, we would like to define parameters of the type of environment the robot is working in, and see what the relationship between these parameters and the performance of the algorithm is. Parameters that could be investigated include distribution of obstacles, density of obstacles, distributions of robots, and others.

Our current approach uses a path planner that accounts for uncertainty in obstacle locations, but takes a long time to generate all the paths. To help improve the planning time, we would like to look into either using other path planners that are available, or adapting the current one to reduce the number of computations or collision checks that must be performed. Example path planners that could be investigated are Lazy PRM [2] and Informed RRT [9].

In conclusion, we proposed a technique to solve simultaneous task and motion planning by using a data structure called the task reachability graph. Based on this structure we proposed a HMM and MDP based algorithm that performs better then a greedy nearest-task-first task allocation algorithm by at least 40% in most metrics tested, while only traveling 6% more distance. Our approach was verified in accurate simulations and using

physical robots.

Bibliography

- [1] F. Aurenhammer. Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.
- [2] R. Bohlin and L.E. Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 521–528 vol.1, April 2000.
- [3] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, USA, 1988.
- [4] H. Choset, W. Burgard, S. Hutchinson, G. Kantor, L. Kavraki, K. Lynch, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.
- [5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, June 2009.
- [6] V. Desaraju and J. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.
- [7] L. Dou, M. Li, Y. Li, Q. Zhao, J. Li, and Z. Wang. A novel artificial bee colony optimization algorithm for global path planning of multi-robot systems. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 1186–1191, Dec 2014.
- [8] D. Ferguson and A. Stentz. Field d*: An interpolation-based path planner and replanner. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, pages 1926–1931, 2005.
- [9] J. Gammell, S. Srinivasa, and T. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pages 2997–3004, 2014.
- [10] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Computers*, 31(1):48–59, 1982.
- [11] B. Gerkey and M. Mataric. Sold!: Auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, pages 758–768, Oct 2002.

- [12] B. Gerkey and M. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.
- [13] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011.
- [14] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 566–580, 1996.
- [15] D. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412 – 442, 1990.
- [16] S. Koenig and M. Likhachev. D*lite. In *Eighteenth National Conference on Artificial Intelligence*, pages 476–483, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [17] J. Kuffner and S. Lavalley. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Intl Conf. on Robotics and Automation*, pages 995–1001, 2000.
- [18] S. Lavalley. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.
- [19] M. Likhachev, D. Ferguson , G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.
- [20] S. Loibl, D. Meyer-Delius, and P. Pfaff. Probabilistic time-dependent models for mobile robot path planning in changing environments. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 5545–5550, 2013.
- [21] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, October 1979.
- [22] B. Luders, S. Karaman, and J. How. Robust sampling-based motion planning with asymptotic optimality guarantees. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Boston, MA, August 2013.
- [23] V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *ALGORITHMICA*, 1987.
- [24] P. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1261–1267, 2006.

- [25] A. Muñoz-Meléndez, P. Dasgupta, and W. Lenagh. A stochastic queueing model for multi-robot task allocation. In *ICINCO (1)*, pages 256–261, 2012.
- [26] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, June 2009.
- [27] S. Rutishauser, N. Correll, and A. Martinoli. Collaborative coverage using a swarm of networked miniature robots. *Robotics and Autonomous Systems*, 57(5):517–525, 2009.
- [28] M. Saha and P. Ito. Multi-robot motion planning by incremental coordination. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5960–5963, Oct 2006.
- [29] I. Sucan and L. Kavraki. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5492–5498, 2011.
- [30] I. Sucan and L. Kavraki. Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs. In *IEEE International Conference on Robotics and Automation*, pages 4822–4828, St. Paul, May 2012.
- [31] G. Wagner and H. Choset. Subdimensional expansion for multirobot path planning. *Artif. Intell.*, 219:1–24, 2015.
- [32] D. Wicke, D. Freelan, and S. Luke. Bounty hunters and multiagent task allocation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’15, pages 387–394, 2015.
- [33] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*, Toronto, Canada, 2010.
- [34] B. Woosley and P. Dasgupta. Multirobot task allocation with real-time path planning. In *FLAIRS Conference*, pages 574–579, 2013.
- [35] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *I. J. Robotic Res.*, 25(1):73–101, 2006.