Bar-Ilan University
Department of Computer Science

Topics in Multi-Robot Teamwork

by

Meytal Traub

Advisor: Prof. Gal Kaminka

Submitted in partial fulfillment of the requirements for the Master's degree
in the department of Computer Science

Ramat-Gan, Israel
July 2011

This work was carried out under the supervision of

Prof. Gal A. Kaminka

Department of Computer Science, Bar-Ilan University.

Abstract

In recent years there is a growing interest in multi-robots systems, where a group of N robots are working collaboratively in order to execute a given task. This thesis addresses two open challenges in multi-robot systems. The first is the challenge of deciding on which robot, out of a group of robots, should travel to a goal location, to carry out a task there. The second is the challenge of integrating multiple and different multi-robot controllers into a robust system.

A common decision problem in multi-robot applications involves deciding on which robot, out of a group of N robots, should travel to a goal location, to carry out a task there. Trivially, this decision problem can be solved greedily, by selecting the robot with the shortest expected travel time. However, this ignores the inherent uncertainty in path traversal times; we may prefer a robot that is slower (but always takes the same time), over a robot that is expected to reach the goal faster, but on occasion takes a very long time to arrive. In the first part of this thesis we make several contributions that address this challenge. First, we bring to bear economic decision-making theory, to distinguish between different selection policies, based on risk (risk averse, risk seeking, etc.). Second, we introduce *social regret* (the difference between the actual travel time by the selected robot, and the hypothetical time of other robots) to augment decision-making in practice. Then, we carry out experiments in simulation and with real robots, to demonstrate the usefulness of the selection procedures under real-world settings, and find that travel-time distributions have repeating characteristics.

In the second part, we address the challenge of integrating multiple and different multi-robot controllers into a robust system. Multi-robot formations are of increasing interest to robotics researchers (as a canonical research problem), and to robot system builders (e.g.,for unmanned convoys). Indeed, there exists vast literature on various techniques for maintaining formations in a variety of settings, and for a variety of robots. However, little attention has been given to the possibility of using multiple formation controllers, all integrated together for greater formation robustness. In this part, we make two contributions. First, we address a

key challenge in integration, that of joint distributed selection and execution of the correct controller, at the same time. We demonstrate how to utilize a teamwork software engine to automate this joint selection. Second, we describe one such integrated system, which uses several different formation-maintenance controllers for greater robustness.

Acknowledgments

This thesis would not have been possible without the support of many people. First, I would like to express my gratitude to Prof. Gal Kaminka, for giving me the opportunity, advising and supporting me, I wish he will be able to drink his coffee quietly from now on.

I am indebted to Noa Agmon, for being wonderful research partners and for helping me through the entire process even on her spare time.

To Yehuda Elmaliach, Dan Erusalimchik, Alon Levy, Eran Sadeh-Or and Vladimir Sadov for their help and sleepless nights during the project.

To the MAVERICK lab, for being my friends and making me feel like at home. Lastly, I am grateful to my family, for loving and supporting me through this journey.

This research was supported in part by Israeli Science Foundation (ISF) grant #1357/07.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent years there is a growing interest in multi-robots systems, where a group of N robots are working collaboratively in order to execute a given task. This thesis addresses two open challenges in multi-robot systems. The first (Section 1.1) is the challenge of deciding on which robot, out of a group of N robots, should travel to a goal location, to carry out a task there. The second (Section 1.2) is the challenge of integrating multiple and different multi-robot controllers into a robust system. We describe the two parts in detail below.

1.1 Selecting a Robot to Reach a Goal

A common decision problem in multi-robot settings involves deciding on which robot, out of a group of N robots, should travel to a goal location, to carry out a task there. This decision repeats in many applications: in multi-robot exploration (e.g., deciding who should go to explore a new frontier), in package delivery robots (e.g., deciding who should go to pick up a package), and in other service robotics applications (e.g., in hospitals). In all of these, robots can plan a path to reach their destination, in an environment that is—for the most part—known to them. Thus, in principle, they can analytically predict their travel time to any location.

Trivially, this decision problem can be solved greedily, by selecting the robot with the shortest predicted travel time [61], or using a market-based allocation

scheme (see [15]). However, this ignores the inherent variance in the actual path traversal times, due both to motion and sensing errors, as well as multiple factors that affect a robot's velocity (e.g., battery level, unknown obstacles). Solutions that have been proposed to address these challenges include using machine learning to better predict actual travel times under varying conditions [37, 66, 38], or other path-generation techniques that provide estimates [10, 6].

A common thread through previous work is that it focuses on scalar predictions; a single number that denotes the expected travel-time for each robot. Unfortunately, scalar predictions hide important information about the uncertainty in the predictions. In particular, a scalar denoting expected cost ignores information about the distribution of possible costs, best- and worst-case costs, etc. As a result, guarantees on the cost of task execution are not possible.

For instance, supposed that we must send one of two robots to a target location X. Robot A's path to X takes 100 seconds, through a free corridor. But if the corridor is busy with traffic (a rare occurrence), it may take up to 200. In contrast, robot B's travel time is always 150 seconds, through a specialized service way. Since the corridor is normally clear, we might choose robot A for the task. But if we wanted to absolutely guarantee delivery within 150 seconds, we would choose robot B. Note that if we only know the expected (i.e., mean) travel time, we cannot make the necessary distinction that allows this decision.

In Part I of this thesis, we make several contributions that address the challenge involved in selecting a robot to go to a target location, given that each robot has a distribution over predicted travel times: First, we bring to bear economic decision theory that distinguishes between different selection policies, based on *risk*: risk averse, risk seeking, risk neutral, and bounded-risk selection. Second, we show that under some conditions, the selected robot may still not be a reasonable choice in practice. We thus introduce the use of *social regret* (the difference between the actual travel time by the selected robot, and the hypothetical time of another robot) to augment decision-making. Social regret is inspired by economic notions of regret, though the definitions differ.

Then, we carry out experiments in simulation and with real robots, to demonstrate the usefulness of the selection procedures under real-world settings. We empirically demonstrate that even under static conditions of the environment, when

it is completely known to the robots, sensor and actuator errors leads to significant variance in the execution of path-following tasks. This variance leads to non-trivial distribution of costs, which in turn necessitates reasoning about the different optimization criteria when making the selection between robots. Finally, we show empirically that travel time distributions have repeating characteristics (specifically, they fit extreme value distributions).

The work reported in this part has been published in:

- Meytal Traub, Gal A. Kaminka and Noa Agmon. 2011. *Who Goes there? Using Social Regret to Select a Robot to Reach a Goal*. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2011) (Full paper).

1.2 Integrating Redundant Multi-Robot Formation Controllers for Robustness

Multi-robot formation maintenance is of increasing interest to robotics researchers (as a canonical research problem), and to robot system builders (e.g., for unmanned convoys). In formation maintenance, the goal is for a group of robots to move while maintaining relative positions with respect to each other (typically describing a specific geometric shape).

Indeed, there exists vast literature on various techniques for maintaining formations in a variety of settings, and for a variety of robots. This includes detailed discussions of separation-bearing control (SBC) [25, 14, 13], separation-separation control (SSC) [25, 14, 13], the use of dead-reckoning and communications [23], and more. Different controllers can be used in different settings; they have advantages and disadvantages which can complement each other. However, little attention has been given to the possibility of using multiple formation controllers, all integrated together for greater formation robustness.

Unfortunately, integrating multiple controllers together is not an easy task. Within each type of multi-robot controller ,each robot executes its own individual controller, and the formation is created by the distributed execution of the individual controllers. To be effective, the selection of the individual controllers,

and their parameters (e.g., which robot is following whom) must be coordinated. Moreover, if multiple multi-robot (joint) control schemes exist, then the robots must also coordinate their selection, so that the same scheme is selected by all robots, at the same time.

We tackle these key challenges in integration, that of joint selection and execution of an agread upon multi-robot controller, at the same time. We demonstrate how to utilize a teamwork software engine to automate this joint selection. Teamwork software (such as BITE [44, 45] or CogniTAO [11]) allows sharing of information, and in particular allows synchronous joint selection of controllers for execution.

Its use in robotics has been reported before [44, 45], but not focusing on formation maintenance or integration of different control schemes. Here, we describe in detail how such software significantly eases the development, integration, and execution of multi-robot formation controllers.

Building on the ability to jointly select and execute a distributed multi-robot controller, we describe a formation maintenance system, which integrates together several different formation-maintenance controllers for greater robustness. The robots jointly switch between different controllers, so as to address intermittent failures in sensing or communications. As a result, robustness of the formation increases. Moreover, the robots—now acting as a team—respond to robot death failures: when one robot fails, the others stop. We demonstrate the capabilities of the system using real and simulated robots.

Part I

Who Goes *There*? Selecting a Robot to Reach a Goal Using Social Regret

A common decision problem in multi-robot applications involves deciding on which robot, out of a group of N robots, should travel to a goal location, to carry out a task there. Trivially, this decision problem can be solved greedily, by selecting the robot with the shortest expected travel time. However, this ignores the inherent uncertainty in path traversal times; we may prefer a robot that is slower (but always takes the same time), over a robot that is expected to reach the goal faster, but on occasion takes a very long time to arrive. We make several contributions that address this challenge. First, we bring to bear economic decision-making theory, to distinguish between different selection policies, based on risk (risk averse, risk seeking, etc.). Second, we introduce *social regret* (the difference between the actual travel time by the selected robot, and the hypothetical time of other robots) to augment decision-making in practice. Then, we carry out experiments in simulation and with real robots, to demonstrate the usefulness of the selection procedures under real-world settings, and find that travel-time distributions have repeating characteristics.

Chapter 2

Background: Choosing a Robot

A common decision problem in multi-robot settings involves deciding on which robot, out of a group of N robots, should travel to a goal location, to carry out a task there. This decision repeats in many applications: in multi-robot exploration (e.g., deciding who should go to explore a new frontier), in package delivery robots (e.g., deciding who should go to pick up a package), and in other service robotics applications (e.g., in hospitals). In all of these, robots can plan a path to reach their destination, in an environment that is—for the most part—known to them. Thus, in principle, they can analytically predict their travel time to any location.

Trivially, this decision problem can be solved greedily, by selecting the robot with the shortest predicted travel time [61], or using a market-based allocation scheme (see [15]). However, this ignores the inherent variance in the actual path traversal times, due both to motion and sensing errors, as well as multiple factors that affect a robot's velocity (e.g., battery level, unknown obstacles). Solutions that have been proposed to address these challenges include using machine learning to better predict actual travel times under varying conditions [37, 66, 38], or other path-generation techniques that provide estimates [10, 6].

A common thread through previous work is that it focuses on scalar predictions; a single number that denotes the expected travel-time for each robot. Unfortunately, scalar predictions hide important information about the uncertainty in the predictions. In particular, a scalar denoting expected cost ignores information

about the distribution of possible costs, best- and worst-case costs, etc. As a result, guarantees on the cost of task execution are not possible.

Section 2.1 describes the related work on task cost prediction while Section 2.2 address the related work on market based techniques

2.1 Task Cost Prediction

There have been several investigations that attempt to predict travel time or related costs. To the best of our knowledge, none addressed complete distributions.

Bobrow [6] presented an algorithm using B-spline polynomials for generating optimized paths for a three degrees of freedom elbow type robot in an environment which contains obstacles. The algorithm requires the joint displacement as a function of the path parametrization, and estimates the time of execution.

Heero et al. [38] present a method for learning the shortest path in a partially-known environment by using a rectangular grid-based map. For each path they saved four parameters: number of re-plans, travel time, travel distance and deviation from the originally pre-planned path. According to their algorithm, when an unknown path need to be followed (i.e. that does not appear in the known paths list) then the path is being chosen by the shortest distance from the robot position to the goal and the parameters are being saved for this new path. If the robot need to reach a goal that already have been learned than the one with the lowest re-planning is being chosen. But the learning is path-specific; changes in the start or end point requires learning the new path from scratch. Furthermore, the method used the distance and time without considering the errors in the robot movements and sensors which were removed from the training data.

Chaudhry [10] presented an algorithm for generating paths using matrix representation of the robot's previous path traversals. New paths are created by applying transformations to the matrix, given the new path requirements. Both investigations use the previous experiences to generate travel time predictions.

Sofman et al. [62] demonstrated an approach for learning Gaussian distributions associated with the local environments of the robot, to improve navigation and the travel speed. They use the models to generalize to new environments. The learning process is done using Bayesian probabilistic framework. But although

the model is represented as distribution, it is being modeled as a Gaussian and the calculations are being done using the mean and variance of the Gaussian while our distributions are heavy tailed distributions. They do not learn travel time distributions, and in any case as we show, travel time distributions are not Gaussian.

A related—though inverse—problem to ours is the problem of choosing a path, out of k possible paths, for a single robot to reach a goal location. Haigh and Veloso developed ROGUE [37]. It learns situation-dependent rules based on the success or failure in carrying out its tasks, and in particular, learns to take different paths depending on the time of day, expected use of the corridor, etc. ROUGE learns these situated-dependent cost predictions by examining the mean costs of travel for given locations. Thurn et al. developed MINERVA, an interactive tour guide robot for the Smithsonian museum [66]. It used POMDP methods to learn and plan its motion. The use of POMDP is similar to the risk-neutral policy, one of a number we present in this chapter.

Our notion of regret is inspired by—but different than—notions of regret in economics. Economic regret were introduced by Bell [4] and by Loomes and Sugden [52], who concluded that people do not necessarily maximize their expected utility, but also consider the possible loss they are willing to accept from making a choice. They defined regret as a symmetric function with respect to two choices: choosing A rather than B (and the associated gain/loss depending on the outcome) minus the gain/loss from choosing B rather than A. In our case we calculate the regret with respect to all other choices, yet the comparison between the symmetric cases is done after the calculation (hence our regret function is asymmetric).

Foster and Vohra [26] discussed regret in online decision-making. Here, the goal is to choose a series of actions that will have minimal average loss with respect to the possible world's states, based on the previous states of the world. It considers decision schemes, according to which decisions are made along time (decision functions). The internal regret of a scheme is how much one will lose while choosing according to this scheme in different world states. The external regret is how much one will lose when comparing choices of scheme A and scheme B. Our definition of regret is similar to the definition of internal regret, as we do not compare between different decision schemes, but make a general computation

as for how much we will be sorry for making a certain choice with respect to other possible outcomes. However we evaluate with respect to the probabilities of costs, rather than on a limited history over time.

2.2 Market-Based Techniques

Market-based methods are sometimes used to assigning robots to tasks (e.g., a goal location to be reached; see [15] for a survey). In general, these methods rely on scalar cost estimates, and do not utilize information about travel cost distributions. However, they do address self-interest on the part of the robots, while in our work we assume robots are cooperative and truthful. Koenig et al. [48] uses a regret function, different from ours, to improve such auctions.

In task allocation using market-based approaches, each robot computes its cost and encapsulates it in the bid. The bids are being sent to the auctioneer whose duty is to choose the best option according to the bids.

Moreover, some frameworks for task allocation using market-based techniques have been implemented. The frameworks deals with centralized and distributed planning for the task allocation.

Dias et al. [20] dealt with dynamic environment in the context of communication problems, partial robot malfunction and robot death, but they did not handle the problem caused by the uncertainty of the environment because of noise and changes in the environment. Zlot et al [68] worked on solving exploration problem using market-based approach, but their cost function is relating between the robot resources by using the distance that the robots is passing to reach the goal. Their algorithm deals with communication problems but they do not pay attention to the uncertainty in the distance measurements of the robots, and therefore do not handle the case of wrong price estimation. Tradebot [18],[21], [16] [17] is a distributed mechanism framework which in some cases changing the mechanism to a centralized mechanism in order to improve the complexity and the efficiency of the solution, but this framework as well works on a given task and cost functions. Jones et al [41] have worked on the case of pickup robots team using market-based approach in a treasure hunt domain. They tests their work using Tradebot, but they as well used the distance that the robot requires to move as the cost function.

Gerkey and Mataric [28] presented a taxonomy for task allocation, based on scalar costs which were calculated using weights between parameters of the objective function. They did not deal with distribution of costs. In another case [31] they compose a sensor-actuator network while in this network the goals would be achieved using market-based approach. But they made an explicit note that the cost should be a metric cost and left the question of what is the cost as an open question. MURDOCH [30] is a distributed framework which implements market-based approach using negotiation between the robots. But the cost function is given to the framework according to the problem, therefore they are not dealing with the noise of the environment. In the experiments which were done using this framework, the cost function was build using the distance from the goal position to the robot according to the image which was acquired from the camera. In the framework experiment the bid is a scalar and there is no treatment to the sensors noise.

Sheng et al. [60] proposed a distributed algorithm for an area exploration problem using bidding. But they used this mechanism in order to choose the best choice from the options that were given. In their proposed algorithm they are counting on perfect sensing and therefore do not take care of mistakes because of sensing errors.

Simmons et al. [61] proposed an algorithm for coordinated exploration using market-based approach, where the robots send array of bids for frontiers in the map to explore, and the auctioneer chose the winning robot between those who offered the best offer for any frontier. The bids are constructed by the distance between the robots and the frontiers. But they too did not address the effects of sensing and location-estimation errors.

Bererton et al [5] developed a method for solving path planning problems using loosely coupled MDP. But they still using market-based approach, and they did not use any limit on the mistakes of the algorithm.

Hoplites [43] chooses between negotiation mechanisms and passive mechanism in order to give the best solution to the given task. But the task and the bid function is given to the framework. The framework does not handle the cost and utility functions at all.

M+ [8] is an implementation of Contract Net protocol for task allocation prob-

lem. Botelho and Alami send a list of attributes and constraints to fulfill by the robots that wants to participate in the auction, but they do not deal with the cost estimation of the robots bids.

Shehory and Kraus [59] showed an algorithm that allocates tasks to coalitions of autonomous agents. They used market-based approach in order to decide which task to allocate to which coalition, and showed how the dynamic of the environment effects on the size of the group and therefore on the bids that were offered. But they used the market-based approach for coalitions of self-interested agents, where the relation between the coalitions is not cooperative. Kraus [49] also showed cases where the agents are self interested but cooperative and therefore may use market-based approaches in order to allocate tasks. But those cases do not fit multi-robot teams because the cost of two robots completing a mission in the same conditions is equal and not different as in the cases that we discus.

Chapter 3

Selecting a Robot

The problem is to select a robot R_i, out of a group of N robots $R_1, \ldots R_N$, to carry out a task, while minimizing the cost. We assume that each robot can estimate its cost of task execution with some discrete probability distribution over k cost values $c_1, \ldots, c_k$. Each robot R_i has a vector of size k, $< p_1^i, p_2^i, \ldots, p_k^i >$ such that p_j^i is the probability that the cost of task execution (travel time, in our case) by robot R_i is c_j and $\sum_{j=1}^{k} p_j^i = 1$ (note that p_j^i can be equal to 0). We use this discrete distribution formalization for simplicity, in lieu of the continuous distribution case which is more natural for estimated travel times. Note also that each robot's travel time is an independent random variable, i.e., the probability of robot R_i having actual cost of c_j does not depend on the probability of some other robot having this or other cost.

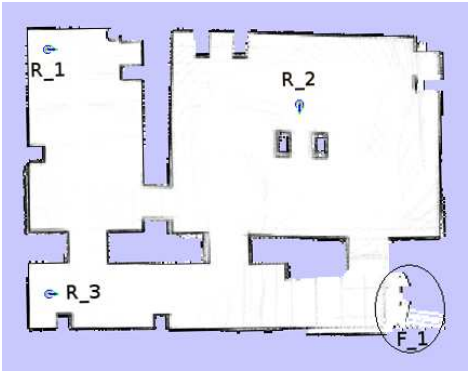


Figure 3.1: Three robots in exploration task. Map was generated using laser-based SLAM.

We use the following running example throughout this section. Figure 3.1 shows three robots $\{R_1, R_2, R_3\}$. One of these robots is to be sent to explore a new frontier, F_1, shown in the bottom right corner (circled). Each robot constructs a path (not shown) to the new location, and reports a distribution over estimated travel times. As we show in the experiments (Section 4.1), even in a completely static environment (let alone in dynamic environments), sensor and motion uncertainties cause some variance in this distribution.

Suppose the travel time distributions reported by the 3 robots are as given in Table 3.1. Each row shows the distribution of a different robot, with different columns denoting different costs. The last column shows the mean (expected) cost for each robot. Given different decision objectives, we would choose different robots to go to F_1. For instance, R_2 is most likely to reach F_1 faster (has a 87% chance of reaching F_1 in 86 seconds). But R_2 may also take up to 134 seconds for the same path. If we wanted to guarantee arrival within 2 minutes, we would choose R_3.

	$c_1 = 86$	$c_2 = 98$	$c_3 = 110$	$c_4 = 122$	$c_5 = 134$	$E(C)$
R_1	$p_1^1 = 0$	$p_2^1 = 0.6$	$p_3^1 = 0.23$	$p_4^1 = 0.17$	$p_5^1 = 0$	104.84
R_2	$p_1^2 = 0.87$	$p_2^2 = 0.03$	$p_3^2 = 0$	$p_4^2 = 0$	$p_5^2 = 0.1$	91.16
R_3	$p_1^3 = 0.6$	$p_2^3 = 0.22$	$p_3^3 = 0.1$	$p_4^3 = 0.08$	$p_5^3 = 0$	93.92

Table 3.1: Possible cost distribution for $R_{1,2,3}$ for arriving to F_1.

3.1 Risk-Based Selection

Choosing the robot $R_c \in \{R_1, \ldots, R_N\}$ to perform the given task is dependent on a decision policy, which prefers robots—all else being equal—based on the risk involved. For instance, if we have a fixed amount of time to explore a given area, we may want to select a robot that will definitely reach its target within the time allotted. On the other hand, we may decide to take more risks, hoping to reach the target faster than expected.

Such decision policies are well known in economic decision theory. We distinguish four well-defined policies, and outline the selection algorithm for each:

1. Minimize the expected travel time (risk neutral selection, Section 3.1.1).

2. Minimize the expected maximal travel time (risk averse selection, Section 3.1.2).

3. Maximize the expected minimal travel time (risk seeking selection, Section 3.1.3).

4. Bound the travel time by a constant A (bounded risk, Section 3.1.4).

3.1.1 Risk-Neutral Selection

Risk-neutral selection implies that we select the robot that minimizes the expected (mean) travel time. To do this, we compute the mean of every robot's distribution, and choose the robot whose mean is minimal

$$MinExp_C = \underset{1 \le a \le N}{\operatorname{argmin}} \{ \sum_{i=1}^{k} p_i^a c_i \}$$

where in case of a tie, we choose arbitrarily.

3.1.2 Risk-Averse Selection

In some cases we want to make sure that the worst-case scenario is addressed first, and that we have an absolute guarantee that the task will be carried out within a given amount of time. To do this, we need to look at the robots whose greatest time of arrival is minimal. Of course, the probability of actually taking this long time must also be taken into account. Thus what we want is to find the robot which minimizes the expected maximal cost. This is done in Algorithm 1.

Algorithm 1 MinExpMaxCost(R)

Require: $C = \{c_1, c_2, \ldots, c_k\}, R = \{R_1, R_2 \ldots, R_n\}$
  $v \leftarrow k$
  $Robots_{list} \leftarrow \{R_1, R_2, \ldots, R_n\}$
  while $\exists p_v^j = p_v^h, R_j, R_h \in Robots_{list}$ **do**
    $Robots_{list} \leftarrow \operatorname{argmin}_{R_i \in Robots_{list}} \{p_v^i\}$
    $v \leftarrow v - 1$
  return : $\operatorname{argmin}_{\mathbf{R_i} \in \mathbf{Robots_{list}}} \{\mathbf{p_v^i}\}$

Note that ties can be broken in different ways. For instance, we can choose the robot with the lower expected time among those that are returned.

We use Table 3.1 to illustrate. The algorithm creates a list of all the robots that available to execute the task $\{R_1, R_2, R_3\}$, and starts the run with the highest cost (134). It looks for two robots with the same probability to arrive the goal in cost 134. In this example R_1 and R_3 have the same probability ($p_5^1 = p_5^3 = 0$), so the loop will be entered. The algorithm choose the robots with the minimal probability to execute the task in cost 134 by argmin. By doing it, all the robots with probability higher than 0 will be removed from the list, i.e. robot R_2. The algorithm then examines the next highest cost, 122. While looking on the remaining robots $\{R_1, R_3\}$, their probability to arrive the target is different, therefor the loop will not be entered and the robot with the lowest probability to use this cost will be returned: R_3 ($p_4^1 = 0.17 > p_4^3 = 0.08$).

3.1.3 Risk Seeking Selection

The opposite policy to being risk averse is to be risk seeking; to hope for the best possible travel time of any of the robots. Here the selection is exactly the inverse of the above: We select the robot that maximizes the expected minimal cost. Algorithm 2 is thus the inversion of Algorithm 1.

Algorithm 2 MaxExpMinCost(R)

Require: $C = \{c_1, c_2, \ldots, c_k\}, R = \{R_1, R_2 \ldots, R_n\}$
  $v \leftarrow 1$
  $Robots_{list} \leftarrow \{R_1, R_2, \ldots, R_n\}$
  while $\exists p_v^j = p_v^h, R_j, R_h \in Robots_{list}$ **do**
    $Robots_{list} \leftarrow \text{argmax}_{R_i \in Robots_{list}} \{p_v^i\}$
    $v \leftarrow v + 1$
  return : $\text{argmax}_{\mathbf{R_i} \in \mathbf{Robots_{list}}} \{\mathbf{p_v^i}\}$

We again use Table 3.1 to illustrate. The algorithm starts the run with the lowest cost, i.e. 86. It looks for two robots with the same probability to arrive the goal in cost 86. In this example, there is no two such robots, and so it does not enter the loop and return the robot with the highest probability to arrive the goal in this cost, R_2 ($p_1^1 = 0 < p_1^3 = 0.6 < p_1^2 = 0.87$).

3.1.4 Bounded-Risk Selection

Finally, we may want to choose the robot that maximizes the probability of reaching the target within some limited amount of time. This is different from guaranteeing arrival within this time; it would still be possible that in the worst case, travel time will be longer. Nevertheless, we want to improve its chances of success within the time allotted.

Suppose we are given a time limit T. We can then calculate for each robot the cumulative probability that its travel time be smaller than T, and choose the robot that maximizes this probability.

For each robot R_a, we will calculate the following probability:

$$P[C <= T] = \sum_{c_i <= T, c_i \in C} p_i^a c_i$$

We will choose the robot that maximize the result of this equation. Note, that if only one robot have distribution of cost bellow the constant, then it will be chosen with probability of 1. If there is more than one robot that fits this, then we can select based on any of the other criteria (e.g., the best risk-seeking robot out of the candidates that fit the bound T).

3.2 Regretting the Selection

Despite the economic elegance of the selection policies described above, choosing the robot according to the risk type will not always give us a reasonable selection in practice. To see this, consider the following case (Table 3.2). Here, we apply the risk-averse policy, and select R_2: It is guaranteed to reach the goal in 199 seconds. However, unless this risk-averseness is somehow extremely strict, R_1 would have been a more reasonable choice: 90% of the time it would have reached the goal in 1 second. And even when it fails, it would do it in 200 seconds, a mere 1 second more than R_2.

Note that this is not always the case: It depends very much on the values c_i. If the c_i would have been $1, 2, 200$ rather than $1, 199, 200$, our deliberation would

	$c_1 = 1$	$c_2 = 199$	$c_3 = 200$	$E(C)$	E_{SoR}
R_1	$p_1^1 = 0.9$	$p_2^1 = 0$	$p_3^1 = 0.1$	20.9	0.1
R_2	$p_1^2 = 0$	$p_2^2 = 1$	$p_3^2 = 0$	199	178.2

Table 3.2: Robots distributions of costs to arrive at a goal. Expected cost and expected *SoR* are shown. Selecting R_1 over R_2 makes sense in practice.

not have reached the same conclusion, and the selection of R_2 would have held.

To conduct this deliberation formally, we define the *social regret* function, which measures, intuitively, the post-hoc payment (in travel time) that we make, given the selected robot.

Social Regret *SoR* is defined as the difference between the *actual* cost c_r of the task executed by robot R_a and the minimal cost of task execution in case some other robot would have executed the task in lower cost. In other words, looking at it from the team's perspective: How bad did the team do by choosing robot R_a to perform the task, given R_a's actual cost was c_r. Formally, *SoR* of robot R_a executing a task with actual cost c_r is $SoR(R_a, c_r) = \max_{j \neq i}(c_r - c_j, r > j)$.

Since we do not know *SoR* for any specific selection (it is by definition hypothetical), we compute the *expected SoR* for each robot R_a, given all other robots, and all possible outcomes. The expected social regret from choosing R_a, $E_{SoR}(R_a)$, is the probability that some other robot will execute the task with lower cost multiplied by the difference between the costs. We denote the probability that the actual *minimal* cost of task execution by some robot other than R_a is c_i by $PM_i(R_a)$. Note that by *minimal* cost we mean that there is no other robot that executed the task with cost $c_j, j < i$, and that at least one robot executed the task with cost c_i. Therefore, $E_{SoR}(R_a) = p_1^a \times 0 + p_2^a \times PM_1(R_a)(c_2 - c_1) + p_3^a \times [PM_1(R_a)(c_3 - c_1) + PM_2(R_a)(c_3 - c_2)] + \ldots$, and formally

$$E_{SoR}(R_a) = \sum_{i=2}^{k} p_i^a \times \sum_{j=1}^{i-1} PM_j(R_a)(c_i - c_j)$$

In order to complete the definition, it is necessary to determine $PM_j(R_a)$, i.e., the probability that some robot R_o, $1 \leq o \leq N$, $o \neq a$ will have minimal cost of c_j. This is the probability that all robots have minimal cost higher than c_{j-1} minus the probability that all robots have minimal cost higher than c_j, i.e., $E_{SoR}(R_a) =$

$$\sum_{i=2}^{k} p_i^a \{ \sum_{j=1}^{i-1} (c_i - c_j) [\prod_{h=1}^{N, h \neq a} (\sum_{l=j}^{k} p_l^h) - \prod_{h=1}^{N, h \neq a} (\sum_{l=j+1}^{k} p_l^h)] \}$$

3.2.1 When Should We Overrule The Selection?

Intuitively, $E_{SoR}(R_a)$ measures the potential cost of selecting R_a to carry out a task, given the estimated costs of its peers. Suppose that we have two robots R_i and R_j. What we want, is to compare the difference in the expected *SoR* of the two robots, to the gain from choosing one over the other. If this gain is smaller than the difference in expected *SoR*, then we should consider switching between them.

To illustrate, suppose R_i has been selected by some policy, and has a predicted travel time c_i (this is, for instance, its maximal time). Suppose we want to consider switching to a different robot R_j, with predicted cost c_j. In order to compute the profit form switching two robots we will calculate the distance between the expected *SoR* of R_i and R_j, $E_{SoR}(R_i) - E_{SoR}(R_j)$. We compare this value to the difference in costs between R_i and R_j, which is $(c_j - c_i)$, using the following function.

The Switch function *SwF* is defined as follows:

$$SwF = \begin{cases} 1 & \text{if } (E_{SoR}(R_i) - E_{SoR}(R_j)) > (c_j - c_i) \\ 0 & \text{otherwise} \end{cases}$$

If the *SwF* is 1, the social regret of using R_i is greater than the expected gain of using it, and we should consider switching our selection to R_j instead. We examine this in different selection policies below.

Example: Minimize the expected maximal cost Table 3.2 above describes the cost distributions for two robots, R_1 and R_2. As previously discussed, strict risk-averse policy would select R_2 for the task, since it is guaranteed to reach the target in 199 seconds. However, by risking just one additional second, we actually have much better average performance if we choose R_1.

SwF identifies this opportunity. The difference between c_3 (R_1's cost) and c_2 (R_2's cost) is 1, while the distance between the $E_{SoR}(R_2)$ and $E_{SoR}(R_1)$ is 178.1.

Thus *SwF* is 1, and we should consider switching our selection to the other robot.

Example:Maximize the expected minimal cost We use another example, this time of risk-seeking policy, to illustrate further. In this scenario we try to reach the best performances but not necessarily we will choose the robot with the best expected regret, i.e., a case where there is a robot with a larger minimal cost but with a lower expected *SoR*).

Table 3.3 shows distribution of costs of two robots, R_1 and R_2. According to the risk-seeking policy describe above, the chosen robot to execute the task would be R_1. But although R_2's minimal cost is higher than the minimal cost of R_1, in most of the cases R_1 will preform the task in 200 seconds while R_2 will always perform the task in 2 seconds. Therefore we will prefer to switch our chosen robot to be R_2 and not R_1.

Again, *SwF* can help us make this decision. By looking on the distance between the two minimal costs of the robot, we can see that $c_2 - c_1 = 4$, while the distance between the expected *SoR* is 175.1. Thus $SwF = 1$, and we should consider switching to R_2 as the selected robot.

	$c_1 = 1$	$c_2 = 5$	$c_3 = 200$	E_{SoR}
R_1	$p_1^1 = 0.1$	$p_2^1 = 0$	$p_3^1 = 0.9$	175.5
R_2	$p_1^2 = 0$	$p_2^2 = 1$	$p_3^2 = 0$	0.4

Table 3.3: Robots' distributions of costs to arrive at a goal. Expected *SoR* are shown. Selecting R_2 over R_1 does not make sense, if we want to guarantee best worst-case settings.

Example: Switching in the case of a bounded risk Using a bounded-risk policy, we normally select the risk the is most likely to carry out the task within the time allotted. But by bounding the cost, we are not bounding the regret function. In other words, choosing the best robot given the bound T, does not reduce our expected *SoR* for the bounded cost, and we can still choose to switch based on *SwF*.

For example, table 3.2.1 shows distributions of costs of two robots, R_1 and R_2. A bound of $T = 7$, yields selection R_1 (with cumulative likelihood 0.2), over R_2 (cumulative likelihood 0). But the expected *SoR* of R_1 is much higher that the expected *SoR* of R_2. Indeed, using the *SwF* we might consider to change the constant T to be higher. By changing the constant T from 7 to 10, R_2 will have higher probability than R_1 to execute the task under the new bound. We will pay 3 in the bound but gain 70.6 in the expected regret $(E_{SoR}((R_1)) - E_{SoR}(R_2))$. The *SwF* will be 1 ($70.6 > 3$).

	$c_1 = 1$	$c_2 = 5$	$c_3 = 10$	$c_4 = 100$	E_{SoR}
R_1	$p_1^1 = 0.1$	$p_2^1 = 0.1$	$p_3^1 = 0$	$p_4^1 = 0.8$	72
R_2	$p_1^2 = 0$	$p_2^2 = 0$	$p_3^2 = 1$	$p_4^2 = 0$	1.4

Table 3.4: The bounded cost does not minimizes the expected *SoR*. When should we replace

3.2.2 A Short-Cut to Determining *SwF*

The computation of E_{SoR} for each robot, which is necessary whenever we select robots based on a policy different from risk-neutral selection, is tedious, and potentially time-consuming if the distribution's domains are large, or there are many robots.

Thankfully, it turns out that we do not need to compute E_{SoR} directly. To compute *SwF*, we want the difference $E_{SoR}(R_i) - E_{SoR}(R_j)$ for the two robots R_i, R_j. It turns out that this difference is exactly the difference in expected costs of the two robots, which is much easier to compute:

Theorem 1. *The difference between the expected costs of any two robots in a given team of N robots $\{R_1, \ldots, R_N\}$ (each with a discrete probability distribution over possible costs $c_1, \ldots, c_k$) is equal to the distance between the expected SoR of the same robots.*

Proof. Let R_1 and R_2 be two robots in a team, with discrete probability distribution $\{p_1^1, p_2^1, \ldots, p_k^1\}$ and $\{p_1^2, p_2^2, \ldots, p_k^2\}$ (respectively) over possible costs

$c_1,\ldots,c_k$ of a given task. We prove that, $\sum_{i=1}^{k} p_i^1 c_i - \sum_{i=1}^{k} p_i^2 c_i = E_{SoR}(R_1) - E_{SoR}(R_2)$.

$E_{SoR}(R_1)$, can be represented as,
$\sum_{i=2}^{k} p_i^1 \{\sum_{j=1}^{i-1}[(c_i - c_j)(\prod_{h=2}^{N}(\sum_{l=j}^{k} p_l^h) - \prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h))]\}$.
Similarly, $E_{SoR}(R_2) = \sum_{i=2}^{k}\{p_i^2(\sum_{j=1}^{i-1}[(c_i - c_j)((\sum_{x=j}^{k} p_x^1)(\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h)) - (\sum_{x=j+1}^{k} p_x^1)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h))]\}$.

Therefor, $E_{SoR}(R_1) - E_{SoR}(R_2)$
$= \sum_{i=2}^{k} p_i^1 \{\sum_{j=1}^{i-1}[(c_i - c_j)(\prod_{h=2}^{N}(\sum_{l=j}^{k}(p_l^h)) - \prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h))]\} - \sum_{i=2}^{k} p_i^2 \{\sum_{j=1}^{i-1}[(c_i - c_j)((\sum_{x=j}^{k} p_x^1)(\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h)) - (\sum_{x=j+1}^{k} p_x^1)(\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h)))]\}$
$= \sum_{i=2}^{k} p_i^1 [c_i \prod_{h=2}^{N}(\sum_{j=1}^{k} p_j^h) - c_i \prod_{h=2}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{j=1}^{i-1} c_j(\prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h) - \prod_{h=2}^{N}(\sum_{l=j}^{k} p_l^h))] - \sum_{i=2}^{k} p_i^2 [c_i(\sum_{j=1}^{k} p_j^1)\prod_{h=3}^{N}(\sum_{j=1}^{k} p_j^h) - c_i(\sum_{j=1}^{k} p_j^i)(\prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h)) + \sum_{j=1}^{i-1}(c_j(\sum_{l=j+1}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) - (\sum_{l=j}^{k} p_l^1)(\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h)))]$
$= \sum_{i=2}^{k} p_i^1 [c_i \prod_{h=2}^{N}(1) - c_i \prod_{h=2}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{j=1}^{i-1} c_j(\prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h) - \prod_{h=2}^{N}(\sum_{l=j}^{k} p_l^h))] - \sum_{i=2}^{k} p_i^2 [c_i \prod_{h=3}^{N}(1) - c_i(\sum_{j=i}^{k} p_j^1)\prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{j=1}^{i-1} c_j((\sum_{l=j+1}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) - (\sum_{l=j}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h))]$
$= \sum_{i=2}^{k} p_i^1 [c_i - c_i \prod_{h=2}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{j=1}^{i-1} c_j(\prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h) - \prod_{h=2}^{N}(\sum_{l=j}^{k} p_l^h))] - \sum_{i=2}^{k} p_i^2 [c_i - c_i(\sum_{j=i}^{k} p_j^1)\prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{j=1}^{i-1} c_j((\sum_{l=j+1}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) - (\sum_{l=j}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h))]$
$= \sum_{i=2}^{k} p_i^1 c_i + \sum_{i=2}^{k} - p_i^1 c_i \prod_{h=2}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{i=2}^{k}\sum_{j=1}^{i-1} p_i^1 c_j \prod_{h=2}^{N}(\sum_{l=j+1}^{k} p_l^h) - \sum_{i=2}^{k}\sum_{j=1}^{i-1} p^1, ic_j \prod_{h=2}^{N}(\sum_{l=j}^{k} p_l^h) + \sum_{i=2}^{k} - p_i^2 c_i + \sum_{i=2}^{k} p_i^2 c_i(\sum_{j=i}^{k} p_j^1)\prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h) - \sum_{i=2}^{k}\sum_{j=1}^{i-1} p_i^2 c_j(\sum_{l=j+1}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) + \sum_{i=2}^{k}\sum_{j=1}^{i-1} p_i^2 c_j(\sum_{l=j}^{k} p_l^1)\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h)$
$= \sum_{i=2}^{k} p_i^1 c_i - \sum_{i=2}^{k} p_i^2 c_i + \sum_{i=2}^{k}\sum_{j=i}^{k} - p_i^1 c_i p_j^2 \prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h) + \sum_{i=2}^{k}\sum_{j=1}^{i-1}\sum_{j=1}^{i-1} p_i^1 c_j p_l^2 \prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) - \sum_{i=2}^{k}\sum_{j=1}^{i-1}(\sum_{l=j}^{k} p_i^1 c_j p_l^2)\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h) + \sum_{i=2}^{k}(\sum_{j=i}^{k} p_j^1 c_i p_i^2)\prod_{h=3}^{N}(\sum_{j=i}^{k} p_j^h) - \sum_{i=2}^{k}\sum_{j=1}^{i-1}(\sum_{l=j+1}^{k} p_l^1 c_j p_i^2)\prod_{h=3}^{N}(\sum_{l=j+1}^{k} p_l^h) + \sum_{i=2}^{k}\sum_{j=1}^{i-1}(\sum_{l=j}^{k} p_l^1 c_j p_i^2)\prod_{h=3}^{N}(\sum_{l=j}^{k} p_l^h)$

By opening the equation,

$= \sum_{i=2}^{k} p_i^1 c_i - \sum_{i=2}^{k} p_i^2 c_i - p_2^1 c_2 p_2^2 \prod_{h=3}^{N}(\sum_{j=2}^{k} p_j^h) - p_2^1 c_2 p_3^2 \prod_{h=3}^{N}(\sum_{j=2}^{k} p_j^h) - \ldots - p_2^1 c_2 p_k^2 \prod_{h=3}^{N}(\sum_{j=2}^{k} p_j^h) - p_3^1 c_3 p_3^2 \prod_{h=3}^{N}(\sum_{j=3}^{k} p_j^h) -$

$p_3^1 c_3 p[2,4] \prod_{h=3}^N (\sum_{j=3}^k p_j^h) - \ldots - p_3^1 c_3 p_k^2 \prod_{h=3}^N (\sum_{j=3}^k p_j^h) - \ldots - p_k^1 c[k] p_k^2 \prod_{h=3}^N (\sum_{j=3}^k p_j^h) + p_2^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_2^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_2^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_3^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_3^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_3^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_3^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + p_3^1 c_2 p_4^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + \ldots + p_3^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + \ldots + p_k^1 c_1 p_1^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_k^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_k^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_k^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + p_k^1 c_2 p_4^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + \ldots + p_k^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) + \ldots + p_k^1 c_{k-1} p_k^2 \prod_2^k (p_k^h) - p_2^1 c_1 p_1^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_2^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - \ldots - p_2^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_3^1 c_1 p_1^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_3^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - \ldots - p_3^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_3^1 c_2 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_3^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_3^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_1 p_1^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_k^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - \ldots - p_k^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) - p_k^1 c_2 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_k^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_{k-1} p_{k-1}^2 \prod_{h=3}^N (\sum_{l=k-1}^k p_l^h) - p_k^1 c_{k-1} p_k^2 \prod_{h=3}^N (\sum_{l=k-1}^k p_l^h) + p_2^1 c_2 p_2^2 \prod_{h=3}^N (\sum_{j=2}^k p_j^h) + p_3^1 c_2 p_2^2 \prod_{h=3}^N (\sum_{j=2}^k p_j^h) + \ldots + p_k^1 c_2 p_2^2 \prod_{h=3}^N (\sum_{j=2}^k p_j^h) + p_3^1 c_3 p_3^2 \prod_{h=3}^N (\sum_{j=3}^k p_j^h) + p_4^1 c_3 p_3^2 \prod_{h=3}^N (\sum_{j=3}^k p_j^h) + \ldots + p_k^1 c_3 p_3^2 \prod_{h=3}^N (\sum_{j=3}^k p_j^h) + \ldots + p_k^1 c[k] p_k^2 \prod_{h=3}^N (p_k^h) - p_2^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_3^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_2^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_3^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_3^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - p_4^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - \ldots - p_k^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - \ldots - p_1^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_2^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - \ldots - p_k^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) - p_2^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - p_3^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - \ldots - p_k^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=3}^k p_l^h) - \ldots - p_{k-1}^1 c_{k-1} p_k^2 \prod_{h=3}^N (p_k^h) - p_k^1 c_{k-1} p_k^2 \prod_{h=3}^N (p_k^h) + p_1^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_2^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + \ldots + p_k^1 c_1 p_2^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_1^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_2^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + \ldots + p_k^1 c_1 p_3^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_2^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_3^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_k^1 c_2 p_3^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_1^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_2^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + \ldots + p_k^1 c_1 p_k^2 \prod_{h=3}^N (\sum_{l=1}^k p_l^h) + p_2^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + p_3^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_k^1 c_2 p_k^2 \prod_{h=3}^N (\sum_{l=2}^k p_l^h) + \ldots + p_{k-1}^1 c_{k-1} p_k^2 \prod_{h=3}^N (\sum_{l=k-1}^k p_l^h) + p_k^1 c_{k-1} p_k^2 \prod_{h=3}^N (\sum_{l=k-1}^k p_l^h)$

And by minimization of the elements,

$= \sum_{i=2}^k p_i^1 c_i - \sum_{i=2}^k p_i^2 c_i + \sum_{i=2}^k p_1^1 c_1 p_i^2 \prod_{h=3}^N (\sum_{j=1}^k p_j^h) - \sum_{i=2}^k p_i^1 c_1 p_1^2 \prod_{h=3}^N (\sum_{j=1}^k p_j^h)$

$= \sum_{i=2}^k p_i^1 c_i - \sum_{i=2}^k p_i^2 c_i + p_1^1 c_1 \sum_{i=2}^k p_i^2 - p_1^2 c_1 \sum_{i=2}^k p_k^1$

$= \sum_{i=2}^k p_i^1 c_i - \sum_{i=2}^k p_i^2 c_i + p_1^1 c_1 (1 - p_1^2) - p_1^2 c_1 (1 - p_1^1)$

$= \sum_{i=2}^{k} p_i^1 c_i - \sum_{i=2}^{k} p_i^2 c_i + p_1^1 c_1 - p_1^1 c_1 p_1^2 p_1^2 c_1 + - p_1^1 c_1 p_1^2$
$= \sum_{i=2}^{k} p_i^1 c_i - \sum_{i=2}^{k} p_i^2 c_i + p_1^1 c_1 - p_1^2 c_1$
$= \sum_{i=1}^{k} p_i^1 c_i - \sum_{i=1}^{k} p_i^2 c_i$

Therefore the distance between the expected costs of any two robots, and the distance between the expected *SoR* are equal.

□

3.2.3 Minimal Expected Cost is Safe Selection

For one of the policies we introduced, it turns out that we do not need to consider regret. We prove that by minimizing the expected cost, the expected social regret function, *SoR*, is minimized as well, and thus we would not want to switch to a different robot.

Theorem 2. *Given a team of N robots $\{R_1, \ldots, R_N\}$ each with a discrete probability distribution over possible costs $v_1, \ldots, v_k$ for a given task, if we choose a robot R_c that minimizes the expected cost for the task, then the expected social regret function SoR is minimized.*

Proof. Let $\{R_1, ..., R_{N-1}$ be a team of N robots. We will assume, without loss of generality that R_1 minimizes the executed cost of the task execution. Therefore in particular, R_1 minimizes the expected cost for all the possible couples of robots in the environment, i.e., for any given pair of robots (R_1, R_i) where $1 \leq i \leq N-1$, the selection of R_1 results in a minimal cost compared to the selection of R_i. We will prove that R_1 minimizes the expected *SoR* for N robots.

Let R_N be a robot that joins the task execution. If R_1's expected cost is smaller than R_N's expected cost $(E_C(R_1) < E_C(R_N))$ (R_1's expected cost is minimized), then according to Theorem 1, $(E_{SoR}(R_1) - E_{SoR}(R_N) = (E_C(R_1) - E_C(R_N) < 0$, i.e., the expected social regret of R_1 is minimized.

If R_1's expected cost is bigger than R_N's expected cost $(E_C(R_1) > E_C(R_N))$ (R_N's expected cost is minimized), then again according to Theorem 1, $(E_{SoR}(R_1) - E_{SoR}(R_N) = (E_C(R_1) - E_C(R_N) > 0$, i.e., the expected social regret of R_N is smaller than the expected social regret of R_1, and of any other robot in the team.

Therefore, the theorem holds for N robots, $N \geq 2$, i.e. if we choose a robot R_c that minimizes the expected cost for the task, then the expected social regret function SoR is minimized. □

Example, table 3.2 gives travel times distributions of two robots to a goal. As it shows in the table, if we will choose robot R_1 by minimizing its expected cost, we will minimizes the E_{SoR} as well.

Chapter 4

Path Travel in Practice

We experimented with simulated and physical robots, to examine the travel time distributions, and evaluate the use of social regret in practice. We use the results to demonstrate in Section 4.1 that even under ideal conditions, robots do indeed have variance in the time that it takes them to travel a given path, and that this variance needs to be taken into account as described above. Using those variance we examine and evaluate in Section 4.2 the decision-making policies we showed earlier. Finally in Section 4.3 we show that the travel time distributions have distinctive shapes, and in general fit the Generalized Extreme Value family of distributions.

4.1 Experiments with Robots

We used our laboratory as the environment for the experiments. First, we used a popular open-source laser-based SLAM package, GMapping [34], to allow the robots to construct a map of the environment. The results of the exploration and mapping process were used as the basis for the experiments; this is to make sure that all path-planning and movements were carried out using a map with realistic quality. For path planning, we used A^* with a fixed 4-neighbor grid laid out over the map. If the robot discovered an unknown obstacle on the way, it tried to go around the obstacle until a timeout occurred, in this case a new path was planned from the current location to the goal, given the new information about the

Figure 4.1: RV400 robot.

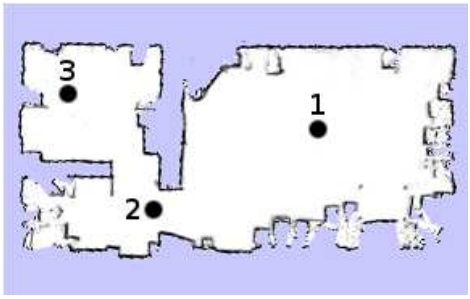


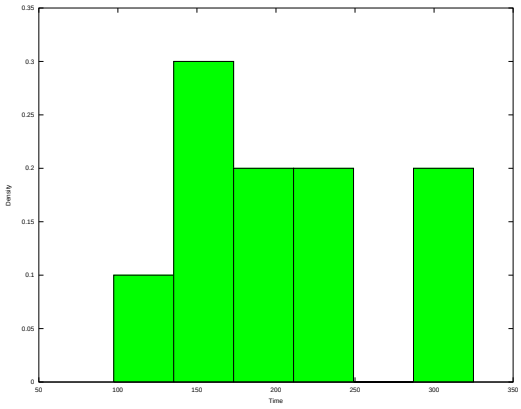
Figure 4.2: The mapped lab used in the robotics experiments.

discovered obstacle.

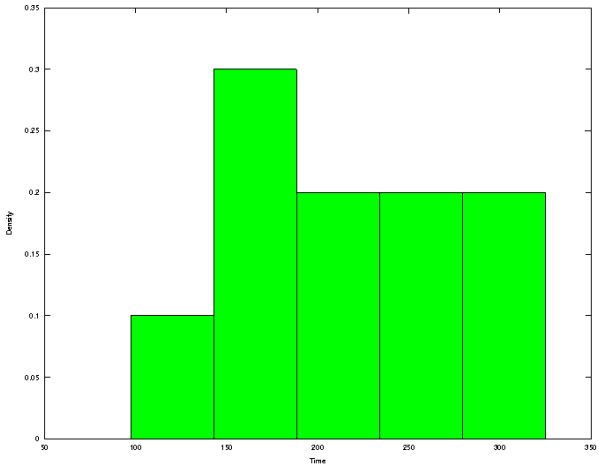
4.1.1 Physical Robot Experiments

We utilized the RV-400 differential-drive robots (see Figure 4.1) for experiments in our lab. The RV-400 was equipped with a Hokuyu UTM-30LX laser, with nominal range of 30m (though in practice effective range was slightly smaller). The RV-400 robot has an approximate size 40×40 (width, length), and so this was used as the grid cell-size. We kept the environment static, with no obstacles or other changes to the environment that are unknown to the robot.

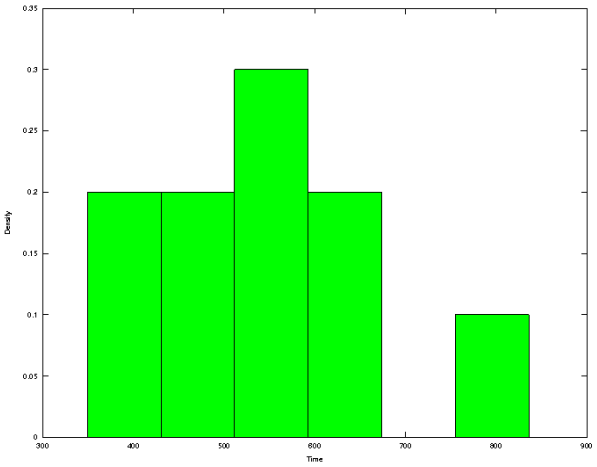
Figure 4.2 shows the environment used for the experiments, as mapped by the robot. We tested three paths: A short 6.4*m* path (point 2 to point 3), with a narrow pass; an 8*m* path (1 to 2), through open space; and a 14.6*m* path which combined both (1 to 3). We measured the travel time in each of these paths 10 times.



(a) 6.4*m* path.



(b) 8*m* path.



(c) 14*m* path.

Figure 4.3: RV-400 Travel Time Distributions.

Figures 4.3(a), 4.3(b), and 4.3(c) show the distribution, in histogram form, of travel time that were measured in these experiments, for the 6.4*m*, 8*m*, and 14*m* paths. For each of the settings, the planned path was identical, and the environment kept strictly static. The associated figure shows the path traversal time

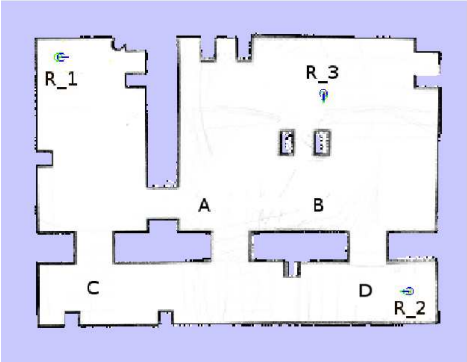


Figure 4.4: The mapped simulated environment.

(horizontal axis) versus its probability (vertical axis). Despite these ideal conditions, the robot took varying amount of time getting to the target locations. This variance is caused because of inaccuracies in the movement and sensing, which lead to actual execution of the path to differ between runs. In addition, changes to battery power also affect the robots linear and angular velocities. Indeed, Figure 4.3(b) does not include four data points that were removed from the data, because in their associated runs the robot operated with a faulty battery, and was almost twice as slow as in the other runs.

4.1.2 Simulation Experiments

We also conducted experiments in simulation, where we scaled up the number and complexity of the paths. We utilized the Webots 3D physics-based robotics simulator [54] to create the virtual world which the robots mapped and navigated as part of the experiments (see Fig. 4.4 for the resulting map). Webots has high fidelity, and models realistic sensor and motion errors, as we demonstrate below. In the simulation experiments, we simulated three RV-400 robots and their Hokuyu lasers. The openings between the rooms are doors which were open or close according to the evaluated criteria. Minor obstacles (boxes to be bypassed) are not shown. The doorway between the rooms is $1.2m$ wide.

The following configurations were used in the simulation experiments: From every robot location, to targets location A, B, C (9 combinations), and robots R_1, R_3

to target location D. we tested 4 obstacle settings: (i) static world (i.e., conforming to the map); (ii) with an unknown obstacle (a box placed on the planned path, that can be avoided and bypassed); (iii) an unexpected closed door blocking the original path (if the path was through an opening); and (iv) two unexpected closed doors blocking the original path, then a re-planned path. Each of the configuration (11 initial-target location pairs, 4 obstacle settings) was repeated 30 times. In all the experiments the robot had a path to the last target.

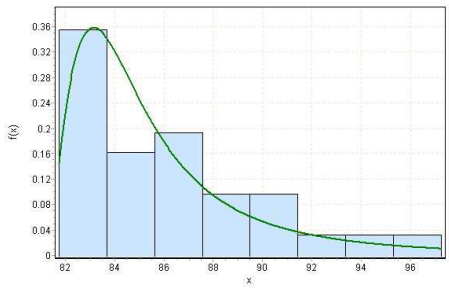
A small subset of the results from the simulation experiments are shown in Figures 4.5(a)–4.5(c) (the rest of the results are discussed later). These are the results for one robot R_1, and for a single target point A. Our intent is to demonstrate the variance that exists even under idealized simulated conditions.

Figure 4.5(a) shows the distribution of traversal times of R_1 for arriving at target A in a static world. As seen in the figure, even for a static world, and even under the relative noise-free world of simulation, there is variance in traversal time, due to motion and sensing uncertainties.

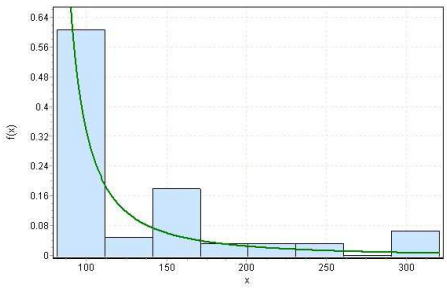
Of course, when choosing a robot for executing a task the world cannot typically be assumed to be static. These increase the variance in the actual travel times. Figure 4.5(b) shows the wider distribution of traversal times when an obstacle was added to the path of the robot, in 50% of 60 cases (the X axis scale is 80 to 350). This obstacle could be locally avoided (bypassed), and thus only a minor change was required to the pre-planned path. Note, that all the distribution of the static environment become a part of the first bin of the new distribution. Figure 4.5(c) shows the even wider distribution when we also take into account a door that was closed in a third of 90 cases, and which blocked the original path. This requires a new path to be planned and executed from the point where the closed door was discovered, to the target location.

The results above are similar to the distributions collected for the other experiment configurations, i.e., for other robots and other target locations. In all cases, even for paths that involve very few heading changes, and no obstacles or narrow passages, we see distributions that require reasoning about which robots to select, given the decision-maker's policy towards risk.

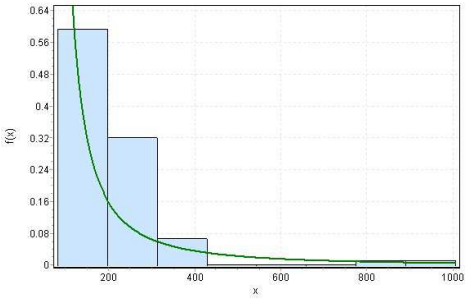
Figures 4.6-4.15 show the distributions collected for the other experiment configurations. The figures describe the histogram as followed: A - static environ-



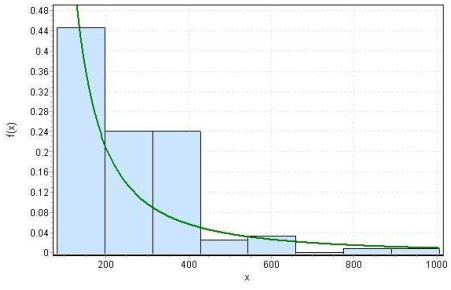
(a) Static environment. The X axis scale is 80 to 100.



(b) Avoidable obstacle in 50% of trials. The X axis scale is 80 to 350.



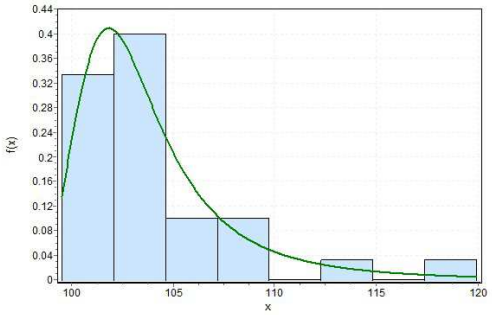
(c) Moving in a static environment, facing occasional avoidable obstacles, and sometimes needing to re-plan a path. The X axis scale is 80 to 1050.



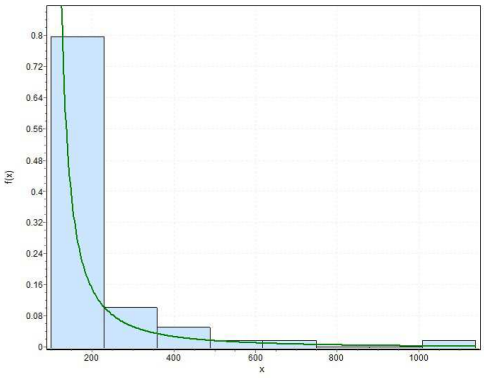
(d) Moving in a static environment, facing occasional avoidable obstacles, sometimes needing to re-plan a path and sometime needing to re-plan a path more than once. The X axis scale is 80 to 1050.

Figure 4.5: R_1 Travel Time Distributions to point A.

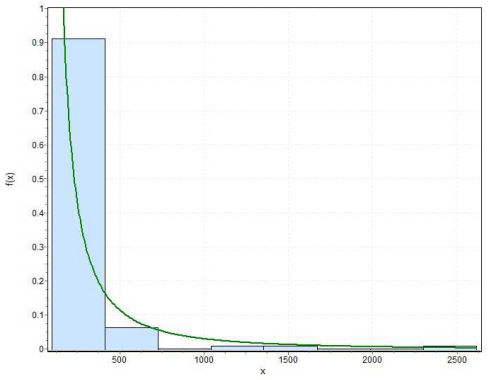
ment, B - Avoidable obstacle in 50% of trials, C - Moving in a static environment, facing occasional avoidable obstacles, and sometimes needing to re-plan a path and D - Moving in a static environment, facing occasional avoidable obstacles, sometimes needing to re-plan a path and sometime needing to re-plan a path more than once. The X and Y axis scales are indicated under the figures. Note, when the robot located in the same room as the target, histograms of types C and D do not exist.



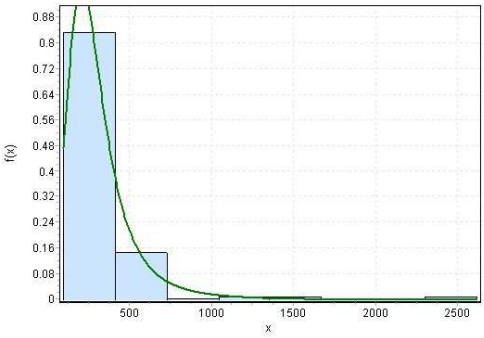
(a) X: 99 to 120. Y: 0 to 0.44.



(b) X: 99 to 1150. Y: 0 to 0.86.

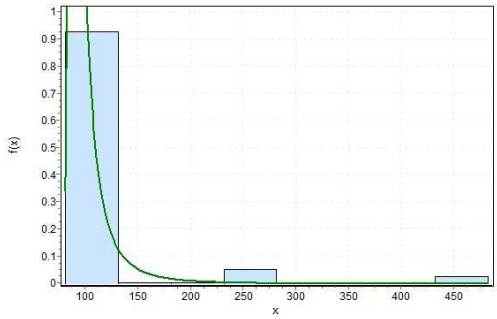


(c) X: 99 to 2625. Y: 0 to 1.

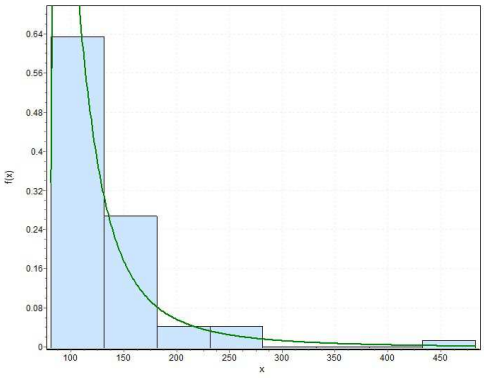


(d) X: 99-2625. Y: 0-0.83.

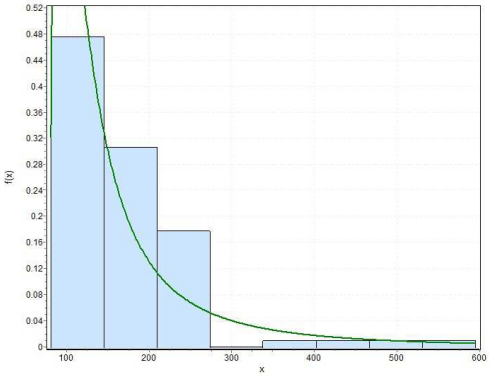
Figure 4.6: R_1 Travel Time Distributions to point B.



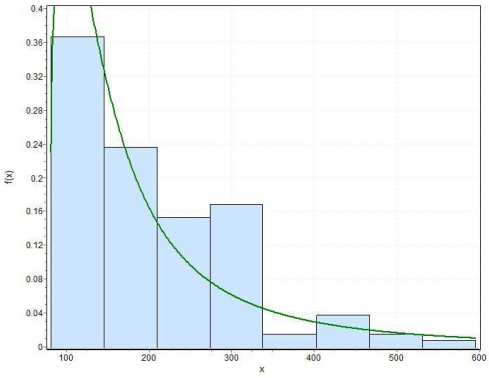
(a) X: 85 to 487.5. Y: 0 to 1.



(b) X: 85 to 487.5. Y: 0 to 0.68.

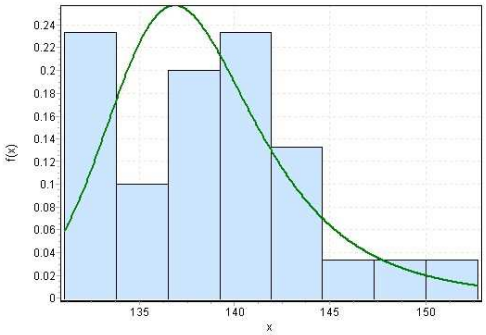


(c) X: 85 to 600. Y: 0 to 0.52.

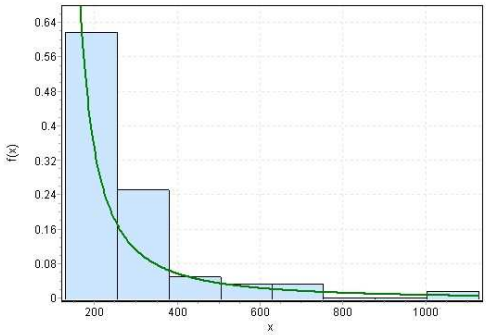


(d) X: 85 to 600. Y: 0 to 0.4.

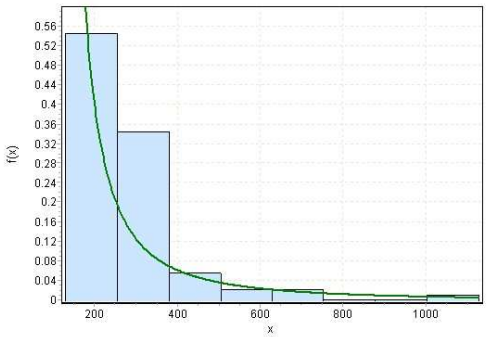
Figure 4.7: R_1 Travel Time Distributions to point C.



(a) X: 130 to 153. Y: 0 to 0.27.

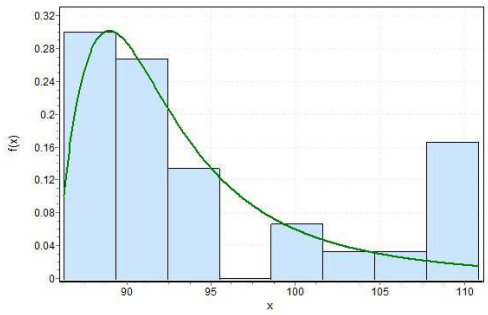


(b) X: 130 to 1127. Y: 0 to 0.66.

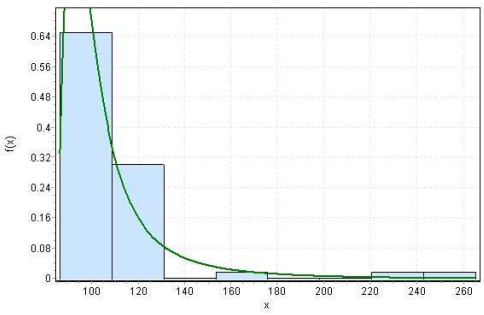


(c) X: 130 to 1127. Y: 0 to 0.59.

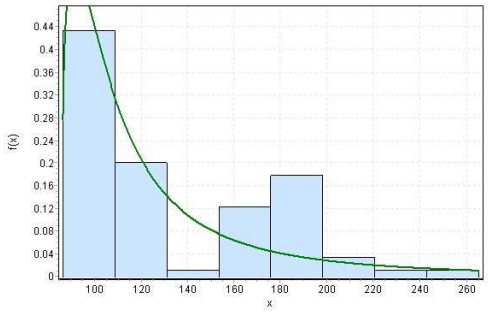
Figure 4.8: R_1 Travel Time Distributions to point D.



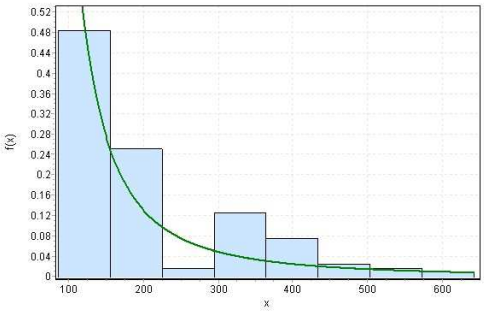
(a) X: 86 to 111. Y: 0 to 0.32.



(b) X: 86 to 265. Y: 0 to 0.7.

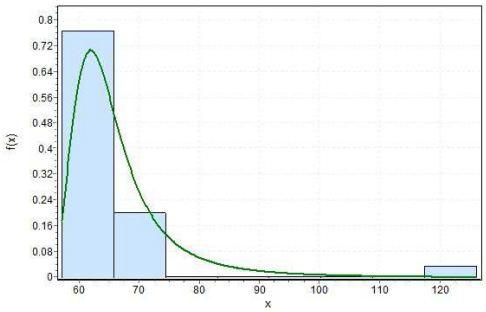


(c) X: 86 to 265. Y: 0 to 0.47.

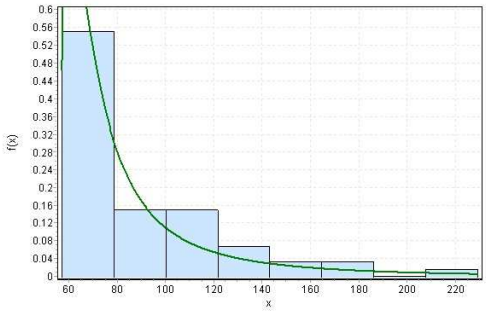


(d) X: 86 to 650. Y: 0 to 0.52.

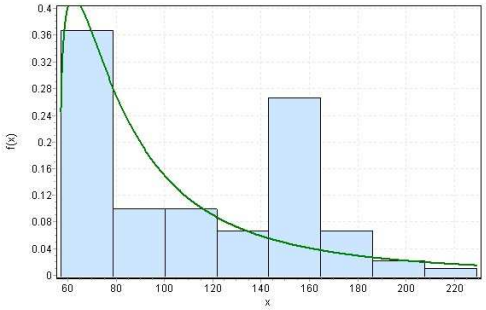
Figure 4.9: R_2 Travel Time Distributions to point A.



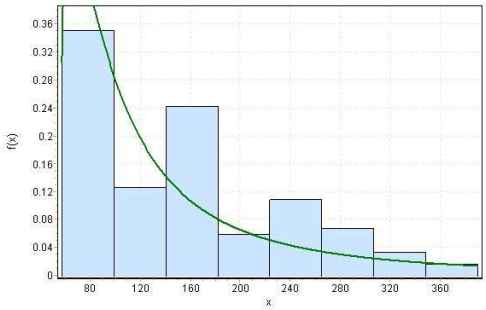
(a) X: 54 to 127. Y: 0 to 0.82.



(b) X: 54 to 230. Y: 0 to 0.6.

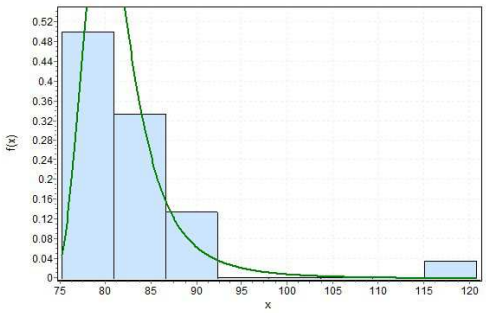


(c) X: 54 to 230. Y: 0 to 0.4.

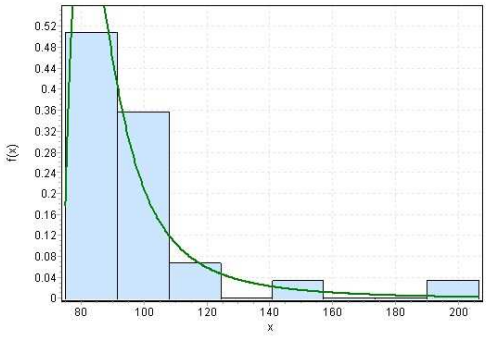


(d) X: 54 to 386. Y: 0 to 0.38.

Figure 4.10: R_2 Travel Time Distributions to point B.

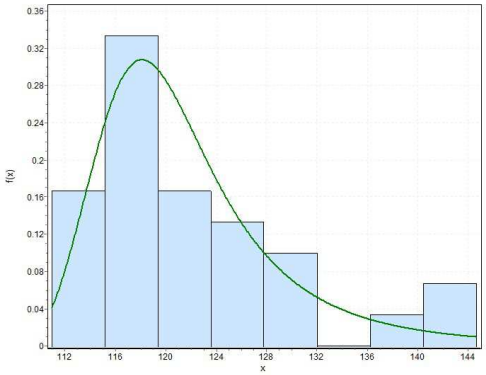


(a) X: 75 to 121. Y: 0 to 0.54.

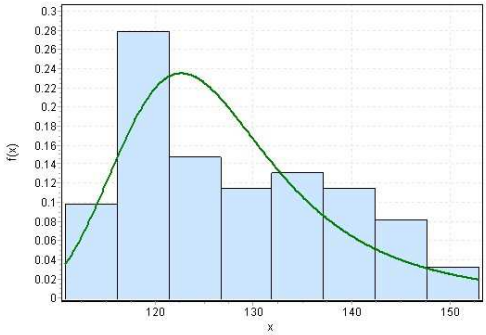


(b) X: 75 to 207. Y: 0 to 0.55.

Figure 4.11: R_2 Travel Time Distributions to point C.

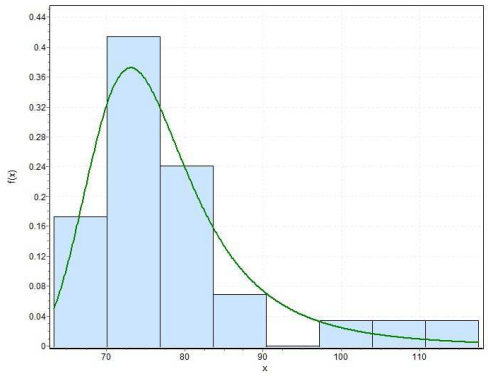


(a) X: 111 to 145. Y: 0 to 0.45.

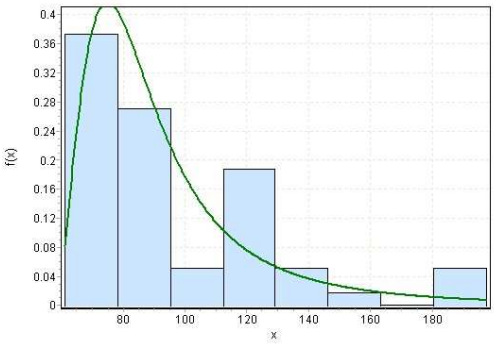


(b) X: 111 to 197. Y: 0 to 0.4.

Figure 4.12: R_3 Travel Time Distributions to point A.

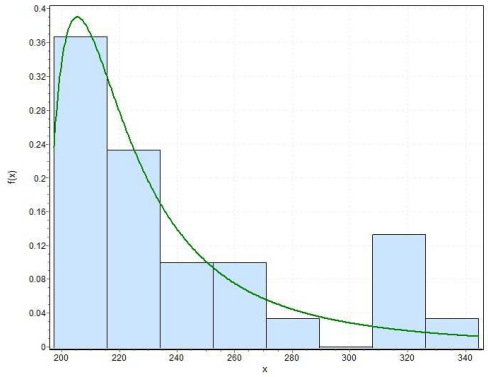


(a) X: 63 to 145. Y: 0 to 0.36.

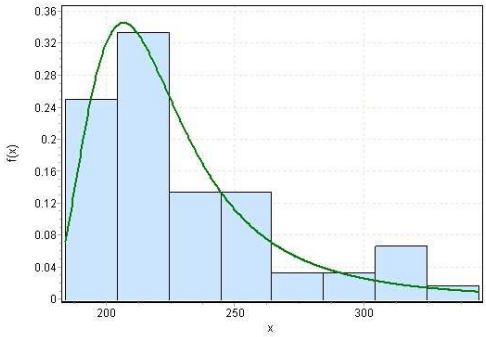


(b) X: 63 to 156. Y: 0 to 0.31.

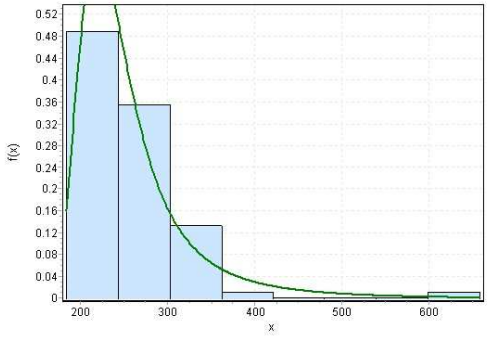
Figure 4.13: R_3 Travel Time Distributions to point B.



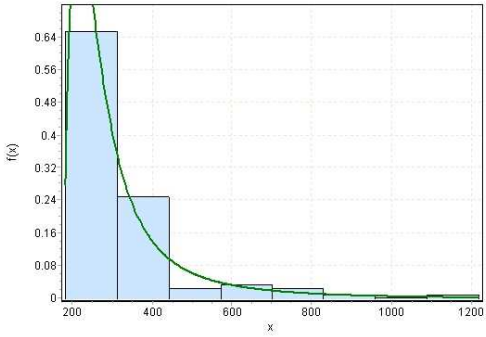
(a) X: 197 to 345. Y: 0 to 0.4.



(b) X: 184 to 345. Y: 0 to 0.36.

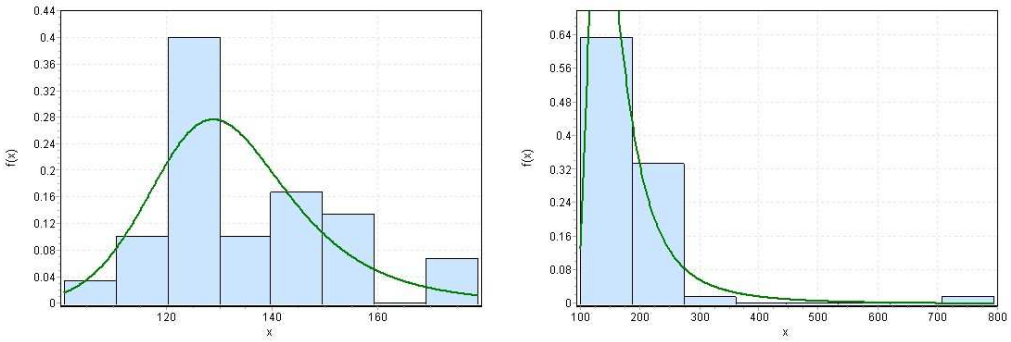


(c) X: 184 to 670. Y: 0 to 0.53.



(d) X: 184 to 1220. Y: 0 to 0.7.

Figure 4.14: R_3 Travel Time Distributions to point C.



(a) X: 100 to 180. Y: 0 to 0.44.

(b) X: 100 to 800. Y: 0 to 0.68.

(c) X: 100 to 800. Y: 0 to 0.46.

(d) X: 100 to 800. Y: 0 to 0.34.

Figure 4.15: R_3 Travel Time Distributions to point D.

4.2 Selection Based on Experiment Data

We use the collected data to execute the decision-making policies described earlier, in both the simulated and physical world. To do this, we discretized the collected data into bins of approximately 25 seconds, and chose the robots according to the different decision policies.

4.2.1 Simulation Experiments

Robots $\{R_1, R_2, R_3\}$ compete on reaching targets A, B, C, and robots $\{R_1, R_3\}$ on target D. Table 4.1 shows the chosen robot using each of the decision policies.

	$MinExp_C$ (risk-neutral)	$MinExpMax_C$ (risk-averse)	$MaxExpMin_C$ (risk-seeking)
A	R_3	R_3	R_1
B	R_3	R_3	R_2
C	R_2	R_2	R_2
D	R_1	R_3	R_3

Table 4.1: Selected robots for targets, according to each policy.

We find that indeed, the selected robot is not always the closest one to the target. For instance, R_3 is closest to point A. But when selecting a risk-seeking policy, R_1 is chosen. Likewise, R_3 is closest to point B, and yet R_2 is selected when a risk-seeking policy. R_3 is also closer to D, yet R_1 is selected in the risk-neutral policy. Moreover, R_3 is selected in the risk-averse and the risk-seeking policies but not is the risk neutral policy (Figures 4.8 and 4.15). This is a direct result of the uncertainty inherent in the robots' movements.

We note that the selected robots for points {A, B, C} in the risk-averse $MinExpMax_C$ criteria were the same as the robots with the minimal expected cost. This is because the robots were in the same rooms with the target locations, and thus the closing and opening of doors–which would otherwise create large worst case travel times (and therefore large expected maximal times)–did not affect the ability of the robots to reach these targets.

Table 4.2 shows a case where an overruling of the selected robot is recommended by the *SwF* function. When selecting which of the robots should reach target A through a risk-seeking policy, both robots R_1, R_2 have a minimal cost of 88. However, robot R_1 is chosen because its probability for this cost is a bit higher.

	p_{88}	E_{SoR}
R_1	0.438679	84.5507
R_2	0.433333	49.2834
R_3	0	29.0444

Table 4.2: The robots expected minimal cost, and expected *SoR* for the minimal cost of 88, while competing on point A.

But looking at the E_{SoR} of the robots, it is clear that R_2 has lower expected regret than R_1. Plugging these values into the *SwF* function yields the following:

$$\begin{aligned}(E_{SoR}(R_1) - E_{SoR}(R_2)) &= 84.5507 - 49.2834 && (4.1)\\ &= 35.2673 && (4.2)\\ &> 0 && (4.3)\\ &= 88 - 88 && (4.4)\\ &= \min_C(R_2) - \min_C(R_1) && (4.5)\end{aligned}$$

In this case, *SwF* returns 1, and we should consider selecting R_2 despite its slightly higher expected minimal traversal time.

4.2.2 Physical Robot Experiments

We utilized the RV-400 data in similar experiments. Abstracting away from the map, we used the distributions for traversal times of $6.4m$, $8m$ and $14.6m$ paths, for three robots: RV_1 positioned $6.4m$ away from a target point, RV_2 positioned $8m$ away from the same point, and RV_3 which is positioned $14.6m$ away. Table 4.3 shows the chosen robot in each of the decision policies.

$MinExp_C$ (risk-neutral)	$MinExpMax_C$ (risk-averse)	$MaxExpMin_C$ (risk-seeking)
RV_1	RV_2	RV_1

Table 4.3: Selected physical robot, according to each policy.

The results show that in the physical world as well, the closest robot is not always the robot to choose. Due to the narrow pass in the $6.4m$ path, the worst case travel time for RV_1 was worse (though less likely) than the worst case of RV_2 (which traveled $8m$ through open space).

4.3 Parametric travel time distributions

The experiments conducted reveal repeating characteristics of the emerging distributions, in particular their sharp lower bound and long tail. This is a result of having a clear lower bound on path traversal time (there's a limit as to how quickly a path can be traversed), and the increasingly rare (but still occurring) long arrival times, due to getting stuck by unforeseen obstacles, decreasing battery levels, etc. On such occasions, robots would re-plan their path several times on the way to the goal, and would sometimes need to traverse long distances to bypass a closed door.

We thus hypothesized that in fact known (parametrized) heavy-tailed continuous distributions may fit the data, allowing for improved prediction. We began experimentally, by fitting familiar distributions to the data, and using the Kolmogorov-Smirnov and Anderson-Darling fitness tests to determine the best-fitting distributions.

The fitness results for the best three distributions are shown in Table 4.4. The table shows the average matching functions, for all the paths that were followed, for the top three matching functions that were found.

The three best-fitting functions were found to be the General Log-Logistic (also called the 3-parameter Log-Logistic distribution), the General Extreme

	Gen. Log-Logistic	Gen. Extreme Value	Frechet $(3P)$
Kolmogorov-Smirnov	0.132	0.135	0.136
Anderson-Darling	1.051	1.512	0.69

Table 4.4: The average fitness of the top three matching distributions using Kolmogorov-Smirnov & Anderson-Darling tests. The lower the number the better the distribution fits.

Value, a limit distribution of the maximum of a sequence of independent random variables which are identically distributed. and Frechet $(3P)$, a special case of the General Extreme Value distribution. The table shows that The General Log-Logistic distribution has the best average fitness using Kolmogorov-Smirnov test, and Frechet $(3P)$ has the best average fitness using Anderson-Darling test. Both of them, however, are strongly related (special cases) of the General Extreme Value distribution. Figures 4.5(a), 4.5(b), 4.5(c) show a curve which is the best-fit Log-Logistic continuous probability distribution fitting the simulation experiments data. In addition, Figure 4.16 shows the travel times distribution of robot R_1, traveling to point B in a static environment, 132 times. As seen is the figure, as the number of experiments grow the log-logistic distribution is a good fit.

We focused on the General Log-Logistic distribution. It has three parameters: shape, scale and shift. The shift parameter was found to be almost perfectly linearly correlated with the minimal travel time of each one on the paths that were traveled. Figure 4.17 shows the relation between the minimal execution time of all the paths that were traveled, and the shift parameter of the General Log-Logistic distribution that was fitted to the histogram of the path travel times. It is clear from the figure that there exist a direct relation between the two.

Looking on the other parameters, we found that the shape parameter was quite steady on values between 0.89304 to 3.3684 while its declaration is $(-\infty, \infty)$. We did not find a consistent value for the scale parameter.

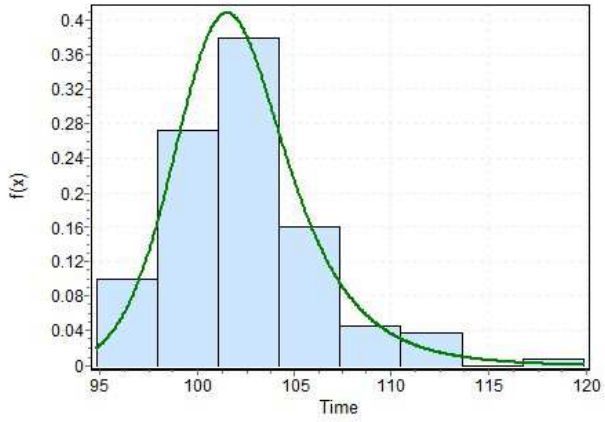


Figure 4.16: Distribution of R_1's travel times to point B in a static environment, over 132 path following experiments. The line shows the fitted log-logistic distribution. The goodness of fit according to Kolmogorov-Smirnov test is 0.0565.

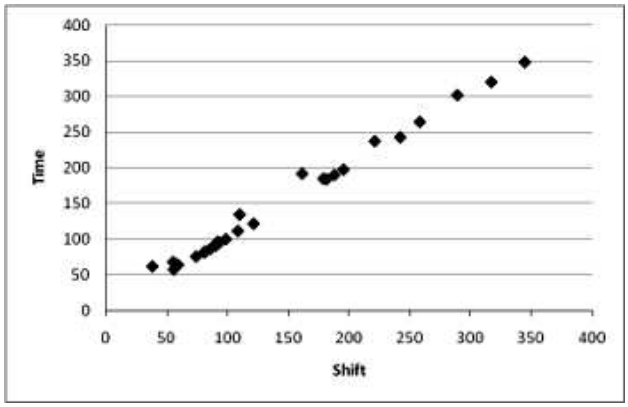


Figure 4.17: Measured minimal travel time versus fitted shift.

Part II

Using Teamwork to Integrate Redundant Multi-Robot Formation Controllers for Robustness

Multi-robot formations are of increasing interest to robotics researchers (as a canonical research problem), and to robot system builders (e.g.,for unmanned convoys). Indeed, there exists vast literature on various techniques for maintaining formations in a variety of settings, and for a variety of robots. However, little attention has been given to the possibility of using multiple formation controllers, all integrated together for greater formation robustness. In this part, we make two contributions. First, we address a key challenge in integration, that of joint distributed selection and execution of the correct controller, at the same time. We demonstrate how to utilize a teamwork software engine to automate this joint selection. Second, we describe one such integrated system, which uses several different formation-maintenance controllers for greater robustness.

Chapter 5

Background: Formations and Teamwork

There exists vast literature on various techniques for maintaining formations in a variety of settings, and for a variety of robots. We cannot hope to cover it all. We therefore provide a sample of important works from this area of research.

In formation-maintenance tasks, the objective is to move a group of robots on a desired path, while they maintain their relative position with respect to their peers, according to a desired geometric shape. Various formation maintenance methods have been investigated (e.g., [2, 13, 53, 51, 27, 3, 14, 56, 9, 39, 46, 23]). We discuss the most popular methods below.

Many controllers assign each robot with a single or multiple neighboring robots (*targets*) that it must monitor to maintain the given geometric shape while moving. Desai et al. [14, 13] show, in theory, how a formation can be maintained if each robot monitors an angle and distance to another robot (separation-bearing control), or distances to two other robots (separation-separation control). They use an un-weighted directed graph, called *control graph*, to describe monitoring from a global perspective. The graph is formed by the set of robots (vertices) and the monitoring relationship (directional edges), from a robot to its target(s). Desai et al. discuss how joint (synchronous) switching of the geometric shape defining the formations (and the associated control graphs) can be used to tackle terrain changes, but do not address how such joint synchronous switching can be

implemented in practice. Our paper focuses on a principled way for implementing controllers in practice, such that synchronized switching can be executed robustly.

In *Separation-Bearing Control* (SBC), each robot maintains its distance (separation) and angle (bearing) to no more than one other robot, called a target. If (i) the control graph is connected; (ii) there exists exactly one target with an out-degree of zero (called *leader*); and (iii) there exists a graph path from each robot to the leader, then maintenance of the control graph leads, over time, to the robots stably maintaining their relative positions in the formation [25, 14, 13].

Another important methods for formation maintenance in practice is *Separation-Separation Control* (SSC). In SSC, each robot (but the leader) maintains its distance with respect to two different targets. If (i) the control graph is connected; (ii) there exists exactly one target with an out-degree of zero (called *leader*); and (iii) there exists a graph path from each robot to the leader, then once again, maintenance of the control graph leads to the robots maintaining a stable formation [25, 14, 13].

For a given geometric formation, combinatorically-large number of stable possible control graphs can exist [14, 13]; this scaleup is exacerbated when a variety of sensors exists. Thus there is a need to efficiently generate good control graphs. Kaminka et al. [46] discuss a general way for efficiently computing *optimal* SBC controllers for a given formation, and a given set of robots and their sensors. This is done by the robots using a version of Dijkstra's algorithm to compute the optimal control graph from a compact hyper-graph representation of all possible control graphs. Mourikis and Roumeliotis [55] discuss optimal sensor scheduling policies for SBC formations, in which sensor use for localization within the formation is optimally balanced between resource consumption (e.g., energy) and localization accuracy.

Fierro et al. [25] analyzed the stability of SBC and SSC controllers, and proposed using manually-constructed control targets to allow up to three robots to switch between alternative SSC and SBC schemes, in essence, switching between alternative control graphs controllers on-line without relying on communications. Our work does use communications (at the teamwork software level), but allows for any number of robots to participate, up to the bandwidth limitations.

Lemay et al. [51] and Michaud et al. [53] present a distributed method for

assigning robots to their positions in a formation. Each robot in a formation determines a cost for assigning its teammates to positions in the given formation, assuming it is the leader (which they refer to as *conductor*). Then the best (minimal cost) assignment of roles to robots (including the leader) is made. This type of negotiations over roles and position assignments is easily modeled and executed in teamwork software, as we demonstrate in this paper. However, we did not experiment with leader assignment, only with dynamic assignment of robots to follower roles. Furthermore, the work by Lemay et al. and Michaud et al. allows switching of the formation shapes, which we do not address in this paper.

In general, there are other control methods for formations. Maintaining formation while moving requires the robots to locate themselves according to reference points. We discuss several different methods below.

Balch and Arkin [2] examine three techniques for formation maintenance. Two of these (*Leader-Referenced* and *Neighbor-Referenced*) techniques are essentially SBC controllers, using static (fixed) control graphs. The third, *Unit-Center-Referenced*, is fundamentally different. Here, the robots place themselves according to X, Y coordinates defined by the formation. Balch and Arkin's study compares between the methods using teams of up to four physically homogeneous robots, and attempts to draw conclusions as to their relative benefits. Our approach builds such comparisons to allow switching between different types of control as best fits the immediate needs of the robots.

Balch and Hybinette [3] use social potential fields which use attraction and repulsion to position robots within their relative positions in a defined formation. This technique is robust to obstacles in the path of the robots, an important challenge our approach does not yet take into account.

Fredslund and Matarić [27] describe an algorithm for generating SBC monitoring rules for robots in a given formation. The robots are assumed to have specific sensing capabilities, and the position of the leader is given. The monitoring rules are supplemented by communications for robustness, thus fusing SBC and communications-based control. Elmaliach and Kaminka [23] build on this to experiment with different ways of integrating communications and SBC controllers (switching between them and/or fusing their actions). However, neither investigation discusses integration of multiple controllers in general, and both ignore

the requirement for a principled way for the robots to jointly select their control scheme. In this paper, we describe how to use teamwork software to automate the joint online selection of controllers by the robots.

Teamwork software (sometimes referred to as "teamwork engine") has been been discussed in the multi-agent systems and multi-robot systems literature. Its beginnings are in the use of formal logic to describe ideal collaboration between agents [36, 12, 35]. These theoretical investigations have discussed a number of principles for collaboration between abstract agents, and in particular point out the importance of team-members agreement on a goal to be reached, and a plan by which to achieve it.

Theory inspired implementations of software frameworks that facilitate the development of distributed multi-agent systems that collaborate towards a joint goal, via an agreed-upon plan of execution [40, 64, 65]. While many of these systems have been applied in simulations and virtual environments (see, e.g., [58, 22], there have been a few that have been utilized with robots.

BITE (Bar Ilan Teamwork Engine) is a behavior-based distributed teamwork control software, specifically targeting multi-robot teams [44, 45, 47]. While BITE has been used with multi-robot formations [45], it has only been previously used with a single type of controller (SBC). CogniTAO [11] is a commercial teamwork software development kit, which can be similarly used to develop multi-robot applications. We use CogniTAO in this paper.

Goldberg et al. [32, 33] have explored a different basis for multi-robot architectures. Rather than taking a behavior-based approach, Goldberg et al. focus on extending a 3-tier architecture with an impressive set of capabilities, including task-sequencing and task-allocation, and distributed resource management. We believe that the lessons in this paper regarding the use of synchronized task sequencing and joint selection of tasks can be used in their architecture (which has not been applied to multi-robot formations).

There have been many other multi-robot investigations which focus on automating interactions between robots. However, they mostly focus on task allocation, rather than synchronization and joint selection of activities.

Among those, the ALLIANCE behavior-based architecture [57] focuses on robustness, by allowing robots to dynamically re-allocate themselves to tasks, based

on failures in themselves in their teammates. ALLIANCE offers robust dynamic task allocation, but does not explicitly synchronize robots as they jointly take on tasks.

Other systems have focused on using auctions and market-based task-allocation methods in multi-robot systems. TraderBot [19] explored the use of markets to allow robots to bid for tasks in spatial sensing domains. Goldberg et al. [33] explore a distributed three-tier architecture, in which multiple robots interact with each other at all three layers using market-based resource allocation. Gerkey and Matarić [29] discuss the use of such methods in contrast to others.

Farinelli et al. [24] explore novel methods for task allocation in robot teams. Their token-passing method are suitable for teams of larger scale than those discussed in this paper. It should be possible to use this token-passing allocation mechanism for use in large-scale formations.

Jung and Zelinsky [42] have explored the use of a distributed architecture, which is behavior-based. However, the focus of this work is on cooperative spatial planning, and no synchronized joint selection of controllers. Alur et al. [1] offer a comprehensive framework for spatial coordination of multiple robots, and have applied it to formation control, but only using a single type of formation controller at a time (in particular, SBC). They did not address joint synchronized switching between alternative formation controllers.

Chapter 6

Teamwork Software for Joint Formation Control

First, we briefly describe the principles of behavior-based teamwork software. Then, we describe a simple separation-bearing control (SBC) scheme, implemented using such teamwork software.

We utilize a behavior selection mechanism, in which behaviors are proposed and selected based on their preconditions matching the current world state (as perceived by the sensors); once a behavior is selected, its execution is terminated when its termination conditions are satisfied. We describe this process below briefly, and refer the reader to [64, 44] for additional details.

A behavior-based teamwork controller is divided into two computational components. The first component is a *world-modeling process*, which is responsible for processing sensor and communication data, and for sharing this information (when needed) with other team-members. The second component is the *control process*, which runs the control process behavior-manager algorithm (selecting and deselecting behaviors). These two components are described below. For the purpose of clarity, however, we begin by describing the control process (Section 6.1) and only then describe the world-modeling process (Section 6.2).

6.1 Control Process

A *task behavior graph* specifies the sequential and hierarchical relationships between task-oriented behaviors. Each behavior is a semi-independent module, responsible for a portion of the overall task.

Formally, a task behavior graph is an augmented connected graph tuple $\langle B, S, V, b_0 \rangle$, where B is a set of task-achieving behaviors (as vertices), S and V are sets of directed edges between behaviors ($S \cap V = \emptyset$), and $b_0 \in B$ is a behavior in which execution begins. Each behavior in B may have preconditions which enable its selection (the robot can select between enabled behaviors), and termination conditions that determine when its execution must be stopped. S is a set of *sequential* edges, which specify temporal order of execution of behaviors. A sequential edge from b_1 to b_2 specifies that b_1 must be executed before executing b_2. A path along sequential edges, i.e., a valid sequence of behaviors, is called an *execution chain*. V is a set of vertical *task-decomposition* edges, which allow a single higher-level behavior to be broken down into execution chains containing multiple lower-level behaviors. At any given moment, the robot executes a complete path—root-to-leaf—through the behavior graph. Sequential edges may form circles, but vertical edges cannot. Thus behaviors can be repeated by choice, but cannot be their own ancestors.

Behaviors whose execution is to be coordinated in some fashion (henceforth, *team behaviors*) are tagged in advance by the designer. Each robot executes a local copy of the behavior graph [64, 57, 33, 44]. It is the responsibility of the teamwork architecture to automatically takes actions (typically, by communications) to select and de-select these in different robots, when appropriate [64, 44]. Figure 6.1 shows an example of a simple behavior graph, constructed for multi-robot formation maintenance tasks using SBC control.

The teamwork architecture selects and de-selects team-behaviors by using communications to make sure that relevant details of the robots *collaborative world-model* (Section 6.2) are synchronized, and by applying negotiation procedures—decision-making protocols—to ensuring agents jointly select their next behavior, and jointly terminate its execution once selected.

To allow a teamwork architecture to automate synchronization, we impose a

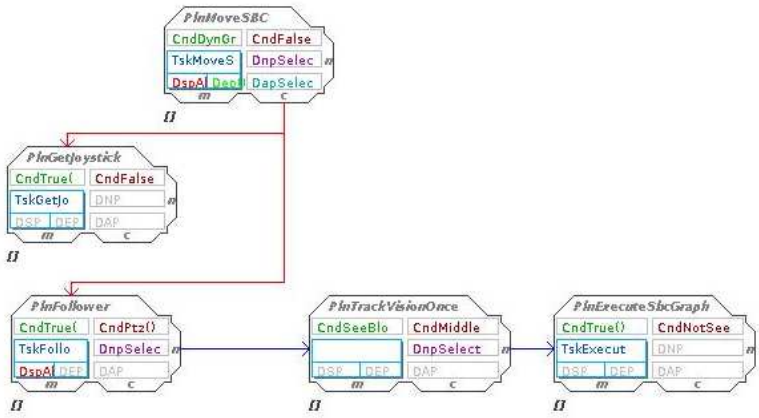


Figure 6.1: A Behavior graph for simple SBC control. Each robot runs its own local copy of this graph. Node names appear at the top of each node above. Other text refers to names of conditions and protocols utilized in the different nodes, and described in the paper. Arrows coming out of the tab marked ‘c‘ are task-decomposition edges, while those coming out from the tab marked ‘n‘ are sequential ordering edges.

constraint on the semantics of multiple outgoing edges. Two outgoing sequential edges $\langle a,b\rangle, \langle a,c\rangle$ signify a choice point between *alternative* execution chains: Either b or c must be selected by the robot once its execution of a is finished. When these execution chains are composed of team behaviors, the selection between alternatives must be coordinated—all (relevant) robots must select the same execution chain (we discuss below complex cases in which only certain subteam members must coordinate). Thus synchronization (see below) is triggered when multiple execution chains are enabled, and the robots must coordinate their joint selection.

To automate allocation, we impose a related semantic constraint on decomposition edges. Two outgoing decomposition edges $\langle a,b\rangle, \langle a,c\rangle$ signify *complementary* execution chains: Both the execution chain beginning with b and the execution chain beginning with c must terminate for a to be considered complete (by convention, vertical edges point only to the first behaviors of execution chains). Thus such multiple outgoing edges indicate that the children (subtasks) can be allocated to different subteams. Therefore, similarly to the synchronization points,

allocation services are triggered when multiple decomposition edges are enabled.

There is one final point in which synchronization is needed. Teamwork theory states that when an agent privately believes that a joint goal has been achieved, or should be abandoned, the agent must make this belief mutual with its teammates. The communication of beliefs is handled by the collaborative world modeling process (next section). Here, the implication is that robots must terminate their execution of team behaviors in a coordinated fashion. Thus when a team behavior's termination conditions are satisfied for a robot, BITE is triggered to coordinate the termination of this behavior with the other robots.

To summarize, a teamwork architecture can easily determine synchronization and allocation points given the constraints above. A split in sequence edges leading to team behaviors signifies a synchronization point. A split in decomposition edges leads to allocation, and synchronized termination is triggered when a team behavior is de-selected. In all of these, the architecture must coordinate with the other robots, through their own local instances of the system.

6.1.1 Principal Control Algorithm

Each of the robots executes Algorithm 3, using its own copy of the behavior graph. The control loop executes *a behavior stack*—root behavior to leaf—where top behaviors on the stack are executed simultaneously with their currently selected children.

Execution begins by pushing the initial behavior of the graph on the execution stack (lines 1–2). Then the algorithm loops over four phases in order. (i) It recursively expands the children of the behavior, allocating them to sub-teams if necessary (lines 3a–3c). (ii) It then executes the behavior stack in parallel, waiting for the first behavior to announce termination (lines 4a–4c). All descendants of a terminating behavior are popped off the stack (i.e., their execution is also terminated—line 4b), and then (iii) a synchronized termination takes place (line 6). This can result in a newly-allocated behavior within the current parent context, in which case, it will be put on the stack for expansion (line 7). Otherwise, (iv) this indicates that the robot should select between any enabled sequential transitions from the terminated behavior (lines 8a–8e). This process normally results in

new behaviors put on the stack, and then a final goto (line 9) back to line 3 begins again.

The recursive allocation of children behaviors to sub-teams in lines 3a–3c relies on the call to the *Allocate*() procedure. It takes the current execution context (i.e., current stack, available children), and then calls the appropriate allocation protocol (defined by the architecture [64], or by the programmer [22, 44, 45]) to make the allocation decision. The current execution stack is used to help guide allocations, e.g., by conveying information about where in the behavior graph the allocation is taking place. Once a final allocation is determined, *Allocate*() returns, for each robot, the child behavior for which it is responsible as part of the split sub-team (or individually, if the sub-team is composed only of the individual robot).

Synchronized termination (line 5–7) and selection (lines 8a–8e) similarly rely on calls to the procedures *Terminate*() and *Decide*(), respectively. *Terminate*() is responsible for evoking the execution termination interaction behavior, which can return a new child behavior for execution under the current parent. If it doesn't, then the next behavior in the execution chain must be selected by *Decide*(), which calls a synchronization protocol. Since synchronized selection involves all members of the current sub-teams selecting together, this behavior would normally communicate with the members of the team. Note that in step 8b we also handle the case where no more behaviors are available in the execution chain. This case signals a termination of an execution chain, which in turn signals termination of the parent, thus the branching back to line 5. We omitted here the obviously needed check on whether a parent actually exists—if not, then the end of the behavior graph has been reached, and execution halts.

6.2 Collaborative World Modeling

Recent behavior-based architectures, inspired from advances in cognitive architectures in AI, introduced the world-model as a separate computational process. This process carries out both traditional sensor-filtering and processing, it also executes collaborative algorithms, which share information with other robots. We focus on this aspect here.

Algorithm 3 CONTROL

Input: behavior graph $\langle B, S, V, b_0 \rangle$

1. $s_0 \leftarrow b_0$ // initial behavior for execution
2. push s_0 onto a new behavior stack G.
3. while s_0 is non-atomic // has children
   (a) $A \leftarrow \{b_i\}$, s.t., $\langle s_0, b_i \rangle$ is a decomposition edge
   (b) if A has only one behavior b, $push(G, b)$.
   (c) else $b \leftarrow Allocate(G, s_0, A)$, $push(G, b)$.
   (d) $s_0 \leftarrow b$.
4. execute in parallel for all behaviors b_i on G: // Execution
   (a) execute b_i until it terminates
   (b) while $b_i \neq top(G)$, $pop(G)$
   (c) break parallel execution, goto 5.
5. $b \leftarrow pop(G)$ // Terminate joint execution
6. $c \leftarrow Terminate(G, b)$
7. if $c \neq NIL$, $push(G, c)$
8. else: // Select next behavior in execution chain
   (a) Let $Q \leftarrow \{s_i\}$, s.t. $\langle b_0, s_i \rangle$ is a sequential edge
   (b) if Q is empty, goto 5 // terminate parent
   (c) if Q has one element s, $push(G, s)$
   (d) else $s \leftarrow Decide(G, b_0, Q)$
   (e) $s_0 \leftarrow s$
9. If G not empty, goto 3.

While protocols may exchange information as needed (e.g., votes and vote outcomes), there are more basic communication needs that underlie behavior execution. To illustrate, consider the following example: Suppose a robot has determined that a running behavior is to be terminated (because its termination condition matches). A call is made to a synchronized-termination protocol. But as it contacts the other robots, it refers to information that is known only to the robot initiating the dialog. Obviously, the termination protocol can be fixed such that it first transmits the missing information, and then argues for termination. But rather than duplicate this functionality in all termination protocols, it makes sense to allow this to happen—in a flexible manner—in the world-modeling process.

Indeed, the world-modeling process can execute distributed information-sharing algorithms. The algorithms can be as simple as an algorithm that updates all team-members with any change in perception; or it could be complex and sophisticated, able to consider uncertainties in fusing information about the world [63].

A naive approach broadcasts all changed information. For obvious reasons of bandwidth usage, it is ruled out in favor of a more focused algorithm, which broadcasts information about changes in the preconditions and termination conditions of currently-executing behaviors of team-members. This agrees with theoretical notions of teamwork, which specify that team-members that privately come to belief a proposition relevant to the team, must establish mutual belief in this proposition [12]. Thus such information is only broadcasted to relevant team-members (i.e., to team-members currently executing a behavior affected by the new information). The algorithm appears below (Algorithm 4).

Algorithm 4 FUSEINFORMATIONWITHTEAMMATES

Input: behavior graph $\langle B, S, V, b_0 \rangle$, set of robots t

1. for all behaviors b on behavior stack G:
   (a) if a termination condition c of b is satisfied,
   Inform(b, t, c)
   (b) if a precondition p of a behavior f, where $\langle b, f \rangle$ a sequence edge, is satisfied,
   Inform(b, t, p)

In Algorithm 4, each robot determines whether new information affects its behavior stack (e.g., newly-satisfied conditions). These potentially affect the robot's teammates, and must therefore be communicated to them by the *Inform*() procedure, which refers to an appropriate protocol for communicating relevant information to others.

For instance, suppose a team of robots is executing the formation task described above (Figure 6.1). Suppose that the robots are executing the *MoveSBC* behavior. One robot (the leader) is executing the *GetJoystic* behavior, while the other robots are executing the *Follower* behavior. Algorithm 4 is used to informed all the robots in the sub-team that the pre-condition of executing the behavior together is satisfied. The two next behaviors of the followers *TrackVisionOnce* and *ExecuteSbcGraph* are being executed in a chain as soon as the termination condition informed. Algorithm 3 guarantees that if one of the following robots behavior *ExecuteSbcGraph* terminates, than the termination condition of *MoveSBC* behavior will be satisfied as well. Algorithm 4 guarantees that if the robot discovers a termination condition for *MoveSBC*, then it will inform the members of the subteam associated with *GetJoystic* and *ExecuteSbcGraph*.

Chapter 7

Integrating Multiple Controllers

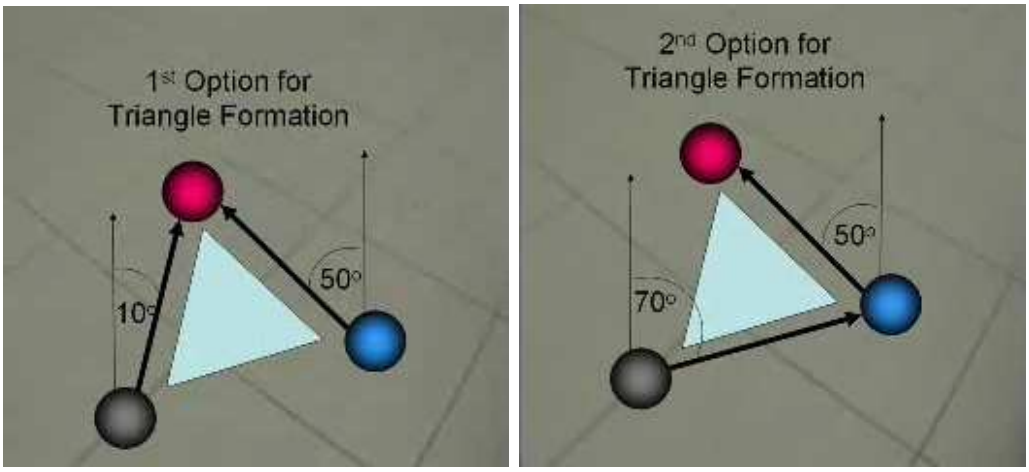
We now turn to discussing various controllers which we integrated together. Section 7.1 describes the implementation of switching separation-bearing control (SBC) [46]. Section 7.2 describes the implementation of communication-based formation control [23]. Finally, Section 7.3 ties these two sophisticated controllers together.

7.1 Robust Formations by Switching SBC Formations

Glick-Schecter et al. [46] have shown that for robustness, a team of robots moving in formation may want to switch its SBC control graph to allow robots to switch their targets (i.e., which robots they monitor), when their sensing of the other robots fails or becomes too costly. For example, the triangle formation may be maintained by either of the following control graphs (Figure 7.1).

The robots continuously monitor the cost of maintaining their target (corresponding to the rate of sensing failures), and when either robot decides that this cost has reached a given threshold, a new control graph should be utilized. In this case, the robots are to jointly select a new control graph, as described in [46].

This process is easily modeled in behavior-based teamwork (Figure 7.2). It works as follows: all the robots start the behavior *PlnBlind* together and the *PlnMovePtzToStart* under it. *PlnMovePtzToStart* initializes all the robots cameras to



(a) (b)

Figure 7.1: Different control graphs for triangle formation.

a fixed starting point. By executing the behavior *PlnUpdateSBC*, a scan of the area in carried out, in order to find all the visible robots. Each time a robot is located, the scanning robot sends the relative location of the robot to the other robots. In this way a matrix of the visible robots is created. Each robot knows the relative angle and distance to the other robots. The behavior terminates after the camera scanned 180 degrees. Using the matrix the next behavior *PlnRunDijkstra*, runs Glick-Shechters algorithm to build a graph of visible robots. After running once the behavior terminates. Because the SBC graph need to be connected, if the formation graph becomes disconnected the *PlnBlind* behavior will be re-executed, as evidences the sequential edge from itself back into itself.

7.2 Robust Formations by Communication-Based Formation Control

A different approach for maintaining robust formation has been described by Elmaliach and Kaminka [23]. Here, the agents utilize communications and dead-reckoning, rather than SBC, for formation control. The controller multiplexes and fuses between open-loop and close-loop in order to prevent disconnection caused by failures of the robots sensors and increase the number of discoverable obstacles

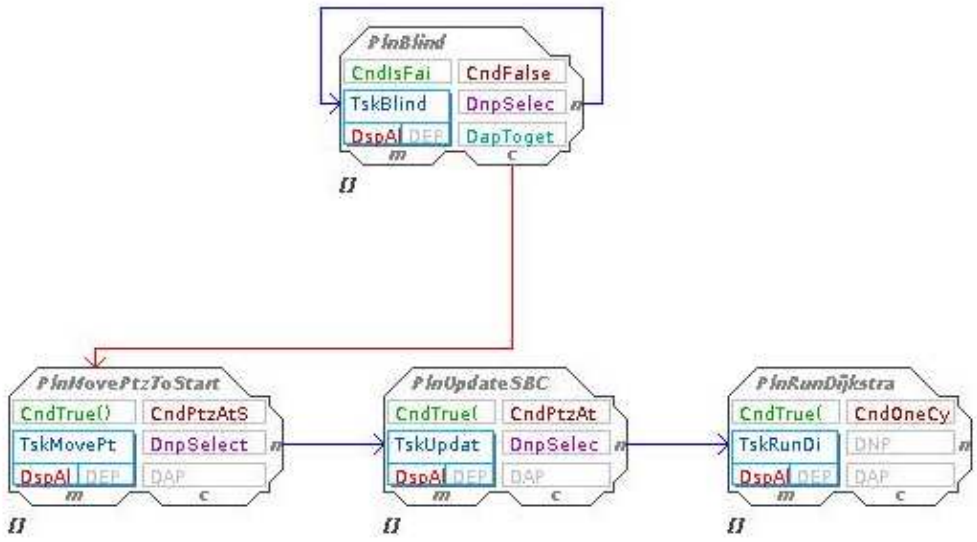


Figure 7.2: A Behavior graph for the switching controller.

in the environment.

This process is modeled in the following way (Figure 7.3). All the robots are executing the *PlnMoveSBC* together. This behavior runs allocation protocol, *DapSelectFirst*, which choses the robot with the lowest id to be the leader and allocates it to the *PlnGetJoystic* behavior, while allocating the other robots to execute the *PlnFollower* behavior. *PlnGetJoystic* behavior is responsible for getting the drive commands from the user, while the *PlnFollower* starts the formation tracking. The two next behaviors of the followers *PlnrackVisionOnce* and *PlnExecuteSbcGraph* are being executed in a chain as soon as the termination condition is satisfied. But in contrast to the simple formation control, the *PlnExecuteSbcGraph* behavior does not controls the tracking using vision, but two more parallel behaviors run as well, *PlnTrackVision* and *PlnTrackComm*.

TrackVision, is responsible for calculating distance and angle to the leading robot by vision recognition, while the *TrackComm* executing track by communication between the robots and synchronization of the wheels tics. Thus, the *ExecuteSbcGraph* fuses and multiplexes between the results of the two behavior.

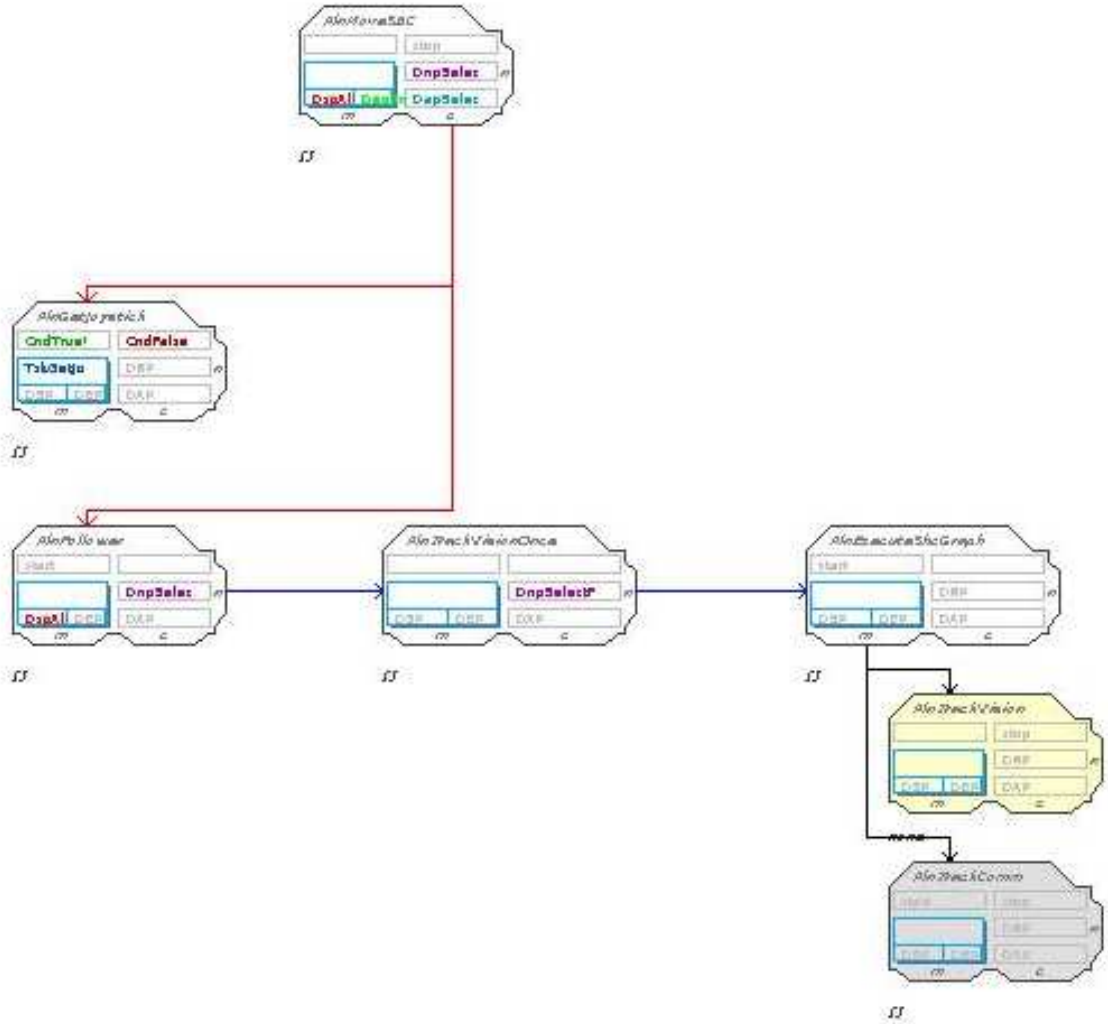


Figure 7.3: A Behavior graph for communication-based controller.

7.3 Integrating Controllers

Using a teamwork architecture, we can easily tie the different controllers together, to achieve enhanced robustness. We do this by constructing a behavior graph that contains the others, with appropriate selection and termination conditions.

A complete behavior graph containing the two previously discussed controllers is presented in Figure 7.4. Here, the execution starts by initialize the graph to *triangle formation* and *line formation*. Execution begins with triangle formation, and can (under specific conditions) switch to the line formation. Both formations use one behavior–*MoveSBC*– which implements the SBC graph controller presented above (Section 7.2).

This controller terminates as a result of a lost of at least one of the robots in the graph. In this case the behavior will terminate and the next behavior will start to execute.

The *Blind* behavior executes the second controller (section 7.1), in which a new SBC graph is build. In this case all the robots stop in place, synchronize and start the new controller together. As explained above, the controller ends when a new SBC graph is created. If the new graph is founded to be a connected graph the next chosen behavior will return to be *MoveSBC*, otherwise the allocation protocol will start the controller again in order to look for a new connected graph.

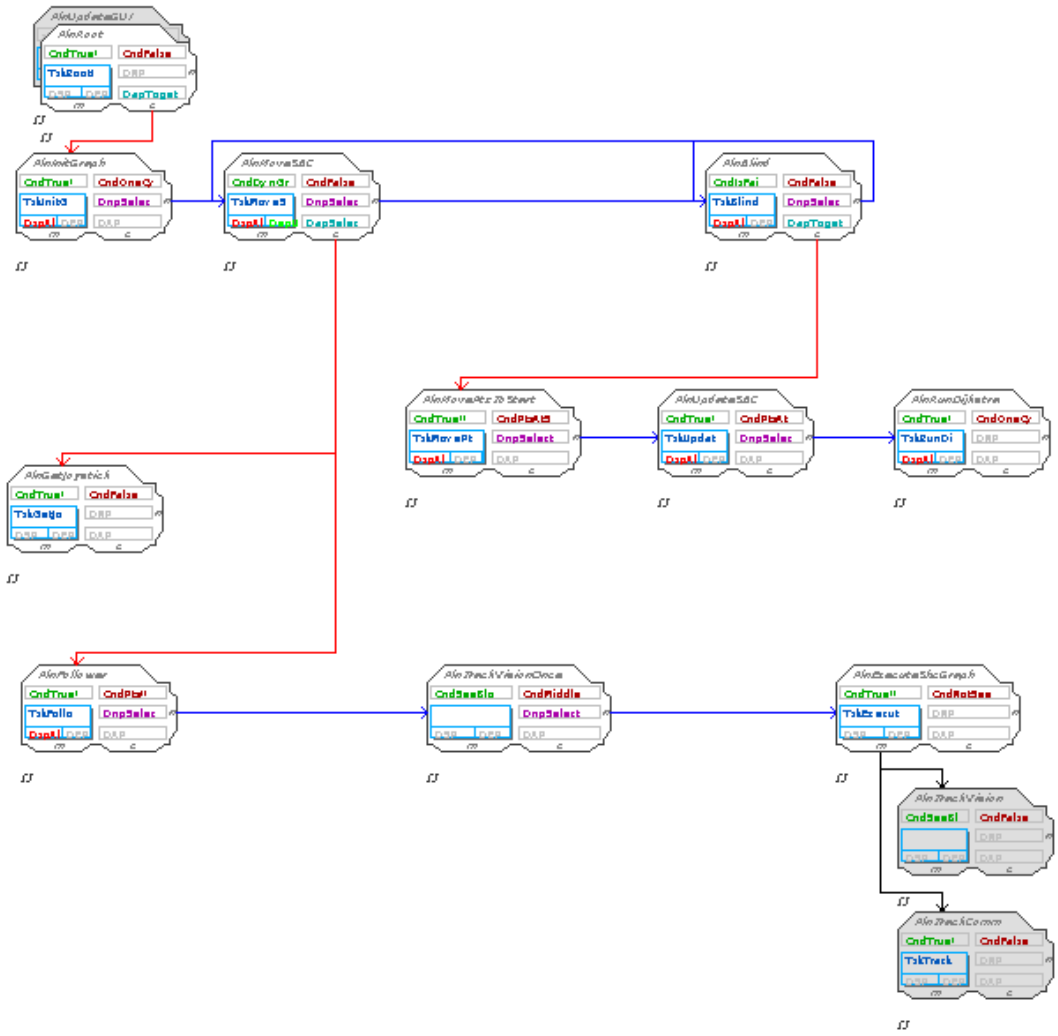


Figure 7.4: The complete Behavior graph.

Chapter 8

Discussion and Evaluation

Evaluation of the use of a teamwork behavior-based architecture for formations is challenging, as we are not interested in the evaluation of the component controllers per-se, and not even in their combined strengths (as these are known). Rather, the principal hypothesis underlying the use of the architecture is that it facilitates deployment in some fashion. For instance, we may wish to argue that the use of the architecture saves programming effort (by reusing code, or automating procedures that were previously manually built), or increases robustness. Of course, there are also limits to the usefulness of using a teamwork architecture as we have, and we discuss these as well.

8.1 It Works!

We fully implemented the behavior graphs and controller described above, using the CogniTAO commercial behavior-based teamwork architecture [11]. The system was deployed with three Blue-Botics Shrimps III robots (shown in Figure 8.1), and also in the Stage simulator (shown in Figure 8.2). The robots utilized IEEE 802.11g communications, as well as Sony PTZ cameras (EVI-D100P model) and Hokuyo lidars (URG-04LX model). Each robot was controlled by a VIA C7 CPU, utilizing 512MB ram.

A key component in the switching SBC formation controller is that the robots need to uniquely identify each other. Relying on the robots' cameras, we used

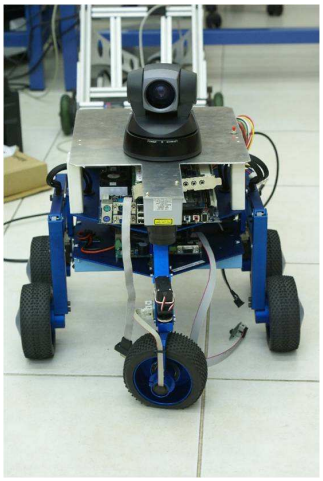


Figure 8.1: The Blue-Botics Shrimps III robot.

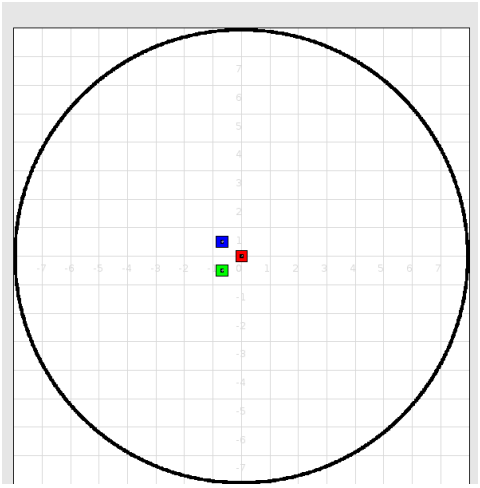


Figure 8.2: Three shrimps robots in Stage simulator.

Figure 8.3: Three shrimps robots in an indoor environment.

color segmentation in indoor settings (see Figures 8.3–8.4). In outdoor settings, we utilized the ARToolKit package for specific pattern recognition, with each robot having its own unique visual pattern (Figures 8.5–8.7).

8.2 Robustness

The use of teamwork architecture gives the ability to integrate different controller in a simple way and increase the robustness of the formation. Each of the controller described above was proposed to address a different set of potential failures. Their combination, made possible by the use of the teamwork architecture, provides robustness against a wide variety of failures.

The communication-based controller was designed to solve problems of short-duration intermittent failures to visually recognize a target robot, by allowing a follower robot to follow its target blindly. The target transmits information as to its movements; the follower translates these into its own target coordinates, and moves accordingly. We found such failures occur often in practice, especially

Figure 8.4: Three shrimps robots in an indoor environment.

Figure 8.5: Three shrimps robots in an outdoor environment.

Figure 8.6: Three shrimps robots in an outdoor environment.

Figure 8.7: Three shrimps robots in an outdoor environment.

when encountering a small vertical obstacle (when the target robot passed over the top, or started climbining from the low point, it becomes temporarily invisible to the followers, whose vertical heading is in an opposite direction).

On the other hand, the switching SBC controller was designed in order to solve longer-duration sensing failures, e.g., caused by permanent loss of the identification pattern, or relatively long visual loss when a target lead took a sharp turn. In these cases the robots look for a new SBC control graph in order to keep the formation structure.

By integrating these controller, we save each controllers robustness abilities and make the system robust to all the problem described above. If a robot loses its vision abilities temporarily, the use of the communication-based controller works to maintain the formation for a short duration using communications from the lead robot. If and when this fails, the switching SBC controller will start and a new connected graph will be build in order to maintain the formation.

Moreover, the use of teamwork architecture increase the robustness by adding new features to the system:

- In the case of a robot-death failure, the teamwork architecture enable the user to remove the robot from the team and continue working with a smaller team of robots.

- All robots start and stop together. A key built-in feature of the teamwork architecture is that information privately available to any individual robot is broadcast to its peers, automatically, if it causes a change in the behavior (i.e., if it causes a condition to be satisfied). As a result, any robot that needs to stop (e.g., because it loses its place in the formation) automatically tells all others to stop. And likewise, when the leader starts moving, it automatically tells the others to move; they don't wait for perception of its movement. As a result, the caterpillar effect often observable in convoys and other formations is gone.

- Because of the above, follower robots are no longer ignored. In typical formation maintenance, the formation is susepctible to robot-death and robot kidnapping in the followers. In particular, because the robots are coordinating, but are not collaborating, no target robot is responsible for the success

of its followers. If a follower robot falls behind, or is stopped in place, the target robot does not stop. However, when using the teamwork architecture, as long as the follower robot is able to communicate, it will automatically let the others know if it fails, and as a result they will stop and wait for it. Potentially, specific behaviors could be run to handle these special cases.

8.3 Using a Teamwork Architecture Cuts Development Efforts

The overall size of the binary code just under 1MB. There are approximately $11,000$ lines of code in the project, excluding the teamwork architecture code to which we have no access. Given the source code, we can estimate using standard software engineering models the effort involved in producing it, and contrast it with the actual development time to validate the model. In addition, we can also estimate the size (in lines of code) of modules that we would have needed to develop, had we not had made use of the architecture. By contrasting these two totals, we can estimate the savings in development efforts due to the use of the teamwork architecture.

We begin by using standard software engineering tools to investigate existing source code. We use the Wheeler's SLOCCount software [67] to count lines of code, and generate estimates for *effort*, *schedule*, and *expected number of programmers*. The software makes these estimates using the basic CoCoMo model [7], well known in the software engineering community.

When we use SLOCCount on the source code tree, it predicts a 31.90 person-month effort (2.66 person-years), which is actually fairly accurate. The code was developed over two years. The team working on the code included:

- Two programmers working at 60% for a year. One continued to the second year of work; the other only worked for three months in the second year.
- Two MSc students who contributed work a few months out of the year (approximately 2 months working on the devices)

Module	LOC With CogniTAO	LOC Without CogniTAO (est.)
Devices and Robot Interfaces	5681	5681
BDI Architecture	0	6714
Behaviors	2958	3268
Conditions	1063	1063
Variables	927	1007
Protocols	330	647
Total	**10959**	**18380**

- Two PhD students who contributed work. One managed the project and worked also on key portions of the code (about 3 months, distributed over the two years of work). The other worked for no more than a month of the project (worked on getting the outdoor vision system to work).

We thus feel comfortable using the predictions of the SLOCCount system in estimating effort.

There are number of essential components in the system (Table 8.3, leftmost column): *Devices and Robot Interfaces* refer to code written to tie the control code, through the player/stage API and the shrimps robot API, to the actual hardware (including motors, sensors, color segmentation, the main() function and various utility code, etc.). This code would have had to be written, regardless of what type of architecture is used (and even if no specific architecture is used). The next two modules are specific to behavior-based or BDI architectures (even if not supporting teamwork). *Behaviors* (first data row) correspond to nodes in the behavior graph described in Figure 7.4. *Conditions* refer to the preconditions and terminations conditions (e.g., a condition that checks whether a robot has not seen its target in the last 30 frames). The *Variables* modules mixes both teamwork and individual code. It contains code to initialize and update variable values in memory. *Protocols* refer to the synchronization and allocation procedures, typically utilizing communications, to coordinate the robots. The protocols utilize existing mechanisms in the architecture (e.g., distributed shared memory, message passing) and need only be specialized to the task.

There are two data columns in Table 8.3. The first, marked *LOC With CogniTAO*, lists the lines of code in each of the main software components. The second,

marked *LOC Without CogniTAO (est.)*, lists the *estimated* number of lines of code in each of these components, had we not used CogniTAO. By contrasting these two, we can learn about the relative contribution that the use of CogniTAO brings to the system. To estimate the number of code lines without CogniTAO, we followed the following procedure (utilized in [64] for similar purposes). While the estimated numbers should not be considered accurate, they do reflect qualitatively the effort spent, and can be used to draw useful lessons.

First, we needed to estimate the number of lines of code for a BDI or behavior-based architecture, that does not have built-in teamwork. We used the SLOCCount software on the UMPRS system [50], a BDI system developed at the University of Michigan and available in open-source form. UMPRS was chosen because it is considered mature, and is a prototypical BDI system. UMPRS consists of about 6.7 thousands lines of code. This is a very conservative estimate for the number of lines of code that would be needed to build a BDI system for the use of the formation maintenance application, as there are certain features that we utilize in the system (such as maintenance behaviors [47]) which are not present in UMPRS.

Next, we turned to estimating the work involved in modifying the individual BDI code so that it supported the necessary teamwork logic. Suppose we started with an individual BDI system that had all the necessary individual controller pieces (i.e., each of the robot could execute its role in any of the different types of formation-maintenance schemes). What effort would be involved in making sure the teamwork logic was in place? This would include code to synchronize shared variables (e.g., for conditions), make joint allocation decisions via protocols, send messages back and forth, synchronize the beginning and ending of behavior execution, etc.

We use the behavior graph described in Figure 7.4 as the basis for the estimate. In different nodes in the behavior graph as it exists, the programmer relied on the teamwork architecture to carry out specific teamwork-related services, such as synchronizing the selection of new behaviors, making allocation decisions, synchronizing the termination of executing behaviors, sharing the values of variables, etc. Indeed, most of this is completely invisible to the programmer. For example, during the move in formation, in case of a failure of one of the robots, the teamwork architecture automatically stop the behavior and inform on the failure to all

other robots. Although the leader is allocated to a different sub-team, it is being informed on the failure and stops its execution.

We estimated the number of code lines it would have taken to duplicate this capability *in each instance* by 80 source code lines, based on example code we wrote for sending and acknowledging a message (both server and client side, socket exceptions, etc.). We include in this code necessary for transmitting variables, which includes also serialization of the variables internal structures, etc. An additional piece of needed code would have handled the sharing of variables across threads, and at least basic thread synchronization and management. We conservatively estimate this at 135 source lines, using a prepared code which runs two parallel threads. This number does not include the memory management and the mutexes needed for synchronization. The shared memory management was estimated as 40 lines of code using the same method.

We now use these estimates to get a sense of the number of lines of code it would have taken to replicate the system without the use of CogniTAO. We conservatively assume that the programmer would have added the communications code once for each protocol (each is a separate set of files), and once for all serialized variables. A less conservative estimate assumes the addition of those lines of code at each instance of the protocol and for each variable. This estimate gives less credit to the programmers (and/or the software engineering process) in terms of the expected reuse, and this in fact was the estimate used by others [64]. Nevertheless, we use the more conservative estimate.

Contrasting the two columns reveals interesting insights. First, a very sizeable portion of the source code is devoted to interfacing with the robot hardware, sensors, and actuators. In the version using CogniTAO, this accounts for about 50% of the code, and another 50% (approximate 5000 lines of code) are used to build the actual controllers. In the non-CogniTAO version, about 13,000 lines of code are used to build the application.

Ignoring the interfaces component in both cases, we see that the CogniTAO version uses only about 42% of the source code lines of the non-TAO version, i.e., it offers almost 60% savings in development efforts. About half of these are in the individual BDI architecture, and the rest in the teamwork services.

8.4 What Does Not Work (at least not easily)?

Due to the communication protocol of the teamwork architecture needed for the team synchronization, in case of communication suffers, the use of the architecture is going to go bad. This is in some sense limiting for formations, because one can certainly build formations that don't use communications, but we cannot do it with a teamwork architecture.

Without the communication, formation would not be as robust as with. There would be no ability to inform the other team members on case of failure, or to increase the robustness with the communication controller which improve the tracking and handle vision failure and robustness to obstacles.

Moreover, the switch controller would not be able to chose a new SBC graph, because it would not have the ability to build a new full graph for the Dijkstra algorithm or insure that the graph is connected.

Chapter 9

Conclusions

In Part I of this thesis, we developed techniques that allow robot selection that maintains bounds and guarantees as to travel times, even under uncertainty. We showed that even under the term of static environment, it takes the robots varying amount of time getting to a target location. Due to this variance, choosing a robot to perform a task cannot be done based on greedy selection (shortest path).

Thus, we introduced a decision making technique, inspired by economic decision theory, to distinguished between different policies based on risk. The experiments in simulated and physical robots demonstrated that different robots were chosen according to the different policies because of time travel variance: And indeed sometimes the closest robot is not the one to be selected, given the decision-making policy. Furthermore, we have shown that under some conditions, choosing the robot according to the selection policies will not always give a reasonable selection in practice. We defined the social regret function *SoR* which measure the cost of choosing specific robot over all other robot, and allow us to evaluate the gain from switching the chosen robot to a robot that will preform better. In our future work we plan to expand this techniques for allocating teams of N robots to K tasks.

While examining the experiments' data, we found that the data distributions had a good fit to the family of General Extreme Value distributions, and specifically to the General Log-Logistic distribution. We plan to explore this fit and learn to predict the parameters of the distributions for future real world paths and

obstacles.

In the second part of the thesis, we showed the use and contribution of a teamwork architecture. First, we described the principles of behavior-based teamwork software and the use of collaborative algorithms, which allows to share information with other robots. Then, we presented an implementation of a robust formation control using integration of different controllers over the teamwork architecture described.

We showed that the use of this architecture increases the reuse of code and gives the developer the ability to work only on the formation algorithms and reduce the need to handle the team communication and synchronization.

But while saving programming efforts, the teamwork architecture rely on a perfect communication. Any limitation of the communication will reduce the robustness of the formation and will limit the use of some controllers.

Bibliography

[1] R. Alur, A. Das, J. Esposito, R. Fierro, G. Grudic, Y. Hur, R. V. Kumar, I. Lee, J. Ostrowski, G. J. Pappas, B. Southall, J. Spletzer, , and C. J. Taylor. A framework and architecture for multirobot coordination. *International Journal of Robotics Research*, 21(10–11):977–995, 2002.

[2] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[3] T. Balch and M. Hybinette. Social potentials for scalable multirobot formations. In *Proceedings of IEEE International Conference on robotics and automation (ICRA-00)*, 2000.

[4] D. E. Bell. Regret in decision making under uncertainty. *Operations Research*, 30(5):961–981, 1982.

[5] C. Bererton, G. Gordon, S. Thrun, and P. Khosla. Auction mechanism design for multi-robot. In *In Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS 03*. MIT Press, 2003.

[6] J. Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE Journal of Robotics and Automation*, 4(4):443–450, 1988.

[7] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981.

[8] S. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *ICRA-99*, volume 2, pages 1234–1239, 1999.

[9] S. Carpin and L. Parker. Cooperative leader following in a distributed multi-robot system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[10] A. Chaudhry. Path generation using matrix representations of previous robot state data. In *2006 45th IEEE Conference on Decision and Control*, pages 6790–6795, 2006.

[11] L. CogniTeam. CogniTAO (Think As One). Think As One.

[12] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35, 1991.

[13] J. P. Desai. A graph theoretic approach for modeling mobile robot team formations. *Journal of Robotic Systems*, 19(11):511–525, 2002.

[14] J. P. Desai, J. P. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.

[15] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[16] M. B. Dias. *TraderBots: A New Paradigm for Robust and Efficient Multi-robot Coordination in Dynamic Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2004.

[17] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

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

[19] M. B. Dias and A. T. Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

[20] M. B. Dias, M. B. Zinck, R. M. Zlot, and A. Stentz. Robust multirobot coordination in dynamic environments. In *ICRA-04*, volume 4, pages 3435–3442, 2004.

[21] M. B. Dias, R. M. Zlot, M. B. Zinck, J. P. Gonzalez, and A. Stentz. A versatile implementation of the traderbots approach for multirobot coordination. In *IAS-8*, 2004.

[22] T. D.Vu, J. Go, G. A. Kaminka, M. M. Veloso, and B. Browning. MONAD: A flexible architecture for multi-agent control. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, 2003.

[23] Y. Elmaliach and G. A. Kaminka. Robust multi-robot formations under human supervision and control. *Journal of Physical Agents*, 2(1):31–52, 2008.

[24] A. Farinelli, L. Iocchi, D. Nardi, and V. A. Ziparo. Assignment of dynamically perceived tasks by token passing in multi-robot systems. *Proceedings of the IEEE*, 2006. Special issue on Multi-Robot Systems.

[25] R. Fierro, A. K. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-01)*, 2001.

[26] D. Foster and R. Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1–2):7–35, 1999.

[27] J. Fredslund and M. J. Mataric. A general algorithm for robot formations using local sensing and minimal communications. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[28] B. P. Gerkey and M. M. J. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[29] B. P. Gerkey and M. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[30] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multi-robot coordination. *TROA*, 2001. Special Issue on Multi-robot Systems.

[31] B. P. Gerkey and M. J. Mataric. A market-based formulation of sensoe-actuator network coordination. In *AAAI Spring Symposium on Intelligent Embedded and Distributed Systems*, pages 21–26, 2002.

[32] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. T. Stentz. A distributed layered architecture for mobile robot coordination: Application to space exploration. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

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

[34] G. Grisettiyz, C. Stachniss, and W. Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *ICRA-05*, pages 2432–2437, 2005.

[35] B. J. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[36] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 417–445. MIT Press, Cambridge, MA, 1990.

[37] K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 120–127. AAAI Press, 1998.

[38] K. Heero, J. Willemson, A. Aabloo, and M. Kruusmaa. Robots find a better way: A learning method for mobile robot navigation in partially unknown environments, 2004.

[39] G. Inalhan, F. Busse, and J. How. Precise formation flying control of multiple spacecraft using carrier-phase differential GPS. In *Proceedings of AAS/AIAA Space Flight Mechanics*, 2000.

[40] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[41] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coupled tasks. In *ICRA-06*, 2006.

[42] D. Jung and A. Zelinsky. An architecture for distributed cooperative planning in a behaviour-based multi-robot system. *Robotics and Autonomous Systems (RA&S)*, 26:149–174, 1999.

[43] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *ICRA-05*, pages 1170–1177, 2005.

[44] G. A. Kaminka and I. Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005.

[45] G. A. Kaminka and I. Frenkel. Integration of coordination mechanisms in the BITE multi-robot architecture. In *ICRA-07*, 2007.

[46] G. A. Kaminka, R. Schechter-Glick, and V. Sadov. Using sensor morphology for multi-robot formations. *IEEE Transactions on Robotics*, pages 271–282, 2008.

[47] G. A. Kaminka, A. Yakir, D. Erusalimchik, and N. Cohen-Nov. Towards collaborative task and team maintenance. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[48] S. Koenig, X. Zheng, C. Tovey, R. Borie, P. Kilby, V. Markakis, and P. Keskinocak. Agent coordination with regret clearing. In *AAAI-08*, pages 101–107, 2008.

[49] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1–2):79–97, 1997. Economic Principles of Multi-Agent Systems.

[50] J. Lee, M. J. Huber, P. G. Kenny, and E. H. Durfee. UM-PRS: An implementation of the procedural reasoning system for multirobot applications. In *Proceedings of the Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS-94)*, pages 842–849, 1994.

[51] M. Lemay, F. Michaud, D. Létourneau, and J.-M. Valin. Autonomous initialization of robot formations. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-04)*, 2004.

[52] G. Loomes and R. Sugden. Regret theory: An alternative theory of rational choice under uncertainty. *The Economic Journal*, 92(368):805–824, 1982.

[53] F. Michaud, D. Létourneau, M. Gilbert, and J.-M. Valin. Dynamic robot formations using directional visual perception. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[54] O. Michel. Webots^{TM}: Professional mobile robot simulation. *CoRR*, abs/cs/0412052, 2004.

[55] A. I. Mourikis and S. I. Roumeliotis. Optimal sensor scheduling for resource constrained localization of mobile robot formations. *IEEE Transactions on Robotics*, 22(5):917–931, October 2006.

[56] D. J. Naffin and G. S. Sukhatme. Negotiated formations. In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

[57] L. E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[58] P. Scerri, L. Johnson, D. Pynadath, P. Rosenbloom, M. Si, N. Schurr, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, 2003.

[59] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[60] W. Sheng, Q. Yang, J. Tan, and N. Xi. Distributed multi-robot coordination in area exploration. *RAS*, 54(12):945–955, 2006.

[61] R. Simmons, D. Apfelbaum, W. Burgard, M. Fox, D. an Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *AAAI-00*, 2000.

[62] B. Sofman, E. Lin, J. A. Bagnell, J. Cole, N. Vandapel, and A. Stentz. Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, 23(11-12):1059–1075, 2006.

[63] A. W. Stroupe, M. C. Martin, and T. R. Balch. Distributed sensor fusion for object position estimate by multi-robot systems. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-01)*, pages 1092–1098. IEEE Press, 2001.

[64] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[65] M. Tambe, D. V. Pynadath, N. Chauvat, A. Das, and G. A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-00)*, pages 301–308, Boston, MA, 2000.

[66] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, et al. Probabilistic algorithms and the interactive museum tour-guide robot Minerva. *The International Journal of Robotics Research*, 19(11):972—999, 2000.

[67] D. A. Wheeler. Sloccount.

[68] R. M. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *ICRA-02*, volume 3, pages 3016–3023, 2002.