Summer 8-12-2011

Coordination of Multirobot Teams and Groups in Constrained Environments: Models, Abstractions, and Control Policies

Nora Ayanian
University of Pennsylvania, nayanian@seas.upenn.edu

Coordination of Multirobot Teams and Groups in Constrained Environments: Models, Abstractions, and Control Policies

Abstract
Robots can augment and even replace humans in dangerous environments, such as search and rescue and reconnaissance missions, yet robots used in these situations are largely tele-operated. In most cases, the robots' performance depends on the operator's ability to control and coordinate the robots, resulting in increased response time and poor situational awareness, and hindering multirobot cooperation.

Many factors impede extended autonomy in these situations, including the unique nature of individual tasks, the number of robots needed, the complexity of coordinating heterogeneous robot teams, and the need to operate safely. These factors can be partly addressed by having many inexpensive robots and by control policies that provide guarantees on convergence and safety.

In this thesis, we address the problem of synthesizing control policies for navigating teams of robots in constrained environments while providing guarantees on convergence and safety. The approach is as follows. We first model the configuration space of the group (a space in which the robots cannot violate the constraints) as a set of polytopes. For a group with a common goal configuration, we reduce complexity by constructing a configuration space for an abstracted group state. We then construct a discrete representation of the configuration space, on which we search for a path to the goal. Based on this path, we synthesize feedback controllers, decentralized affine controllers for kinematic systems and nonlinear feedback controllers for dynamical systems, on the polytopes, sequentially composing controllers to drive the system to the goal. We demonstrate the use of this method in urban environments and on groups of dynamical systems such as quadrotors.

We reduce the complexity of multirobot coordination by using an informed graph search to simultaneously build the configuration space and find a path in its discrete representation to the goal. Furthermore, by using an abstraction on groups of robots we dissociate complexity from the number of robots in the group. Although the controllers are designed for navigation in known environments, they are indeed more versatile, as we demonstrate in a concluding simulation of six robots in a partially unknown environment with evolving communication links, object manipulation, and stigmergic interactions.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Mechanical Engineering & Applied Mechanics

First Advisor
Vijay Kumar

Second Advisor
Daniel E. Koditschek

Keywords
Robotics, Multirobot Control, Feedback Control, Multirobot Systems, Motion Planning, Dynamical Systems

Subject Categories
Controls and Control Theory | Dynamics and Dynamical Systems | Electrical and Computer Engineering | Mechanical Engineering | Robotics

COORDINATION OF MULTIROBOT TEAMS AND GROUPS IN CONSTRAINED ENVIRONMENTS: MODELS, ABSTRACTIONS, AND CONTROL POLICIES

Nora Ayanian

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2011

Supervisor of Dissertation

Vijay Kumar
Professor of Mechanical Engineering and Applied Mechanics

Co-Supervisor

Daniel E. Koditschek
Professor of Electrical and Systems Engineering

Graduate Group Chairperson

Jennifer R. Lukes
Associate Professor of Mechanical Engineering and Applied Mechanics

Dissertation Committee

George J. Pappas, Professor of Electrical and Systems Engineering
Steven M. LaValle, Professor of Computer Science, University of Illinois

COORDINATION OF MULTIROBOT TEAMS AND GROUPS
IN CONSTRAINED ENVIRONMENTS:
MODELS, ABSTRACTIONS, AND CONTROL POLICIES

To my parents.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Vijay Kumar, who has provided me with guidance on my course through graduate studies. You have made this a most rewarding experience, and I am grateful to you for being my mentor, advocate, and friend. I will not forget the encouragement and support through these years, which have not only been my most difficult academically, but also my most difficult personally. I am also grateful to my coadvisor, Professor Dan Koditschek, whose questions have challenged me, ideas have inspired me, and advice has buoyed me. I am indebted to you both for your immeasurable contributions to my success.

I would also like to express gratitude to the remainder of my dissertation committee, Professors George Pappas and Steven LaValle, for their comments and insights.

I am also extremely grateful to the National Science Foundation, for supporting this work with a Graduate Research Fellowship.

Special thanks to Vinutha Kallem for numerous discussions, impromptu lessons, putting up with my Mac-illiteracy, and my addiction to afternoon coffee. Chapters 6 and 10 would not have been possible without you. I'd also like to thank Hadas Kress-Gazit, with whom I had the pleasure of collaborating on automatic synthesis of controllers from natural language specifications, the topic of Chapter 9.

Thanks to Nathan Michael for helping me set up experiments, and to Daniel Mellinger for sharing his quadrotor model to use in simulations.

I would not have enjoyed this experience without the GRASPees whom I had the pleasure of befriending throughout my time at GRASP, especially Ani Hsieh, Dave Cappelleri, Quentin Lindsey, Ben Cohen, Victor Preciado, Jerome Le Ny, Jim Keller, Mike Shomin, Adam Halasz, and Paul White. Thanks also to my friends in the MEAM department who understood when I left the party early: Jack Franklin, Jason Thompson, Joe Grogan, and Rick Springman.

I'd also like to thank the MEAM faculty who have treated me with respect and entrusted me with responsibilities beyond those expected of a student, specifically Professors Mark Yim and John Bassani. Special thanks to Professor Harry Kwatny at Drexel University for having great faith in me, and helping me find the right direction instead of pointing me to it; I would not have considered Penn if it were not for you. Thanks also to the staff of MEAM, ESE, and GRASP: Sue Waddington Pilder for her amazing support, Maryeileen Banford Griffith for answering so many questions, Brianna Banford and Delores Magobet for performing magic with Vijay's and Dan's calendars, Olivia Brubaker for her consistent willingness to help, and Charity Payne for always coming through in my most (and least) dire times.

To my family, especially my parents, my brother, and my sister: I am extremely fortunate to have a family that is so supportive and has let me forge my own path through life. I am also thankful to my grandparents, living and deceased, who have always been so proud of even my most minor accomplishments. I still remember practicing arithmetic with my grandfather after school, and I am truly lucky to have had so much time with all of my grandparents, and to still have time with one. My family has made me who I am and this accomplishment belongs to them as much as it does to me.

Finally, my deepest thanks to Ed, who has always believed in me and encouraged me to follow my heart.

ABSTRACT

COORDINATION OF MULTIROBOT TEAMS AND GROUPS
IN CONSTRAINED ENVIRONMENTS:
MODELS, ABSTRACTIONS, AND CONTROL POLICIES

Nora Ayanian

Vijay Kumar and Daniel Koditschek

Robots can augment and even replace humans in dangerous environments, such as search and rescue and reconnaissance missions, yet robots used in these situations are largely tele-operated. In most cases, the robots' performance depends on the operator's ability to control and coordinate the robots, resulting in increased response time and poor situational awareness, and hindering multirobot cooperation.

Many factors impede extended autonomy in these situations, including the unique nature of individual tasks, the number of robots needed, the complexity of coordinating heterogeneous robot teams, and the need to operate safely. These factors can be partly addressed by having many inexpensive robots and by control policies that provide guarantees on convergence and safety.

In this thesis, we address the problem of synthesizing control policies for navigating teams of robots in constrained environments while providing guarantees on convergence and safety. The approach is as follows. We first model the configuration space of the group (a space in which the robots cannot violate the constraints) as a set of polytopes. For a group with a common goal configuration, we reduce complexity by constructing a configuration space for an abstracted group state. We then construct a discrete representation of the configuration space, on which we search for a path to the goal. Based on this path, we synthesize feedback controllers, decentralized affine controllers for kinematic systems and nonlinear feedback controllers for

dynamical systems, on the polytopes, sequentially composing controllers to drive the system to the goal. We demonstrate the use of this method in urban environments and on groups of dynamical systems such as quadrotors.

We reduce the complexity of multirobot coordination by using an informed graph search to simultaneously build the configuration space and find a path in its discrete representation to the goal. Furthermore, by using an abstraction on groups of robots we dissociate complexity from the number of robots in the group. Although the controllers are designed for navigation in known environments, they are indeed more versatile, as we demonstrate in a concluding simulation of six robots in a partially unknown environment with evolving communication links, object manipulation, and stigmergic interactions.

Contents

List of Tables

List of Figures

Part I

Goals and Background

Chapter 1

Introduction

Every day, soldiers and emergency responders put their life at risk for the greater good. In such applications as reconnaissance missions, mine detection, and disaster rescue, groups of robots can augment and even replace humans in order to spare injury to those that protect us. For example, a group of unmanned aerial vehicles (UAVs) can be deployed to survey the aftermath of a natural disaster, determining a safe path for ground vehicles' approach. A team of robots can be released into buildings which have been a target of a bioterrorist attack, searching for survivors. Teams of robots are currently being deployed to detect and disable mines all over the world [110, 135].

During such missions, it may be necessary to maintain communication within the team of robots, or to maintain communication with a base of operations. Other constraints may also be necessary, such as keeping specific robots together or prohibiting certain robots from entering specific regions in the environment.

The benefits of extended autonomy in these situations are numerous. Robots are capable of executing many tasks much more accurately than humans. They can reduce the number of emergency responders which must be sent into dangerous

environments, and in some situations entirely eliminate the need. Robots can also be extremely efficient; they can quickly calculate optimal deployments, minimize risk, and complete a task in less time than humans. Although all of these capabilities have not yet been achieved in all situations, computers are continually advancing, and it is only time until robots surpass humans in many tasks.

There are, however, several factors other than computing power standing in the way of extended autonomy in these situations. First, every situation is *unique*. For each different application and workspace, from office buildings to warehouses to city blocks, we must be able to automatically generate a control policy. Another factor is *scale*: how many robots are needed to efficiently complete the task? This leads to the *expense* of purchasing multiple robots, and converting existing infrastructure for compatibility. As we increase the number and type of robots the *complexity* also increases, and the capabilities of each robot must be considered. In real-world situations, the robots will have to deal with *dynamic conditions* such as weather, unforeseen obstacles, moving targets, or a period of communication loss, and they must be empowered with making the right decision, or know when to defer to human intervention. The robots must be able to guarantee *safety*, whether it is not causing harm in reconnaissance missions, bringing victims to safety in search and rescue, or maintaining their own safety. Finally, *time* may be a critical factor in all of these situations. While there are numerous things robots can do faster than humans, they must somehow be programmed to do them, and waiting for an expert in these circumstances might mean the difference between saving or losing hundreds of lives.

At least in part, these factors can be dealt with in two ways: increasing the number of robots, and making controller synthesis automatic. By having many cheap, less capable robots instead of fewer expensive, more capable robots, one can address both scale and expense. There are many benefits intrinsic to multiple robots

working cooperatively, including increased efficiency because of parallelism within the system. Complexity and situational uniqueness can also be addressed in part by having many cheap robots with different capabilities. Heterogeneous teams of robots can accomplish tasks that individual robots cannot perform, such as aerial and ground approach, sensing and disabling land mines, transporting heavy objects, etc. Having control policies which deal with these factors by construction, as well as handling dynamic conditions when necessary, further addresses complexity and situational uniqueness. Safety can be addressed by using control policies which are provably correct, guarantee convergence to the goal, prevent collisions, and preserve communication when critical. Finally, we can minimize the time it takes to deploy the robots by automatically synthesizing controllers which satisfy all of these criteria.

1.1 Problem statement

We focus in this thesis on *automatic* synthesis of feedback policies for heterogeneous groups and teams of robots. We address the problem of coordinating a team of robots to accomplish a large task which can be divided into smaller parallel subtasks. For example, sweeping through a building can be divided into subtasks such as sweeping multiple floors simultaneously, or reconnaissance missions into surveying separate city blocks concurrently. We adopt definitions of groups and teams of robots from Anderson and Franks [1], where they are used to describe animal societies. We call a collection of robots that closely coordinate with each other a *group* of robots: a group works on an individual subtask. A collection of groups is called a *team*: a team coordinates to ensure that the large task is accomplished. The team of robots is deployed with some initial *organization*, describing which robot is in which group.

The first part of the thesis is concerned with the foundations necessary to coordinate multiple robots while providing guarantees on convergence, maintaining communication, and preventing collision. Specifically, we model *configuration spaces for multirobot navigation problems* where each robot is assigned a specific, distinct goal in the workspace, as well as *configuration spaces for group navigation problems* where a single goal is assigned for the entire group. We develop *abstractions on groups of robots*, making the complexity of group navigation independent of the number of robots in the group. In these spaces, we develop two feedback policies and demonstrate them in simulation. The first is *decentralized affine feedback control for navigation of kinematic systems*, which is decentralized in the sense that the feedback of one robot is not dependent on the state of another if they are not in direct communication. The second is *smooth feedback control for navigation of second order systems*, which we couple with a graph embedding to apply to complex dynamical systems such as quadrotors.

The problem of accomplishing a large task involves a number of challenges beyond modeling and robot navigation, including task assignment, rendezvous, object manipulation, and intergroup coordination. Task assignment involves determining the order in which subtasks should be completed, the groups that should complete each subtask, and which robots should belong to which groups. Once the task assignment is complete, robots must rendezvous with their new groups and navigate to their task location, or vice versa. There they must create the structure required for the task, i.e. some arrangement of the group, such as a formation or encircling of a target. The task itself may require interaction with the environment, such as manipulating or transporting an object. Additionally, groups must be able to communicate with other groups to ensure that the overall task is completed. While in-depth exploration of task assignment, object manipulation, and rendezvous are outside the

scope of this thesis, we do address them, both in discussion and in simulation, as any true task will require all of these.

The second part of this thesis presents applications of the foundations we develop in the first part. First, we present a method by which groups of robots can create and maintain formations, as well as merge and split. Second, we demonstrate the capabilities of such controllers to complete dynamic tasks. Finally, we combine configuration space modeling, abstracting groups of robots, and synthesizing feedback policies in a simulation which involves dynamic conditions, task assignment, object manipulation, and multirobot coordination by stigmergic interaction. *Stigmergy* is a mechanism by which agents coordinate by modifying their local environment, stimulating action in other agents. For example, ants lay pheromone trails to direct other ants in their colony to the nest.

In summary, the contributions of this thesis are threefold: 1) modeling configuration spaces for multirobot and multigroup coordination, such that desired constraints will always be maintained; 2) abstractions on groups of robots, which reduce the complexity of multirobot navigation, making it independent of the number of robots; and 3) automatic synthesis of feedback policies which, by construction, maintain the desired constraints. All of these are done *automatically*: there is no hand tuning and no expert intervention necessary.

1.2 The case for centralization

The work presented in this thesis requires centralized computations. The argument can be made that a decentralized approach would reduce the computational power necessary to solve these problems, as well as enabling robots to truly self-organize. However, without some centralization we cannot guarantee completeness for the

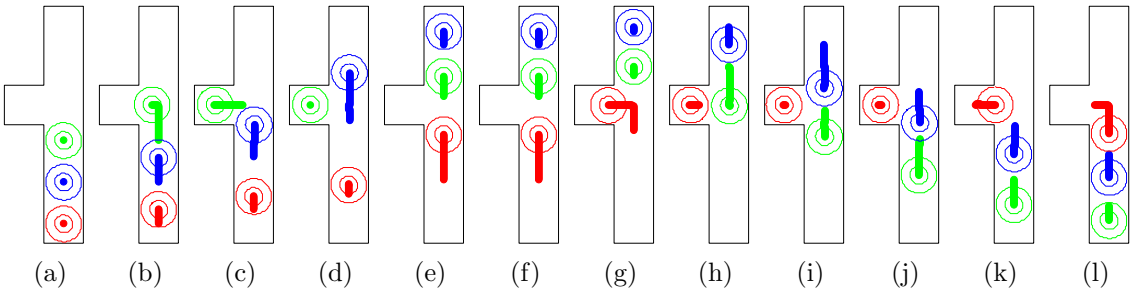


Figure 1.1: A challenging problem easily solved with our centralized method. The simulation starts in (a) with the robots stacked green-blue-red. (b)-(l) show sequential frames of the simulation, with the robot trajectory since the last frame. In (l), the robots reach the goal position, red-blue-green.

types of problems we solve in this thesis. Completeness means that a solution will be found by an algorithm if one exists, and if not, the algorithm will return failure. An example of a problem which would be impossible to solve with typical decentralized algorithms is shown in Fig. 1.1. Here, the robots are stacked green-blue-red in a narrow corridor, and must switch to red-blue-green. Using a decentralized approach, each robot would attempt minimize (or maximize) some utility function by moving towards its goal position. However, the red and green robots are blocked from moving any closer to their goal position by the blue robot, which happens to be in its own goal position. (Even with two robots, this problem would be extremely difficult to solve with a decentralized approach.) By using a centralized approach, however, we can search the state space for a solution to the problem in a fraction of a second.

While it is true that some decentralized methods, such as stochastic approaches, can solve this problem, it would take much more actual run time on the robots, which is more costly than computation time on a computer. With the centralized approach, we increase precomputation time, but can optimize the run time, which can translate into less expense. However, as the number of robots grows, the benefit of a centralized approach decreases. To solve this type of problem for 100 robots

with a centralized algorithm would be impossible with today's computational power. With 100 robots, though, it's probably be much less important what order they are in; hence, with large group sizes, decentralized approaches are both necessary and sensible. However, when specific labeled formations or configurations are necessary, centralized approaches are superior.

1.3 Organization of this work

The thesis is organized as follows. In the following chapter, we discuss multirobot control as it has appeared in the literature, beginning with foundational work in single robot control. We also characterize multirobot controllers and discuss applications and important considerations such as task assignment.

In Part II, Chapters 3 and 4 present the first two major contributions of this thesis, modeling configuration spaces for groups of robots and abstractions on groups of robots. Specifically, Chapter 3 introduces robot configuration space modeling with constraints such as communication maintenance and collision avoidance, building a discrete representation of the configuration space, and efficient construction of the configuration space via heuristic-based graph search on its discrete representation. Chapter 4 begins with abstractions on groups of robots, then presents modeling the abstract configuration space which are used for group navigation. The third major contribution of the thesis, automatic feedback controller synthesis is presented in Chapters 5 and 6. Chapter 5 formulates decentralized affine feedback controllers for kinematic systems while Chapter 6 formulates nonlinear feedback controllers for kinematic and dynamic systems. Chapter 7 introduces alternate controllers developed by others which we have made use of in this work, and demonstrates their use on group navigation.

In Part III, Chapter 8 introduces formations on groups of robots, and a method for joining and splitting groups. Chapter 9 applies configuration space modeling to the dynamic task of sorting recyclables. Finally, Chapter 10 incorporates modeling, abstractions, and controller synthesis in a comprehensive simulation with six robots. While the methods presented in Chapters 3 through 6 are computationally intensive when used exactly as presented (a complexity analysis is included within these chapters), they are indeed extremely versatile and useful for real-time planning as we demonstrate in this chapter. In the simulation, computation is done online and not in advance. Communication graphs are dynamic, tasks assignment is online and unknown a priori, and the robots interact with the environment, yet we still provide the same guarantees of convergence and safety.

The work in Chapters 3 and 5 was previously published in part in [2, 5, 6]. The abstract configuration space modeling in Chapter 4 was previously published in [4]. The nonlinear feedback control for second order systems presented in Chapter 6 was done in collaboration with Dr. Vinutha Kallem, and has been submitted for publication [3]. Chapter 8 was previously published in [6]. Finally, Chapter 9 was previously published in [57], and was done in collaboration with Dr. Hadas Kress-Gazit, who is responsible for the natural language portion of the work.

We start with Chapter 2, which reviews the robot navigation literature, including single and multiple robot navigation for small to large groups of robots, formation control, and cooperative transport, as well as other topics.

Chapter 2

Literature Review

Multi-robot problems have been discussed in the literature in many forms, including creating and maintaining formations, using abstractions to lower dimensionality, and using gradient descent for navigation. Each of these methods is applied for groups or teams of varying size. Recall that a group is a number of robots working closely together to accomplish a single task, while a team is a number of groups working together to accomplish a task comprised of multiple subtasks.

2.1 Single robot controllers

Before discussing multirobot controllers, we will review single robot controllers. Single robot controllers have been developed for both holonomic and nonholonomic robots.

Gradient descent algorithms have been discussed by Rimon and Koditschek in synthesizing *navigation functions* [104], which are nonlinear feedback controllers that guarantee safety (obstacle avoidance) and global convergence. This method generates a single analytical feedback control law in star-shaped environments with obstacles

which can be applied to robots. When obstacles are present, hand tuning is necessary to prevent local minima.

2.1.1 Hybrid systems approaches

These gradient descent algorithms have inspired other hybrid systems approaches. A hybrid system is one which has both discrete and continuous aspects. Perhaps the most commonly used hybrid systems approach in robot control is to decompose the configuration space into obstacle-free cells and synthesize controllers in each cell to guarantee convergence from any starting configuration to the goal configuration. In other words, controllers are *sequentially composed* to drive the system to the goal configuration. Depending on the controller, the cells can be polygonal, analytical, rectangular, etc. This simplifies the construction of controllers by creating the gradient using obstacle-free cells, preventing local minima and thus removing the need for hand-tuning.

In Conner et al., a Laplace heat equation is solved on a 2-dimensional disk, which is then mapped to the convex polygons [19]. Composing these mappings results in a smooth controller for a single robot on the 2-dimensional workspace. This work is not applicable to multi-robot problems without the addition of constraints (such as prohibiting more than one robot from occupying a room simultaneously) and requires an analytical function to describe the cells. The function must be twice differentiable in order to maintain the qualities of the navigation function, therefore the cell is approximated at the corners with a polynomial. In contrast, Lindemann and LaValle directly construct a smooth vector field in the obstacle-free cells, avoiding the analytical construction of the navigation function [71, 72]. Thus, the controller can be used in large dimensional spaces since construction of a smooth

approximation of the cell boundary is not required.

While the above controllers were developed specifically for robot control, controllers not necessarily intended for robot control can also be applied to robot control problems. For piecewise affine systems, Habets and van Schuppen [43] and Roszak and Broucke [105] synthesize controllers which drive a system in a polytope to a specified exit facet. Habets and van Schuppen use linear programming to synthesize controllers inside polytopes and provide guarantees on existence of controllers for fully actuated systems [43]. Roszak and Broucke extend this work, solving a larger class of problems on simplices, but their solution requires nonlinear programming, making it more difficult to implement [105].

2.1.2 Approaches for nonholonomic robots

The previously mentioned controllers are generally derived for holonomic robots, then are applied to nonholonomic robots using feedback linearization. It is also possible to synthesize controllers for nonhonolomic robots, such as one with a unicycle model [18, 73, 74].

For nonholonomic systems such as cars, which are limited in their mobility since they can't move sideways, generating a controller for a holonomic robot then using feedback linearization to apply it to the robot can result in some instability. Trees of trajectories, however, are well suited for such robots, since they can generate paths that a specific robot can follow, for example, by taking into account a minimum turning radius. LaValle and Kuffner introduced Rapidly-exploring Random Trees (RRTs), which construct a tree that rapidly and uniformly explores the state space according to the constraints on the robot [64]. Much work has expanded on RRTs,

including Karaman and Frazzoli's RRT*, which has asymptotic optimality guarantees without a significant increase in complexity over RRTs [50]. Without including feedback, however, these methods cannot compensate for motor drift or inaccuracies in robot models. Tedrake et al. introduce LQR-trees, which combine local linear quadratic regulator (LQR) controllers into a feedback policy [125]. The LQR-trees probabilistically cover the reachable area with a region of stability, finding a locally optimal solution at every step; thus, the method does not necessarily produce the globally optimal solution. While these methods are well suited for nonholonomic constraints, trees of trajectories and LQR-trees can search forever without finding a path to the goal and without returning failure.

2.2 Multirobot controllers

Multirobot controllers can be classified using a number of criteria. It will be useful to define a taxonomy for multirobot controllers to use throughout the discussion to compare controllers. We will initially classify a controller based on its target number of robots. While it is difficult to quantify what we classify as small, medium, and large groups of robots, it will become clear in their description that controller capabilities enable this distinction. For example, guarantees become difficult to provide as group size changes from medium to large. Our taxonomy for classification of multirobot controllers is described in Table 2.1.

Table 2.1 is a starting point for distinguishing between types of multirobot controllers. There are many characteristics which can be classified in many different types. Furthermore, not all controllers will fall into a distinct category. For example, a controller may be automatically generated in some scenarios, but need hand

Attribute	Types	Characterization
Number of robots	(a) Small	Labeled robots, unique formations, in many cases guarantees of convergence
	(b) Medium	Use of virtual structures, some guarantees are possible
	(c) Large	Unlabeled robots, convergence to unique formations not achievable, swarms of robots, no guarantees in the presence of obstacles
Automatic Generation	(a) Automatic	No hand tuning necessary, controller is synthesized automatically based on environment parameters
	(b) Not Automatic	Controller requires hand tuning to prevent local minima, or requires input other than environmental parameters
Computations	(a) Centralized	All computations are centralized
	(b) Partially Decentralized	Some computations are decentralized, such as determining individual controllers, however, other parts are centralized
	(c) Decentralized	Computations are fully decentralized, each agent determines its own action
Environment	(a) Known	Workspace boundary and all obstacles fully known, including known moving obstacles
	(b) Partially known	Unknown moving or stationary obstacles
	(c) Unknown	Nothing about the environment is known
Convergence	(a) Guaranteed	Robots are guaranteed to converge to the goal location
	(b) Qualified	Robots are guaranteed to converge given some criteria (e.g. obstacles no larger than robots)
	(c) None	No convergence guarantees.
Obstacles	(a) None	Does not account for obstacles
	(b) Static	Accounts for static obstacles
	(c) Dynamic	Accounts for dynamic obstacles.
Collisions	(a) Prevented	Guarantees that robots will not collide
	(b) Allowed	Collisions between robots are allowed
Level	(a) High	Controls behavior, interactions, etc.
	(b) Low	Directly synthesizes control inputs
System Order	(a) 1st Order	Single integrator system
	(b) 2nd Order	Double integrator
Robot Model	(a) Holonomic	Robot can move in any direction
	(b) Nonholonomic	Robot motion directions limited

Table 2.1: Table of multirobot controller characteristics.

tuning in others. It is also difficult to categorize the environment in which the controllers can be used. A controller meant for use in a known environment might lend itself to finding an object whose location is unknown. This is different than a controller which is designed for use in an environment which may have the correct environment topology but with nonexact measurements. These are both partially known environments, but are entirely different. Therefore, while this table is provided as a taxonomy of multirobot controllers, it is still quite challenging to classify even existing controllers.

We will use some of the attributes in Table 2.1, such as number of robots, computations, and convergence, to differentiate controllers which specifically drive a group of agents from one location to another. More specifically, we will discuss controllers which do so while maintaining desired inter-robot constraints, such as the distance or relative position of robots from a real leader, virtual leader, or from each other.

2.2.1 Small groups of robots

In small sized groups we can provide guarantees and formal proofs that specific formations can be achieved and maintained, even in the presence of obstacles. Egerstedt and Hu synthesize controllers for robots to maintain a desired formation in the presence of obstacles, both for holonomic and nonholonomic systems [29]. Olfati-Saber and Murray use potential functions to create a unique desired formation of robots using weakly connected graphs [92]. Desai et al. synthesize feedback controllers to navigate a group of robots to a goal location, providing proofs of convergence; as the groups grow, however, formal proofs become prohibitively complex [26].

Navigation functions are developed for multi-robot problems in [28,75,76]. While

this approach has the advantage of resulting in controllers with almost global convergence and smooth feedback, it is tedious for complex spaces, involves nonlinear equations, and requires hand-tuning of parameters, making them impractical for a non-expert user.

2.2.2 Midsize groups of robots

With sizes between small and large groups, one can provide some guarantees while taking advantage of some reduction in complexity. In Ogren et al. provide proofs for maintaining desired formations using a *virtual leader* from which robots must maintain desired constraints [90]. Leonard and Fiorelli use virtual leaders to create unlabeled formations of groups of robots, but undesirable local minima can occur if sufficient virtual leaders are not added [67]. Smith et al. present a decentralized controller for maintaining formations [114, 115]. In these controllers, no guarantees are made in the presence of obstacles and, the authors do not provide a stable way to switch between formations.

2.2.3 Large groups of robots

In large groups, one cannot feasibly synthesize a specialized controller for each robot, thus achieving specific, labeled formations is not addressed. Some works use abstractions to control an entire group [11, 87, 130]; in this case, navigation and obstacle avoidance is at the abstraction level, decreasing computation significantly. However, we forfeit control over the network topology, which can change as the group moves. Belta and Kumar do not guarantee safety: robots can collide and escape from the abstraction [11]. Some limitations of [11] are addressed by Michael and Kumar, but this still does not enable us to specify formations in the sense of exact shape and

topology [87]. Yang et al. allow a particular formation to be specified, but the number of moments which must be supplied to specify a particular formation increases with the number of robots, and the method is not entirely automatic [130].

Flocking or schooling strategies are inspired by animal societies, where groups of animals coordinate with relatively little communication [23, 95, 101]. These observations can be used to control large groups of robots with relatively little computation [46, 48, 66, 79, 91, 99, 100, 122, 131]. These strategies stabilize the entire group's velocity to a single velocity. However, like the above large scale controllers, they lack the capability of specifying particular formations; the final shape of the formation depends on the initial conditions, and cannot be controlled directly. Results also exist using second order robot models [66, 99].

Also inspired by animal behaviors are controllers which use stigmergic interactions to influence agent behavior. Johansson and Saffiotti use writable RFID tags in the environment to guide robots to a given goal [47]. In task allocation, Meng and Gan use stigmergic interactions to evenly and efficiently distribute robots [81], while Borzello and Merkle use "pheromones" to mark tasks so agents are more likely to choose them [13]. Werfel et al. use stigmergic interactions to perform construction tasks with robots [127, 128]. Lan et al. take inspiration from stigmergy to generate clustering behaviors [62].

In shape generation, large groups of robots are stabilized to lines and circles [129], convex closed loops [94], and more general closed loops [45]. However, the presence of obstacles in the workspace could cause local minima or deadlock. Additionally, there is no ordering to the robots on the shape; this is a function of initial conditions. Along a similar vein, groups are stabilized to patterns in [119, 120].

2.3 Applications

Multirobot controllers are not only used for navigation. They are applicable to many types of problems, including formation control and cooperative transport, which have been extensively studied in the literature.

2.3.1 Formation control

Formation control is one of the most studied problems in multirobot systems. Behavior based formation control involves assigning specific behaviors to robots in the group [7, 8, 65]. Virtual structures approaches can involve creating virtual leaders for the group or treating the entire group as a single entity [10, 49, 67, 68, 90]. In the leader-follower approach, each robot is assigned a leader from which it must maintain certain constraints [26, 32, 82, 109, 124]

Other approaches also exist, such as using navigation functions [40,123], neighbor alignment approaches [36], among many other approaches [31, 79, 132].

2.4 Cooperative transport

Some tasks may require transporting objects in the workspace to a desired location. Transporting objects using robots has been discussed in the literature by pushing [78], towing [17], caging [33,34,96,118,126], and other means of object manipulation [37,88,89]. Some have addressed the aerial transport of objects using cables [35,83]. Others have taken inspiration from ant societies [12, 25, 60].

2.5 Other considerations

The synthesis of controllers to navigate robots to a specific configuration, requires knowledge of that configuration. The configuration can be assigned by an external algorithm or it can be assigned by the robots themselves in a distributed way. The robots can be assigned to different locations, or to rendezvous in one location. Other considerations, such as the type of environment (urban, warehouse, etc.), can be critical in choosing an appropriate controller.

2.5.1 Rendezvous

Rendezvous problems are relevant since we would like robots to join new groups upon task assignment. Rendezvous, in the sense that we will use it, involves a number of robots meeting at one point. This has been discussed by [21,39,69,70,134]. Kranakis et al survey mobile agent rendezvous [56].

2.5.2 Task assignment

In swarm assembly, stochastic procedures such as chemical reaction models are often used, as the complexity of assigning each robot to a task is prohibitive. However, in these cases, there are many iterations of the same task (such as building a two-part assembly), and the large number of task-trained robots randomly walking throughout the space will result in part assembly at a satisfactory rate. In contrast, when dealing with a mid-sized number of robots and different, non-repetitive tasks requiring different numbers of robots, task assignment is necessary [42,80,85,85,88,93,111,113,117,121,133].

2.5.3 Urban environments

In urban environments, we cannot assume that the robots have the state of the entire group, that the information they have is accurate, or that it can be reliably transmitted over a network. In [15, 16], a group of robots is deployed in an urban environment, with localization provided by a UAV. Batalin and Sukhatme present a method for coverage in an urban search and rescue context [9]. In urban environments with many unknown and unreliable communication, indirect communication between agents may be more efficient than direct communication. Steele and Thomas introduce directed stigmergy control, which unlike most works cited here, allows some operator control, but has the benefit of indirect communication between robots [116].

Part II

Foundations

Chapter 3

Robot Configuration Space Modeling with Constraints

In order to drive multiple robots to a desired location, we must model the configuration space of the group of robots. Since we would like to navigate groups of robots in cluttered environments, we choose to construct the group's configuration space by taking the Cartesian products of the individual robots' configuration spaces. By doing so, we can generate a single controller to drive the group to the goal configuration without planning a path and generating a controller for a single robot, then planning a path and generating a controller for another robot so that it avoids the original robot, etc, until a path and controller is generated for all robots. Using such an incremental approach could potentially prevent a solution from being found even if it is otherwise possible to navigate the group to the goal configuration. By using our approach to modeling the configuration space, we preserve all possible solutions in a polygonal environment. The primary disadvantage of our approach is that it is centralized, and since it is precomputed, the environment must be known and static. However, by planning in the Cartesian product of the individual configuration spaces,

we can enable automatic synthesis of controllers which guarantee convergence while avoiding static obstacles and inter-robot collisions.

This approach to modeling the configuration space allows us to control both high- and low-level behavior by setting constraints on groups of robots, e.g. maintain a minimum distance from each other in order to prevent collisions and a maximum distance to prevent loss of communication, or prevent two or more robots from entering a region in the space simultaneously, etc.

In this chapter, we compute this group configuration space, which we call the *task configuration space*. The task configuration space is a space composed of polytopes, in which the robots cannot violate the set constraints. To drive the system to the goal location, we use sequential composition: we find paths on the polytopes to the goal location and synthesize controllers in each polytope to drive the system to the subsequent polytope, until the goal polytope is reached. In the goal polytope, we synthesize a controller to drive the system to the goal configuration.

Consider a group $\mathcal{G}$ of n robotic agents $\mathcal{V}_a = \{a_i | i = 1, \ldots, n\}$. We define a group as a small number of robots which work closely together (adopted from [1]). Starting from some initial configuration, the agents must reach a goal configuration while maintaining constraints between agents and without colliding with each other or obstacles. The agent a_i has the configuration or state $\mathbf{x}_i = [x_i^1 \; x_i^2 \; \cdots \; x_i^{d_i}]^{\mathrm{T}} \in \mathbb{R}^{d_i}$ with the dynamics

$$\dot{\mathbf{x}}_i = \mathbf{u}_i, \; \mathbf{x}_i \in \mathbf{X}_i \subset \mathbb{R}^{d_i}, \; i = 1, \ldots, n, \tag{3.1}$$

if the agent is kinematic or

$$\ddot{\mathbf{x}}_i = \mathbf{u}_i, \; \mathbf{x}_i \in \mathbf{X}_i \subset \mathbb{R}^{d_i}, \; i = 1, \ldots, n, \tag{3.2}$$

if the agent is dynamic.

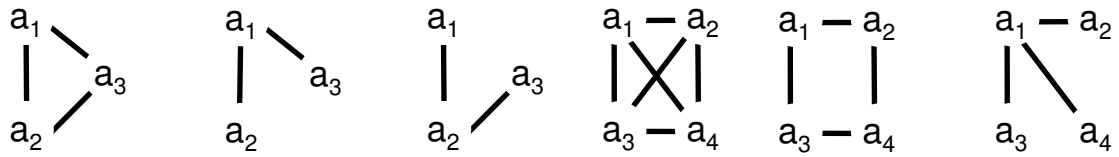
If each robot has a specific goal location, e.g. each robot is assigned a different corner of the space, we construct a *task configuration space* for the group of robots. A single point in the task configuration space specifies the position of all of the n robots. The following section describes the construction of the task configuration space. If a single goal location is specified for the entire group, or if the group is to transit together from one location to another, we create a virtual boundary for the group of robots and construct an *abstract configuration space* to navigate that boundary to the task location. A point in the abstract configuration space describes the position, orientation, and shape of the abstraction as it navigates the space. This is discussed in the following chapter.

3.1 Constraints on robots

In order to safely navigate a group of robots to their goal locations, we must consider constraints such as collision avoidance. In order to avoid collisions, robots must maintain a minimum distance from each other and from objects such as walls in the environment. Furthermore, we would like the robots to maintain a specific communication graph throughout.

We associate these predetermined proximity constraints for collision avoidance and communication with collision and connectivity graphs. We use halfspaces to define the constraints in order to generate a task configuration space composed of polytopes. Recall that a *D-polytope* can be defined as the convex hull of finitely many vertices, or as the bounded intersection of J halfspaces, $P = \{\mathbf{x} \mid H\mathbf{x} \leq K\}$, where $H \in \mathbb{R}^{J \times D}, K \in \mathbb{R}^{J \times 1}$, where J is finite.

Definition 3.1.1. *The* collision graph *on the set of agents is a static graph* $G_L =$



(a) (b) (c) (d) (e) (f)

Figure 3.1: Sample connectivity graphs.

$(\mathcal{V}_a, \mathcal{E}_L)$ *where* $\mathcal{E}_L$ *is the set of all pairs of agents which cannot occupy the same coordinates simultaneously. Pairs* $(a_i, a_j) \in \mathcal{E}_L$ *must maintain a nonzero minimum distance* $|\mathbf{x}_i - \mathbf{x}_j| \geq \boldsymbol{\delta}^{i,j}_{\min}$*, where* $\boldsymbol{\delta}^{i,j}_{\min} \in \mathbb{R}^{\max(d_i,d_j)}$*. This constraint can be written as*

$$\lambda(\mathbf{x}_i, \mathbf{x}_j) = |\mathbf{x}_i - \mathbf{x}_j| - \boldsymbol{\delta}^{i,j}_{min} \geq \mathbf{0} \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E}_L.$$

Definition 3.1.2. *The* connectivity graph *on the set of agents is the static graph* $G_N = (\mathcal{V}_a, \mathcal{E}_N)$ *where* $\mathcal{E}_N$ *is the set of edges describing agent pairs* $(a_i, a_j) \in \mathcal{E}_N$ *that must maintain a maximum distance* $|\mathbf{x}_i - \mathbf{x}_j| \leq \boldsymbol{\delta}^{i,j}_{\max}$*, where* $\boldsymbol{\delta}^{i,j}_{\max} \in \mathbb{R}^{\max(d_i,d_j)}$ *to communicate state information. We call pairs of agents which are adjacent on this graph* neighbors *or* neighboring agents*. The constraint can be written as*

$$\nu(\mathbf{x}_i, \mathbf{x}_j) = |\mathbf{x}_i - \mathbf{x}_j| - \boldsymbol{\delta}^{i,j}_{\max} \leq \mathbf{0} \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E}_N.$$

Examples of connectivity graphs for three or four agents are shown in Fig. 3.1.

The connectivity graph provides constraints on the physical proximity of agents, so that communication links between pairs of agents $(a_i, a_j) \in \mathcal{E}_N$ can be maintained. However, this does not necessarily mean that agents cannot pass other agents' state information through the graph. For example, in Fig. 3.1e, a_1 and a_4 are not connected by an edge, therefore they cannot communicate directly. However, by passing a_1's state information through a_2 or a_3 (and vice versa), a_1 and a_4 would be capable of

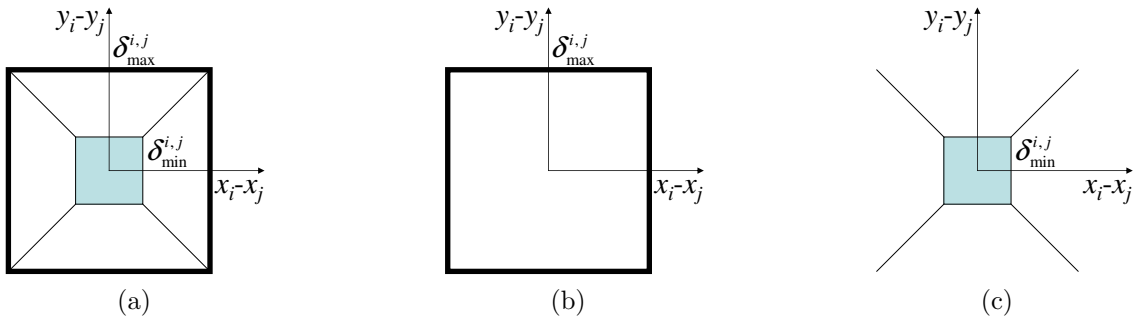


Figure 3.2: Symmetric proximity constraints in the relative space of two agents in 2D. The shaded region indicates configurations that are not allowed. (a) Neighbors with collision constraint. (b) Neighbors with no collision constraint. (c) Collision constraint on neighbors.

using each others' state information for feedback.

Definition 3.1.3. *The* information graph *on the set of agents is the static graph* $G_I = (\mathcal{V}_a, \mathcal{E}_I)$ *where* $\mathcal{E}_I$ *is the set of all pairs of agents which can access each others' state information. State information can be shared through direct communication, if the pair is connected on* G_N*, or indirect communication, by passing state information through other agents.*

Figure 3.2 illustrates symmetric collision and connectivity constraints on a pair of 2D agents. For pairs of agents $(a_i, a_j) \in \mathcal{E}_N \cap \mathcal{E}_L$ the intersection of these constraints corresponds to a rectangular annulus in the relative space of two agents as in Fig. 3.2a, where the shaded region denotes illegal configurations. Pairs of agents $(a_i, a_j) \in \mathcal{E}_N - \mathcal{E}_L$ (neighbors without collision constraint) will have only the maximum distance constraint (Fig. 3.2b). Pairs of agents $(a_i, a_j) \in \mathcal{E}_L - \mathcal{E}_N$ (non-neighbors with collision constraint) will have infinite annuli (Fig. 3.2c).

For groups with complete collision and connectivity graphs, the proximity constraints can be viewed as an annulus around every point in the physical space of an agent, describing the possible locations of other agents. The safe region of a

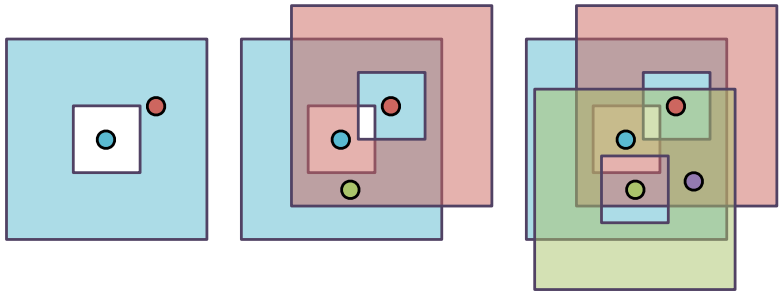
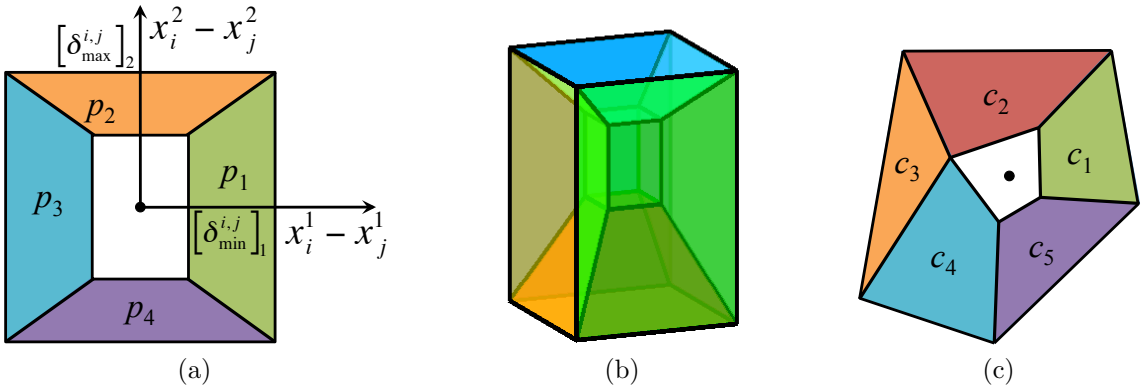


Figure 3.3: Intersection of annuli for multiple agents.

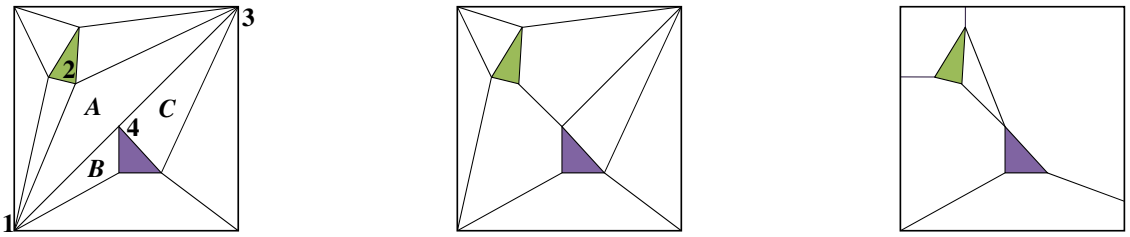
third agent, assuming complete collision and connectivity graphs, would be a square annulus around the original agent intersected with the annulus of the second agent. This is shown in Fig. 3.3 for up to four agents with complete graphs.

Figure 3.4 illustrates nonsymmetric collision and connectivity constraints in the relative space of two agents in 2D and 3D. If no communication constraints are imposed, the regions will be infinite (i.e. $\boldsymbol{\delta}_{\max} = \infty$). In Fig. 3.4a and 3.4c the regions are shown numbered to facilitate the graph search in Section 3.2 (numbers are left off of Figure 3.4b for clarity). Note that as in Figure 3.4b and 3.4c the constraints need not be symmetric. Any convex decomposition with matching facets is admissible. By matching facets we mean that any hyperplane supporting two adjacent polytopes shares the same vertices in both polytopes, and any hyperplane can only support one facet of that polytope. This is illustrated in Fig. 3.5: decomposition 3.5a is an illegal decomposition, since facet (1,3) has an extra vertex from polytopes $\boldsymbol{B}$ and $\boldsymbol{C}$ on the right (i.e. what is one facet on $\boldsymbol{A}$ is two separate facets on $\boldsymbol{B}$ and $\boldsymbol{C}$). The decompositions in Fig. 3.5b and Fig. 3.5c are both legal since all facets match exactly on both sides. Matching facets are necessary to ensure that the state exits the polytope and predictably enters the next polytope on the path. The controller we use cannot restrict the state from exiting via a portion of the exit facet; the entire facet can be used for exit. Thus if the desired path were to enter $\boldsymbol{B}$ from $\boldsymbol{A}$, then $\boldsymbol{A}$



(a) (b) (c)

Figure 3.4: Nonsymmetric collision and connectivity constraints in the relative space of two agents in 2D and 3D. (a) 2D collision and connectivity constraints used in simulations. (b) 3D example of collision and connectivity constraints used in simulations. (c) Asymmetrical constraints in 2D.



(a) (b) (c)

Figure 3.5: Possible decompositions of a space. Panel 3.5a depicts an illegal decomposition (inconsistent number of facets on (1,3)). Panels 3.5b and 3.5c show legal decompositions.

must share an entire facet with $\boldsymbol{B}$ or the state could enter $\boldsymbol{C}$.

Although examples in this work will use proximity constraints of the type $|\mathbf{x}_i - \mathbf{x}_j| \in [\boldsymbol{\delta}^{i,j}_{\min}, \boldsymbol{\delta}^{i,j}_{\max}]$, this framework can be used on any proximity constraints which can be defined as unions of non-overlapping polytopes with matching facets, such as those shown in 3.4c. Note that the proximity constraints are defined on pairs of robots. Therefore, there will be n choose 2, or $\binom{n}{2} = \frac{n!}{2(n-2)!}$ sets of proximity constraints.

Definition 3.1.4. *The* configuration space $\mathcal{C}_i$ *of an agent* a_i *is the set of all transformations of the agent. The* free space $\mathcal{C}_i^{free}$ *of* a_i *is the set of all transformations of* a_i *which do not intersect with obstacles in the configuration space.*

Note that since we model robots as points, the environment must be padded appropriately to ensure the robots do not collide with obstacles in or the physical boundary of the environment. Therefore, the free space of a robot must take into account the extent of the robot and have a reduced size, so that at all configurations within free space, the robot must not collide with the boundary or an obstacle at any orientation. Note that to apply these controllers to nonholonomic robots we use feedback linearization, which will require padding the environment according to the feedback linearization distance; we discuss this in detail in Chapter 5.

We assume $\mathcal{C}_i^{free}$ is tessellated into p_i polytopes with matching facets.

Definition 3.1.5. *The* group configuration space *is the Cartesian product of the configuration spaces of each agent,*

$$\mathcal{C}_{all} = \mathcal{C}_1^{free} \times \mathcal{C}_2^{free} \times \cdots \times \mathcal{C}_n^{free}$$
$$\mathbf{x} = [\mathbf{x}_1^T \ \mathbf{x}_2^T \ \cdots \ \mathbf{x}_n^T]^T \in \mathcal{C}_{all}.$$

Thus the configuration of all n agents is described by a single point in $\mathcal{C}_{all} \in \mathbb{R}^D$, $D \equiv \sum_{i=1}^n d_i$, which contains $\prod_{i=1}^n p_i$ polytopes. In general, the team of agents can be heterogeneous; thus they might not share the same configuration space or dimension so d_i is not necessarily equal to d_j.

In the group configuration space, we can enforce other constraints, such as *mutual exclusion.*

Definition 3.1.6. *The* mutual exclusion graph *on the set of agents is the static graph* $G_M = (\mathcal{V}_a, \mathcal{E}_M)$ *where* $\mathcal{E}_M$ *is the set of all pairs of agents which cannot simultaneously*

occupy the same polytope in $\mathcal{C}_i = \mathcal{C}_j$ *if they share the same decomposition, or in* $\mathcal{C}_i^{free} = \mathcal{C}_j^{free}$ *if they share the same free space. The constraint can be written*

$$\mu(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 0, & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E}_M \text{ in same polytope} \\ 1, & \text{otherwise.} \end{cases}$$

We can also enforce other constraints, such as setting a maximum or minimum number of robots in a location, excluding certain robots from areas, etc. We represent all of these constraints as

$$o(\mathbf{x}) \leq 0.$$

Note that since we desire a space composed of polytopes, only constraints which can be constructed of halfspaces are admissible.

We can rewrite all of our constraints with respect to the group configuration space:

$$\begin{aligned} \mathcal{L} &\equiv \{\mathbf{x} | \mathbf{x} \in \mathcal{C}_{all}, \lambda(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \ \forall (a_i, a_j) \in \mathcal{E}_L\}, \\ \mathcal{M} &\equiv \{\mathbf{x} | \mathbf{x} \in \mathcal{C}_{all}, \mu(\mathbf{x}_i, \mathbf{x}_j) = 1 \ \forall (a_i, a_j) \in \mathcal{E}_M\}, \\ \mathcal{N} &\equiv \{\mathbf{x} | \mathbf{x} \in \mathcal{C}_{all}, \nu(\mathbf{x}_i, \mathbf{x}_j) \leq 0 \ \forall (a_i, a_j) \in \mathcal{E}_N\}, \\ \mathcal{O} &\equiv \{\mathbf{x} | \mathbf{x} \in \mathcal{C}_{all}, o(x) \leq 0\}. \end{aligned} \tag{3.3}$$

The navigation problem for the group of robots with the above described constraints is given by Problem 3.1.7 for a kinematic group of robots, and Problem 3.1.9 for a dynamic group of robots.

Problem 3.1.7 (Full state feedback for kinematic systems)**.** *Consider a group of* n *kinematic robotic agents with dynamics* (3.1)*. For any initial state, find a piecewise smooth state feedback policy* $\mathbf{u}(\mathbf{x})$ *that drives the system to the goal configuration* $\mathbf{x}^g$

such that:

1. $\forall t \in [0, T_0]$ $\mathbf{x} \in \mathcal{C}_{all}$*;*

2. $\dot{\mathbf{x}} = \mathbf{u}$*;*

3. $\mathbf{x} \in \mathcal{L} \cap \mathcal{M} \cap \mathcal{N} \cap \mathcal{O}$*;*

4. $\mathbf{x}(T_0)$ *arbitrarily close to* $\mathbf{x}^g$*.*

Note that in Problem 3.1.7 we can let $\mathbf{u}$ be any function of the state of the team. In other words, the team of robots must respect the desired connectivity, collision, and mutual exclusion graphs, but without limitations on the state information available to the agents, so that agents can share all state information through the communication graph via indirect communication. Problem 3.1.8 restricts the inputs to each agent to contain only information accessible to that agent through the information graph.

Problem 3.1.8 (Partial state feedback for kinematic systems)**.** *Consider Problem 3.1.7 with the additional constraint on* $\mathbf{u}$

5. $\mathbf{u}_i$ *does not depend explicitly on* $\mathbf{x}_j$ *if* $(a_i, a_j) \notin \mathcal{E}_I$*.*

Problem 3.1.9 (Full state feedback for dynamical systems)**.** *Consider a group of* n *dynamic robotic agents with dynamics* (3.2)*. For any initial state, find a piecewise smooth state feedback policy* $\mathbf{u}(\mathbf{x})$ *that drives the system to the goal configuration* $\mathbf{x}^g$ *such that:*

1. $\forall t \in [0, T_0]$ $\mathbf{x} \in \mathcal{C}_{all}$*;*

2. $\ddot{\mathbf{x}} = \mathbf{u}$*;*

3. $\mathbf{x} \in \mathcal{L} \cap \mathcal{M} \cap \mathcal{N} \cap \mathcal{O}$*;*

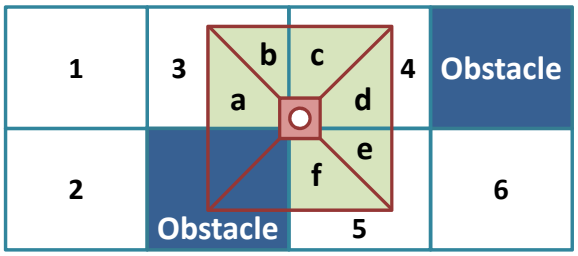


Figure 3.6: Example of a combination goal requirement for two agents. Either agent must reach the goal in polygon 4. Proximity constraints require the other agent to be in one of the regions a-f.

4. $\mathbf{x}(T_0)$ *arbitrarily close to* $\mathbf{x}^g$.

In Problems 3.1.8, 3.1.7, and 3.1.9, we do not threshold the inputs $\mathbf{u}$; therefore, the resulting input can be very high. We can preserve the behavior of the controller by scaling the *entire* input to be below a desired threshold. Note that individual robot inputs cannot be scaled, since this would not preserve the guarantees.

For situations when the goal positions are not specifically assigned to each agent, for example, when only m of n agents are required to reach a goal location, we have a finite number of goal nodes as opposed to a single goal node. Let $\mathcal{X}^g$ be the set of goal nodes. We can then replace $\mathbf{x}^g$ by the set $\mathcal{X}^g$ in Problems 3.1.7–3.1.9 to allow a set of goal configurations. Figure 3.6 depicts an example where either one of two agents must reach the goal configuration in polygon 4. The proximity constraints limit the location of the other agent to the 6 polygons a-f. Thus we have 12 possible goal nodes, six for each agent at the goal.

To solve Problems 3.1.7, 3.1.8, and 3.1.9, we remove from $\mathcal{C}_{all}$ points that violate the constraints, and synthesize controllers on the resulting space to drive the system to the goal configuration.

Definition 3.1.10. *The* task configuration space $\mathcal{C}_T$ *is the set*

$$\mathcal{C}_T = \mathcal{C}_{all} \cap \mathcal{L} \cap \mathcal{M} \cap \mathcal{N} \cap \mathcal{O}. \tag{3.4}$$

Note that because the constraints (3.3) *are described by halfspaces, the task configuration space is composed of polytopes. In* $\mathcal{C}_T$ *the robots cannot collide with each other or obstacles in the space, lose communication, or violate the constraints* $\mathcal{M}$ *and* $\mathcal{O}$*.*

We will also refer to the polytopes in the task configuration space as *cells*.

We would like to synthesize feedback controllers to solve Problems 3.1.7, 3.1.8, and 3.1.9. In other words, we ensure that the agents are always inside the group configuration space $\mathcal{C}_T$ and that they reach the goal configuration. There are two stages in this process. First, we pursue a discrete representation of $\mathcal{C}_T$ and find paths in this discrete representation. In Chapters 5 and 6, we translate these paths into feedback controllers.

3.2 Discrete planning on task configuration space

In this section we build a discrete representation of the task configuration space $\mathcal{C}_T$ and determine the lowest cost path using a graph search algorithm. The key step is to define an adjacency graph on the set of polytopes.

Definition 3.2.1. *The* polytope graph $G_P = (\mathcal{V}_P, \mathcal{E}_P)$ *on the polytopes in* $\mathcal{C}_T$ *is the pair of sets* $\mathcal{V}_P = \{P^0, P^1, P^2 \dots\}$*, where* P^m *is the* m*-th polytope, and* $\mathcal{E}_P = \{e_P^{m,k} | P^m$ *is adjacent to* P^k*,* $\forall m, k\}$*, the set of all pairs of polytopes which share a* (*matching*) *facet.*

If the number of robots is small ($n \leq 3$), it may be possible to compute the

entire task configuration space. In this case, we can search for a path to the goal configuration from any initial configuration.

Problem 3.2.2 (Discrete Global Path)**.** *For all nodes $P^k \in \mathcal{V}_P$, find a path on the polytope graph G_P to the goal node P^g.*

For larger groups of robots ($n > 3$), computing the entire $\mathcal{C}_T$ can be prohibitively complex since the maximum number of polytopes is exponential in n, as we discuss in Section 3.3. Therefore, we use a heuristic-based graph search to find a path from the configuration to the goal configuration, without computing all of $\mathcal{C}_T$. Without loss of generality, let P^0 be the polytope the state is initially in.

Problem 3.2.3 (Discrete Specific Path)**.** *For the node $P^0 \in \mathcal{V}_P$, find a path on the polytope graph G_P to the goal node P^g.*

The path to the goal polytope on the polytope graph determines the exit facet for each polytope.

To solve Problem 3.1.8 specifically, we must triangulate the polytopes P^k into simplices s_q^k due to constraints on sharing state information. The reason for this will become clear in the discussion of the controller. We now define an adjacency graph on the set of simplices in each polytope.

Definition 3.2.4. *The k-th* simplex graph $G_S^k = (\mathcal{V}_S^k, \mathcal{E}_S^k)$ *on the simplices s_q^k of the fixed triangulation of polytope P^k is the pair of sets $\mathcal{V}_S^k = \{s_1^k, s_2^k, \dots\}$, where s_q^k is the q-th simplex $s_q^k \subseteq P^k$, and $\mathcal{E}_S^k = \{e_S^{k(q,r)} | s_q^k$ adjacent to $s_r^k,\ \forall q, r\}$, the set of all pairs of simplices which share a facet. We use the simplex graph to determine a discrete path from each simplex in a polytope's triangulation to the simplices on its exit facet.*

Since there are multiple paths to the goal, we can use the usual notion of a distance on a graph to find the shortest paths in the polytope graph using a graph search algorithm such as Dijkstra [27] or A* [44]. We can associate with each edge $e_P^{k,m} \in \mathcal{E}_P$ between adjacent polytopes P^k and P^m a cost, $\texttt{DIST}(P^k, P^m)$, which we minimize.

To minimize the number of controller transitions between polytopes, one can choose the path which minimizes the number of polytopes that are visited, therefore $|e_P^{k,m}| = 1, \forall e_P^{k,m} \in \mathcal{E}_P$. This is desirable when using the affine feedback control we will introduce in the following chapter since the controller is discontinuous across polytope interfaces. Similarly we find the shortest path in the same sense on the simplex graphs to any of the simplices on the exit facet. In the goal polytope, P^g, we find the shortest path from any simplex to the goal simplex, s^g.

Alternatively, when dealing with a dynamical system such as a quadrotor whose model is an approximation, it may be more desirable to penalize paths which go through smaller cells.

For larger groups of robots ($n > 3$), instead of computing all of $\mathcal{C}_T$, we simultaneously build a discrete representation of the task configuration space and find a path in this representation using a heuristic-based graph search such as A* [44]. A* uses a distance-plus-cost heuristic to determine an order for expanding nodes in the graph; in our case, each node is a potential polytope in $\mathcal{C}_T$, and only as they are visited in the graph search do we compute the polytopes. We add a node to G_P only if the corresponding polytope is valid. In A*, an admissible heuristic function guarantees that the found path is optimal. A heuristic is admissible if it provides a lower bound on the actual cost to go from any single node to any other single node.

We exploit the fact that we are taking Cartesian products of known graphs to build the polytope graph online. The heuristic is defined by adjacency graphs in

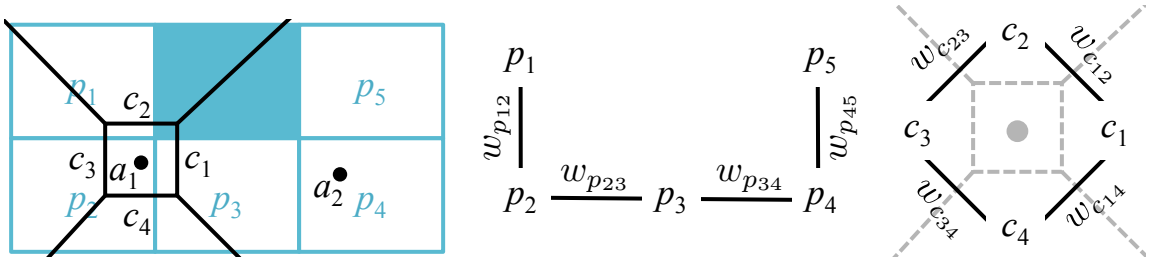


Figure 3.7: A 2D, two robot example of discrete system pose. (a) The two robots in the free configuration space, with proximity constraints. Here the discrete pose is $[p_2\ p_4\ c_1]$. (b) The adjacency graph on the agents' workspaces. (c) The adjacency graph on the agents' proximity constraints.

$\mathcal{C}_i^{free}$ and the graph on the agents' proximity constraints. We use the two robots with the environment and constraints shown in Figure 3.7a as an example. Figure 3.7b shows the adjacency graph on $\mathcal{C}_i^{free}$ and Figure 3.7c shows the graph on the agents' proximity constraints for the example in Figure 3.7a. We generate a discrete system pose by concatenating the location (cell number) of each robot in its free space with the relative location of each pair of robots. Each polytope P^m in $\mathcal{C}_T$ corresponds to a unique discrete pose $\mathcal{F}_d^m$ of the system.

In Fig. 3.7a, $\mathcal{F}_d = [\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_1]$, where $\mathtt{p}_2$ corresponds to the cell a_1 is in, $\mathtt{p}_4$ corresponds to the cell of a_2, and $\mathtt{c}_1$ corresponds to the relative location of a_2 with respect to a_1. Using the discrete pose and the adjacency graphs of the free spaces of the agents, we can generate adjacent cells. By assigning a cost of 1 to each transition, we are able to generate a cost heuristic for the A* algorithm. For example, since $\mathtt{p}_2$ is adjacent to $\mathtt{p}_1$ and $\mathtt{p}_3$, $[\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_1]$ is adjacent to $[\mathtt{p}_1\ \mathtt{p}_4\ \mathtt{c}_1]$ and $[\mathtt{p}_3\ \mathtt{p}_4\ \mathtt{c}_1]$. Similarly, since $\mathtt{p}_4$ is adjacent to $\mathtt{p}_3$ and $\mathtt{p}_5$, $[\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_1]$ is adjacent to $[\mathtt{p}_2\ \mathtt{p}_3\ \mathtt{c}_1]$ and $[\mathtt{p}_2\ \mathtt{p}_5\ \mathtt{c}_1]$. Finally, $\mathtt{c}_1$ is adjacent to $\mathtt{c}_2$ and $\mathtt{c}_4$, generating potential adjacent polytopes corresponding to $[\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_2]$ and $[\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_4]$. However, from Figure 3.7a it is easy to see these last two cells are not valid, and when we expand the node $[\mathtt{p}_2\ \mathtt{p}_4\ \mathtt{c}_1]$ in the A*

algorithm, we check to ensure the polytope exists before adding it to the open set. Therefore, finding a path on the polytope graph is equivalent to finding a path on the discrete poses of the system.

This induces a heuristic cost for going from the initial to the goal formation. If $\mathcal{F}_d^0 = [\mathtt{p}_1^0\ \mathtt{p}_2^0\ \cdots\ \mathtt{p}_n^0\ \mathtt{c}_1^0\ \cdots\ \mathtt{c}_{\binom{n}{2}}^0]$ and $\mathcal{F}_d^g = [\mathtt{p}_1^g\ \mathtt{p}_2^g\ \cdots\ \mathtt{p}_n^g\ \mathtt{c}_1^g\ \cdots\ \mathtt{c}_{\binom{n}{2}}^g]$ are the initial and desired discrete pose of the system, the minimum cost is

$$h\left(\mathcal{F}_d^0, \mathcal{F}_d^g\right) = \sum_{i=1}^{n} \mathtt{COST}(\mathtt{p}_i^0, \mathtt{p}_i^g) + \sum_{\kappa=1}^{\binom{n}{2}} \mathtt{COST}(\mathtt{c}_\kappa^0, \mathtt{c}_\kappa^g),$$

where $\mathtt{COST}(\mathtt{p}_i^0, \mathtt{p}_i^g)$ and $\mathtt{COST}(\mathtt{c}_\kappa^0, \mathtt{c}_\kappa^g)$ are the costs of moving from the initial to the final configurations on the adjacency graph of the workspace for each agent and on the adjacency graph of the proximity constraints for each pair of agents, respectively. Since the heuristic estimates the minimum path cost from P^0 to P^g without taking into account empty polytopes (which are subsequently not added to the graph), the heuristic will always be an underestimate to the actual cost. To minimize the number of transitions, we set all weights in the graph $w_{\mathtt{p}_j} \equiv w_{\mathtt{c}_\kappa} \equiv 1$.

Theorem 3.2.5. *Problem 3.2.2 has a solution to any goal from any initial configuration if and only if G_P is connected.*

Proof. $\mathcal{C}_T$ contains every allowable configuration $\mathbf{x}$ in our polytopic world model. G_P contains all the information about the connectivity of $\mathcal{C}_T$. Thus, if there is a solution to Problem 3.2.2 there must exist a path from any node in G_P to the goal node(s). Conversely if there is no path on G_P between any two nodes, there is no solution to Problem 3.2.2. □

Corollary 3.2.6. *Problem 3.2.3 has a solution from P^0 to P^g if and only these nodes are connected on G_P.*

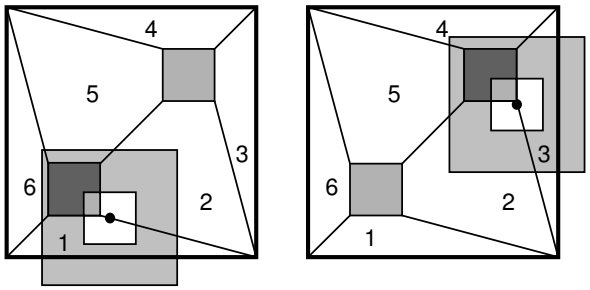


Figure 3.8: Distant pairs of polytopes.

3.3 Task configuration space complexity

The computational complexity of computing the task configuration space, in its entirety, is determined by the number of polytopes in $\mathcal{C}_T$, which scales exponentially with n. $\mathcal{C}_{all} \in \mathbb{R}^D$ contains $\prod_{i=1}^{n} \mathtt{p}_i$ polytopes, where $\mathtt{p}_i$ is the number of polytopes in $\mathcal{C}_i^{free}$. Assuming the proximity constraints shown in Fig. 3.2, for each pair of agents with collision constraints we have one annulus with 4 regions, resulting in a maximum of $4^{n(n-1)/2}$ proximity regions intersected with $\mathcal{C}_{all}$. Thus, the maximum number of polytopes in $\mathcal{C}_T$ is

$$P_{\max} = 4^{n(n-1)/2} \prod_{i=1}^{n} p_i. \tag{3.5}$$

This is a worst-case scenario, as the proximity constraints are dependent ($x_1^1 < x_2^1, x_2^1 < x_3^1 \implies x_1^1 < x_3^1$). Additionally, a portion of these polytopes will violate proximity constraints. If the physical space is larger than the maximum communication distance and two agents are neighbors, then they cannot be at opposite ends of the space. Figure 3.8 depicts a situation of this type for two neighboring agents. If one agent is in region 1, the other cannot be in region 4 since no point in 4 is within the constraints, and vice versa (similarly for regions 3 and 6). Any product of polytopes in neighboring agents' configuration spaces that are beyond

the maximum connectivity limit will be eliminated when intersecting $\mathcal{C}_{all}$ with the proximity constraints (3.3) to create $\mathcal{C}_T$.

Table 3.1: Complexity of constructing the task configuration space.

Task	**Complexity**	**Ref**
Construct $\mathcal{C}_i^{free}$	$\mathcal{O}(v_i + \min\{v_i, r_i^2, r_i^4\})$	[51]
Construct $\mathcal{C}_T$	$\mathcal{O}(P_{\max} \cdot LP(h_i + 1, d))$	[38]
Plan on polytopes in $\mathcal{C}_T$	$\mathcal{O}(\lvert\mathcal{E}_P\rvert + P_{\max} \log P_{\max})$	[20]
Triangulation, per polytope	$\mathcal{O}(\lvert V_k\rvert^{D/2})$	[38]
Plan on simplex graph $\mathcal{G}_S^k$	$\mathcal{O}(\lvert\mathcal{E}_S^k\rvert + \lvert S_k\rvert \log \lvert S_k\rvert)$	[20]

Table 3.1 specifies the complexity of every step in the process of constructing the entire $\mathcal{C}_T$. Here, v_i (resp. r_i) is the total number of vertices (resp. reflex vertices) in $\mathcal{C}_i^{free}$, represented by a quasi-in-simple polygon. LP(c, d) represents the complexity of a linear program in d dimensions and c constraints. h_k is the number of inequalities used to describe a polytope in $\mathcal{C}_T$. $|V_k|$ is the number of vertices in polytope P^k. $|S_k|$ is the number of simplices in the Delaunay triangulation of P^k.

The computational expense of the process depends largely on the methods used for decompositions, Cartesian products, and intersections, as well as the number of agents, connectivity, and complexity of the space.

There are several ways to decrease the computation time required. Combinations of polytopes which violate proximity constraints can be ruled out before taking the Cartesian product. Using a heuristic-based graph search can also reduce the computation.

3.4 Remarks

This chapter presented a method for constructing the configuration spaces for a group of robots. The task configuration space we construct is a space composed of

polytopes, where the agents cannot collide, lose communication, or violate any of the set constraints. Any state that is within the task configuration is considered a safe state.

We choose to model the configuration space as the Cartesian product of the configuration spaces of the individual robots in order to preserve as many solutions as possible. Since the task configuration space is a space composed of polytopes with matching facets, we can automatically synthesize controllers on this space which drive the group to the goal configuration without exiting the space in Chapters 5 and 6. This means the group will avoid violating any of the set constraints, inter-robot collisions, and collisions with the environment.

One limitation of this algorithm is that the complexity is exponential in the number of agents. Because of the size of $P_{\max}$, constructing $\mathcal{C}_T$ is the most time consuming portion of preprocessing. By using a heuristic-based graph search such as that presented in Section 3.2, we can significantly decrease precomputation time. Furthermore, although $P_{\max}$ is exponential in the number of agents n, the proximity constraint dependency combined with connectivity constraints significantly decrease the number of polytopes in the space, as we will show in Chapter 5.

In the following chapter, we construct a different configuration space specifically for a group of robots without unique goal positions. Then, we will translate the paths on the polytope and simplex graphs into feedback controllers to solve Problems 3.1.7, 3.1.8, and 3.1.9.

Chapter 4

Group Configuration Space Modeling

In this chapter we construct configuration spaces specifically for groups of robots by using an abstraction. The abstraction defines a virtual boundary for the group of robots which is used to navigate the group through the space. By this construction, we are naturally able to establish bounds on the positions of the robots. This allows us to guarantee safety–the controllers are designed so that the virtual boundary associated with the abstraction does not pass through the obstacles. Thus the complexity of the problem of synthesizing controllers is *independent* of the number of robots, which promises scalability to large groups.

4.1 Problem formulation

Consider again a group $\mathcal{G}$, of n kinematic agents with dynamics (3.1). The group must together navigate an obstacle-filled environment to a specified task location. Since the group is working closely together, we assume that communication occurs

very rapidly within the group, so that control can be centralized over the group.

We use an abstraction on the group of robots to reduce the computational complexity of the problem of navigating n robots. Our abstraction defines a virtual adaptive *boundary* for the group of robots, while controllers at the robot level ensure the robots stay within the boundary. The abstraction boundary is determined by automatically synthesized controllers, and the robots react to the changing size and shape of the abstraction boundary. Thus, a group of robots can navigate a space knowing only the boundary and the local (within the boundary) state of the group, decoupling planning and control of the agents from the physical environment. The dimension of the abstraction is independent of the number of robots n. We discuss in detail the specific abstraction we use in Section 4.2.

Figure 4.1 is a graphical representation of the hierarchical structure of the approach. At the bottom level, individual robots execute the continuous controllers that are designed to satisfy specifications and desired formation properties. At the middle level, individual robots interact with each other to maintain constraints (this level may not be necessary for all problems). At the top level, the abstracted group navigates the space while maintaining constraints designed to ensure there is enough room for the robots. We sum controllers at these levels to generate the input for the individual robots. We assume for now that there are no obstacles within the group boundary.

Our hierarchical control system enables a multiple time scale approach, as shown in Fig. 4.1, to eliminate the possibility of local minima which may occur when summing two controllers. Group dynamics (motion within a group) are assumed to evolve on a much faster time-scale than abstraction dynamics (overall motion of an entire group); sufficient time scale separation between group and abstraction dynamics ensures the two controllers can be designed independently.

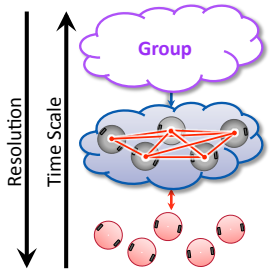


Figure 4.1: Hierarchical structure. At the bottom level, robots implement the continuous controller. At the top level, the abstracted group navigates the space.

Since the abstraction is independent of the robots' configuration, synthesis of and planning in the abstraction workspace occurs in advance. At least one robot must have knowledge of the evolution of the abstraction over time. This information, as well as individual robot state information, propagates through the group rapidly via explicit communication.

4.2 Geometric abstraction

The abstraction defines a virtual boundary for the group of robots; the boundary, in turn, defines an obstacle-free configuration space for the individual robots in the group. Since group navigation is handled by a high-level controller, which treats the abstraction as a single deforming robot, the individual robots must only keep up with the group motion while they interact with each other to maintain the desired constraints. We use a rectangle for this virtual boundary; however, this is only one example of the possible choices for abstractions.

Definition 4.2.1. *The* group abstraction **A** *is a triple*

$$A = (\mathbf{x}_A, \theta, \mathbf{s}) \in SE(2) \times \mathbb{R}^2, \tag{4.1}$$

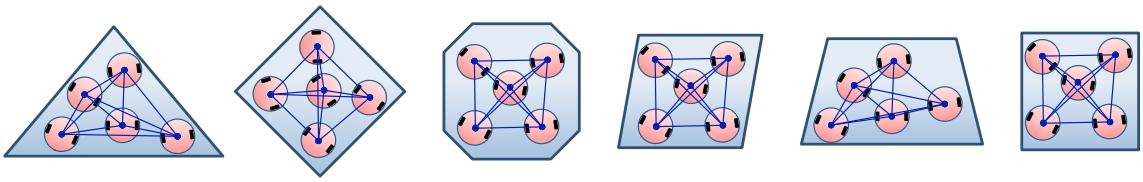


Figure 4.2: Examples of possible abstraction boundaries. We choose a rectangle since it can be described by width and height alone.

where $\mathbf{x}_A$ *is the center of the group abstraction,* θ *is the angular orientation, and* $\mathbf{s}$ *is a shape vector representing the boundary and size of the abstraction which encloses the group of robots. We assign the abstraction dynamics*

$$\dot{\mathbf{A}} = \mathbf{u}_A, \tag{4.2}$$

where $\mathbf{u}_A$ *can be considered a virtual input.*

In this work we choose a rectangle as the abstraction boundary since a rectangle can be described by two parameters, width s_w and height s_h, so that the shape vector is the pair $\mathbf{s} = (s_w, s_h)$. Although we choose a rectangle, it is important to note that the rectangle can be replaced by any convex polytopic shape, as we show in Fig. 4.2, provided the shape vector contains enough information to describe a unique boundary.

We treat the abstraction as a single robot which can change shape and orientation while navigating the space. *Shape constraints* limit the size and shape of the abstraction in order to ensure we have enough room for the number of robots in the group. We set bounds on s_w and s_h, as well as bounds on perimeter, $(s_w + s_h)$. We can write these constraints:

$$H_s\mathbf{s} \leq K_s. \tag{4.3}$$

We choose perimeter bounds instead of area since perimeter is a linear constraint.

The reason for linear constraints will become clearer in Section 4.3.

The input to each agent in the global reference frame is

$$\mathbf{u}_i = R(\theta)\ \mathbf{u}_i^r + \mathbf{u}_A^{x,y}, \tag{4.4}$$

where $\mathbf{u}_A^{x,y}$ is the translational component of the abstraction input, $R(\theta)$ is the rotation matrix at θ, and $\mathbf{u}_i^r$ is the individual input of robot a_i in the local coordinate frame defined by the abstraction.

4.3 Abstract configuration space

We would like to drive the abstraction through the physical workspace using a controller that can be synthesized automatically. To that end, we build a polytopic configuration space based on the parameters which define the abstraction.

Definition 4.3.1. *The configuration space of the abstraction, $\mathcal{C}$, is the set of all transformations, including rotations, of the virtual boundary defined by A. The free space of the abstraction, $\mathcal{C}^{free}$, is the set of all transformations, including rotations, of the virtual boundary defined by A which do not intersect with obstacles in the configuration space.*

To simplify building and planning on $\mathcal{C}^{free}$, we take a hierarchical approach to constructing it. First, we slice the angular component of the configuration space into θ-slices, $\Theta^k = [(k-1)\Delta\theta, k\Delta\theta], k = \{1, \ldots, q\}$, so that $q\Delta\theta = 2\pi$, and tesselate the $\mathbb{R}^2$ workspace into districts $D^m, m \in \{1, \ldots, p\}$, (Fig. 4.3a). Each district is described by constraints:

$$H_D^m \mathbf{x}_A \leq K_D^m,\ m \in \{1, \ldots, p\}. \tag{4.5}$$

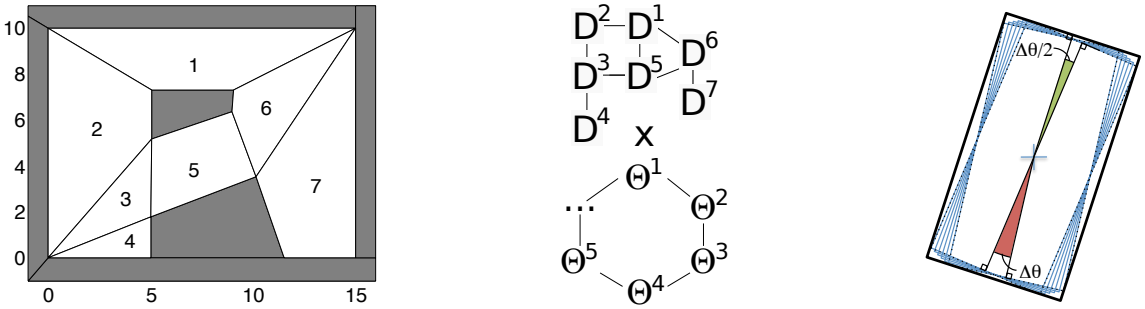


Figure 4.3: (a) A decomposition of the workspace into districts (Obstacles are shaded) (b) We take the Cartesian Product of each district D^m with each θ-slice Θ^k. (c) Our choice for overestimating the abstraction.

Representing the free space exactly is not possible with a finite number of polytopes, since the free space includes curves due to the angular component of the abstraction. Therefore, we seek to underestimate $\mathcal{C}^{free}$ with $\mathcal{C}_A^{free}$, such that $\mathcal{C}_A^{free} \subset \mathcal{C}^{free}$. We do this by overestimating the abstraction for each Θ^k with $\mathbf{A}_{\Theta^k}$:

$$\mathbf{A}_{\Theta^k}(\mathbf{x}_A, \mathbf{s}) \supset \{\mathbf{A}(\mathbf{x}_A, \theta, \mathbf{s}) | \theta \in [(k-1)\Delta\theta, k\Delta\theta]\}, k \in \{1, \ldots, q-1\}.$$

$\mathbf{A}_{\Theta^k}(\mathbf{x}_A, \mathbf{s})$ must be a union of convex polygons, the vertices of $\mathbf{A}_{\Theta^k}$ must be linear functions of x_A and s, and the outward normals must not be a function of s. This is necessary for computing the applicability conditions for types A and B contact [63].

We choose to overestimate the abstraction with a rectangular superset of the abstractions through an angular interval $\Delta\theta$. Figure 4.3c shows the abstraction overestimate that we choose, which is a rectangle rotated to angle $\Delta\theta/2$ (smallest rectangular superset of the abstractions). The overestimate shown is appropriate for small $\Delta\theta$; with large $\Delta\theta$, this overestimate becomes excessive and another overestimate should be chosen.

Workspace obstacles $O^l, l \in \{1, \ldots, o\}$ must be represented as a finite unions of

convex polygons (note that the boundary of the workspace is considered an obstacle). Obstacles in the workspace map to C-obstacles in the configuration space for each Θ^k. The C-obstacle for each θ-slice is computed using the specific overestimate for that θ-slice.

To construct C-obstacles, we follow an algorithm originally proposed by Lózano-Perez [77] and discussed by Latombe [63]. To do so, we must project the vertices and outward normals of the boundary into polyhedral space, thus the outward normals cannot depend on the shape vector. Here we follow Latombe's notation closely.

Let the vertices of the overestimated abstraction be represented $\alpha_j^k(\mathbf{x}_A)$, where $j = 1, \ldots, 4$ for slice Θ^k, and the outward normal for the facet between $\alpha_j^k(\mathbf{x}_A)$ and $\alpha_{j+1}^k(\mathbf{x}_A)$ as $\overrightarrow{\mathbf{n}}_j^{\alpha^k}(x_A)$. Similarly, let the vertices of obstacle O^l be represented β_j^l, with the outward normal for the facet between β_j^l and β_{j+1}^l as $\overrightarrow{\mathbf{n}}_j^{\beta^l}$.

The C-obstacles are computed by calculating for each Θ^k the applicability conditions for type A contact (we drop the superscripts k and l for clarity),

$$\texttt{APPL}_{i,j}^{\alpha}(\mathbf{x}_A) = \left[\overrightarrow{\mathbf{n}}_i^{\alpha}(\mathbf{x}_A) \cdot (\beta_{j-1} - \beta_j) \geq 0\right] \bigwedge \left[\overrightarrow{\mathbf{n}}_i^{\alpha}(x_A) \cdot (\beta_{j+1} - \beta_j) \geq 0\right].$$

If $\texttt{APPL}_{i,j}^{\alpha}(\mathbf{x}_A)$ holds, then add to C-obstacle the constraint

$$f_{i,j}^{\alpha}(\mathbf{x}_A) \equiv \overrightarrow{\mathbf{n}}_i^{\alpha}(x_A) \cdot (\beta_j - \alpha_i(x_A)) \leq 0. \tag{4.6}$$

Similarly, we calculate

$$\texttt{APPL}_{i,j}^{\beta}(\mathbf{x}_A) = \left[(\alpha_{i-1}(\mathbf{x}_A) - \alpha_i(\mathbf{x}_A)) \cdot \overrightarrow{\mathbf{n}}_j^{\beta} \geq 0\right] \bigwedge \left[(\alpha_{i+1}(\mathbf{x}_A) - \alpha_i(\mathbf{x}_A)) \cdot \overrightarrow{\mathbf{n}}_j^{\beta} \geq 0\right].$$

If $\texttt{APPL}_{i,j}^{\beta}(\mathbf{x}_A)$ holds, then add to the C-obstacle the constraint

$$f_{i,j}^{\beta}(\mathbf{x}_A) \equiv \overrightarrow{\mathbf{v}}_j^{\beta} \cdot (\alpha_i(\mathbf{x}_A) - \beta_j) \leq 0. \tag{4.7}$$

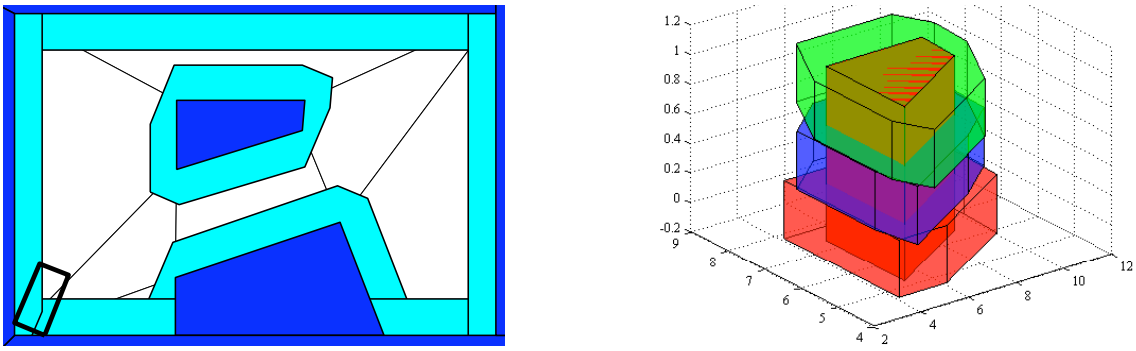
For more details on the applicability condition and the constraints $f_{i,j}^{\alpha}(\mathbf{x}_A)$ and $f_{i,j}^{\beta}(\mathbf{x}_A)$, refer to [63].

This generates obstacles in $\mathbb{R}^4 \times \mathbb{N}$ for each θ-slice, which we can extrude through the interval Θ_k to get $O^{l,k} \in \mathbb{R}^2 \times SE(2)$. We can represent $O^{l,k}$

$$H_O^{l,k}\mathbf{x}_A \leq K_O^{l,k}, k \in \{1, \ldots, q\}. \tag{4.8}$$

As depicted in Fig. 4.4b, the C-obstacles will generally not coincide at the θ-slice interfaces. Since our controller requires facets to match on adjacent polytopes, we reconcile this by considering each interval Θ_k with corresponding $\mathbb{R}^4$ obstacles, and any intersecting districts. In each district, before removing obstacles, we extend all obstacle hyperplanes in the intersection of the obstacle and the district for the adjacent θ-slices. For example, in $\Theta^3 \times D^7$, we intersect the polytopes with the extended hyperplanes for the $\Theta^2 \times D^7$, $\Theta^3 \times D^7$, and $\Theta^4 \times D^7$. This results in a union of polytopes in 4D for each district and θ-slice.

Finally, the resulting polytopes in each district are extruded into their θ-slices to construct *θ-districts*: $D_m^{\Theta_k}, m \in \{1, \ldots, p\}, k \in \{1, \ldots, q\}$. In each θ-district we remove the polytopes which intersect with obstacles. Since the supporting hyperplanes of the intersections of the 4-dimensional districts with every obstacle in every θ-slice were extended into the district before extending into θ-space, the polytopes within each θ-district have matching facets. Furthermore, any pair of polytopes across a θ-slice interface which are adjacent will have matching vertices, ensuring that we



(a) (b)

Figure 4.4: Configuration space obstacles for θ-slices. (a) C-obstacles generated for a rectangular abstraction with fixed length, width, and angle (shown in the bottom left corner). (b) C-obstacles for θ-slices do not coincide on the interface.

cannot drive the abstraction into prohibited regions.

Thus, the overestimated free space is $\mathcal{C}_A^{free}$:

$$C_A^{free} = \bigcup_{k=1}^{q} \bigcup_{m=1}^{d} C_A^{free,k,m}, \tag{4.9}$$
$$C_A^{free,k,m} = \bigcap_{l=1}^{o} \left(D_m^{\Theta_k} \cap O^{l,k} \right).$$

$\mathcal{C}_A^{free}$ is a union of polytopes in which the abstraction cannot collide with any obstacles or the boundary of the space.

Problem 4.3.2 (Control for group abstraction)**.** *Consider the system* (4.2) *and goal group abstraction state* $\mathbf{A}^g$*. Find an input function* $\mathbf{u}_A : [0, T_0] \to \mathcal{U}_A$ *for some initial group abstraction state* $\mathbf{A}^0 \in C_A^{free,k^0,m^0} \subset \mathcal{C}_A^{free}$ *such that*

1. *for all time* $t \in [0, T_0]$, $\mathbf{A} \in \mathcal{C}_A^{free}$ *and* $\mathbf{A}(T_0)$ *arbitrarily close to* $\mathbf{A}^g$,

2. $\dot{\mathbf{A}} = \mathbf{u}_A$.

4.3.1 Discrete planning on abstract configuration space

As we did in Section 3.2 in the previous chapter for the task configuration space, we will now seek a discrete representation of the abstract configuration space.

Each polytope in $\mathcal{C}_A^{free}$ is associated with a θ-district $D_m^{\Theta_k}$. We define two adjacency graphs: one on $\mathcal{C}_A^{free}$ at the θ-district level, and one on each θ-district at the polytope level.

Definition 4.3.3. *The upper-adjacency graph on $\mathcal{C}_A^{free}$ is the triple $G_U = (\mathcal{V}_U, \mathcal{E}_U, C_U)$, where $\mathcal{V}_U = \{[1\ 1], [1\ 2], \ldots, [q\ p]\}$, $\mathcal{E}_U$ is the set of all pairs $(D_m^{\Theta_k}, D_{m'}^{\Theta'_k})$ of θ-districts which share an interface, and C_U is the cost associated with each edge in $\mathcal{E}_U$.*

Definition 4.3.4. *The lower-adjacency graph on each $D_m^{\Theta_k}$ is the triple $G_L^{k,m} = (\mathcal{V}_L^{k,m}, \mathcal{E}_L^{k,m}, C_L^{k,m})$, where $\mathcal{V}_L^{k,m} = \{P_1^{k,m}, P_2^{k,m}, P_3^{k,m}, \ldots\}$, where $P_i^{k,m}$ is the i-th polytope in $D_m^{\Theta_k}$, $\mathcal{E}_L^{k,m}$ is the set of all pairs of polytopes which share a facet, and $C_L^{k,m}$ is the cost associated with each edge in $\mathcal{E}_L^{k,m}$.*

We set the cost $C_L^{k,m} = 1\ \forall k, m$ to minimize transitions within each $\mathcal{C}_A^{free,k,m}$.

Problem 4.3.5 (Discrete Abstraction Path)**.** *For the initial group abstraction state $\mathbf{x}_A^0$, find a path on G_U and corresponding paths through θ-districts to the goal group abstraction state $\mathbf{x}_A^g$.*

Again, we use a graph search algorithm such as Dijkstra or A* to determine the lowest cost path to the goal. We first find a path on the upper-adjacency graph to the goal θ-district . Then we determine a path from every polytope in $D_m^{\Theta_k}$ to $D_m'^{\Theta'_k}$ on the lower-adjacency graph. The heuristic is similar to the one discussed in Section 3.2, except the discrete pose in the abstract configuration space consists of the districts D^m and θ-slices Θ^k instead of the individual robot free space cells

and relative location of robots. Additionally, it must be verified that the interface between the θ-districts exists before an edge is added to the upper adjacency graph.

4.4 Complexity

The complexity of this method is independent of the number of robots. It is dominated by the number of polytopes in $\mathcal{C}_A^{free}$. We extend the hyperplanes which support the intersections of districts with obstacles. The boundary of 2D C-obstacles is made of at most $\mathcal{O}(\mathrm{v}_\alpha, \mathrm{v}_\beta)$ edges where v_α is the number of vertices in the abstraction, and v_β the number of vertices in the obstacle. We construct C-obstacles in $\mathbb{R}^4$, since we consider changing sizes of the abstraction. Since each hyperplane may divide the existing polytopes into two, the maximum number of polytopes in one θ-slice of $\mathcal{C}_A^{free}$ is $\mathcal{O}(2^{\mathrm{v}_\alpha \mathrm{v}_\beta} q)$ where q is the number of θ-slices.

4.5 Remarks

In this chapter, we presented a method for modeling groups of robots using an abstraction, and constructing the abstract configuration space. The key difference from Chapter 3 is that the goal configuration lends itself to navigating as a group or is not finely specified, whereas in Chapter 3, each robot had an individual goal configuration.

In the following chapters we translate the paths on the upper-adjacency and lower-adjacency graphs into feedback controllers to solve Problem 4.3.2.

Chapter 5

Decentralized Affine Feedback Control

In this chapter, we synthesize controllers to solve Problem 3.1.8 once the paths on the polytope and simplex graphs are identified. The synthesis procedure is similar in spirit to those discussed in [18, 43, 72, 105], but is closest to the one developed by Habets and van Schuppen [43] for determining a centralized affine state feedback that satisfies a set of inequalities on a polytope. In their paper, they derive controllers that drive a linear system from any initial condition in a polytope through a desired exit facet of the polytope while guaranteeing the system does not leave the polytope from any other facet. We modify this algorithm to design a decentralized affine controller within each simplex.

5.1 Local polytope controller

We now consider the subproblem of steering states in a simplex to a specified exit facet without limiting state information.

Problem 5.1.1 (Continuous Subproblem, Full Feedback)**.** *Consider the system* (3.1) *on simplex* $s_q^k \in P^k \in \mathcal{C}_T$*, where* s_q^k *is the* q*-th simplex in* P^k*, the* k*-th polytope on a path to the goal, and* $\mathbf{x}^g \notin s_q^k$*. Let* s_r^k *be the next simplex on the path. Let* Φ_q *be the facet shared by* s_q^k *and* s_r^k *with normal vector* $\mathbf{n}_q$ *pointing out of* s_q^k*. For any initial state* $\mathbf{x}_0 \in s_q^k$*, we have to find a time-instant* $T_q \geq 0$ *and an input function* $\mathbf{u} : [0, T_q] \to \mathbf{U}$*, where* $\mathbf{u}$ *is realized by the application of a continuous feedback law* $\mathbf{u}(t) = F\mathbf{x} + \mathbf{g}$*,* $F \in \mathbb{R}^{D\times D}$*,* $\mathbf{g} \in \mathbb{R}^{D\times 1}$*, such that*

1. $\forall t \in [0, T_q] : \mathbf{x}(t) \in s_q^k$*,*

2. $\mathbf{x}(T_q) \in \Phi_q$*, and* T_q *is the smallest time-instant in the interval* $[0, \infty)$ *for which the state reaches facet* Φ_q*,*

3. $\mathbf{n}_q^T \dot{\mathbf{x}}(T_q) > 0$*, i.e. the velocity vector* $\dot{\mathbf{x}}(T_q)$ *at* $\mathbf{x}(T_q) \in \Phi_q$ *has a positive component in the direction of* $\mathbf{n}_q$*.*

Note that Problem 5.1.1 naturally implies that all agents have access to the full state since the information graph is complete. We now consider the subproblem of steering states in a simplex to a specified exit facet with limited state information.

Problem 5.1.2 (Continuous Subproblem, Limited Feedback)**.** *Consider Problem 5.1.1 with the additional constraint:*

4. *matrix* F*, composed of matrices* $F^{ij} \in \mathbb{R}^{d_i \times d_j}, i, j = 1, \ldots, n$*, is such that* $F^{ij} = \mathbf{0}$ *if* $(a_i, a_j) \notin \mathcal{E}_I$*.*

In the goal simplex s^g, we solve the equation

$$\dot{\mathbf{x}}|_{\mathbf{x}=\mathbf{x}^g} = F\mathbf{x}^g + \mathbf{g} = \mathbf{0}. \tag{5.1}$$

Assume (without loss of generality) that the exit facet for each simplex has index 1, outward normal $\mathbf{n}_1$, and contains vertices $\{\mathbf{v}_2, \ldots, \mathbf{v}_{D+1}\}$. Problem 5.1.1 is solved on the simplex by the linear program

$$\begin{aligned} \min \quad & \mathbf{0}^{\mathrm{T}}[\mathbf{u}_1^{\mathrm{T}} \ \mathbf{u}_2^{\mathrm{T}} \ \ldots \ \mathbf{u}_{D+1}^{\mathrm{T}}]^{\mathrm{T}} \\ s.t. \quad & \mathbf{n}_1^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_c} > 0, \quad c \in \{2, \ldots, D+1\} \\ & \mathbf{n}_b^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_1} \leq 0, \quad b \in \{2, \ldots, D+1\} \\ & \mathbf{n}_b^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_c} \leq 0, \quad b, c \in \{2, \ldots, D+1\}, b \neq c, \end{aligned} \tag{5.2}$$

where $\{\mathbf{u}|_{\mathbf{v}_1}, \ldots, \mathbf{u}|_{\mathbf{v}_{D+1}}\}$ are the inputs evaluated at the vertices. The linear program (5.2) generates the inputs at the vertices of each simplex, which must be interpolated within the simplex (for more detail see [43]). Every point in a simplex is described by a unique convex combination of the vertices. The same convex combination of the inputs at the vertices is used to evaluate the controller within each simplex, and determines the calculation of F and $\mathbf{g}$. The feedback matrix F and vector $\mathbf{g}$ must be solved in each simplex, and thus are constant in each simplex.

Theorem 5.1.3. *Problem 3.1.7 has a solution if and only if Problem 3.2.2 has a solution.*

Proof. This proof is a straightforward extension of the results in [43]. □

Theorem 5.1.4. *Problem 3.1.8 has a solution if Problem 3.2.2 has a solution and there exists a corresponding solution to Problem 5.1.2 for each simplex $s_q^k \in \mathcal{C}_T$, as well as a solution to* (5.1) *for the goal simplex s^g.*

Proof. If there is a solution to Problem 3.2.2, there exists a path from any node in G_P to the goal node(s). For each polytope P^k which does not contain the goal, there exists a path in G_S^k to the exit facet of P^k defined by the solution to Problem 3.2.2, since the triangulation of a polytope results in a connected graph. Also,

there trivially exists a path on the simplex graph in P^g to the goal simplex s^g. Paths on the simplex graphs define the exit facet for each simplex in $\mathcal{C}_T$. If there exists a solution to Problem 5.1.2 for the *particular exit facet for each of these simplices*, and a solution for (5.1) in s^g, then Problem 3.1.8 has a solution. □

The constraint (4) in Problem 5.1.2 can be formulated as a supplementary equality constraint on the linear program used to solve for the inputs at the vertices of each simplex. Without requirement (4), F and $\mathbf{g}$ are calculated after solving the linear program, by using the equation

$$\begin{bmatrix} F^{\mathrm{T}} \\ \hline \mathbf{g}^{\mathrm{T}} \end{bmatrix} = WU \tag{5.3}$$

where

$$W \equiv \begin{bmatrix} \mathbf{v}_1{}^{\mathrm{T}} & 1 \\ \vdots & \vdots \\ \mathbf{v}_{D+1}{}^{\mathrm{T}} & 1 \end{bmatrix}^{-1} \equiv \begin{bmatrix} W_1{}^{\mathrm{T}} \\ \vdots \\ W_{D+1}{}^{\mathrm{T}} \end{bmatrix},$$

$$U \equiv \begin{bmatrix} \mathbf{u}|_{\mathbf{v}_1}{}^{\mathrm{T}} \\ \vdots \\ \mathbf{u}|_{\mathbf{v}_{D+1}}{}^{\mathrm{T}} \end{bmatrix} \equiv [U_1 \ \cdots \ U_{D+1}].$$

However, since we know certain entries of F must be zero, we restrict solutions of U accordingly. To more easily impose these constraints on F, we solve for U at the simplex level. (If we fix the triangulation, it is possible to solve for U at the polytope level by calculating the constraints on U at the simplex level but solving F at the polytope level; however, we do not explore this idea any further since we do not

object to non-differentiable controls at the facets.) Then we can write

$$F^{ij} = \begin{bmatrix} {W_{2j-1}}^{\mathrm{T}} U_{2i-1} & {W_{2j-1}}^{\mathrm{T}} U_{2i} \\ {W_{2j}}^{\mathrm{T}} U_{2i-1} & {W_{2j}}^{\mathrm{T}} U_{2i} \end{bmatrix}.$$

The linear program to solve Problem 5.1.2 is:

$$\begin{aligned} \min \ & \mathbf{0}^{\mathrm{T}}[\mathbf{u}_1^{\mathrm{T}} \ \mathbf{u}_2^{\mathrm{T}} \ \dots \ \mathbf{u}_{D+1}^{\mathrm{T}}]^{\mathrm{T}} \\ s.t. \ & \mathbf{n}_1^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_c} > 0, \quad c \in \{2, \dots, D+1\} \\ & \mathbf{n}_b^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_1} \leq 0, \quad b \in \{2, \dots, D+1\} \\ & \mathbf{n}_b^{\mathrm{T}}\mathbf{u}|_{\mathbf{v}_c} \leq 0, \quad b, c \in \{2, \dots, D+1\}, b \neq c, \\ & F^{ij} = \mathbf{0}, \ i, j \in \{1, \dots, n\}, (a_i, a_j) \notin \mathcal{E}_I, i \neq j. \end{aligned} \tag{5.4}$$

The equality constraint in (5.4) results directly from Problem 5.1.2 constraint (4). The linear program (5.4) has $2D(D+1)$ constraints: $D(D+1)$ inequality constraints and $D(D+1)$ equality constraints. Since we have $2D(D+1)$ unknowns, ($2D$ entries of F, D entries of $\mathbf{g}$, and $\mathbb{R}^D$ inputs at each of the $D+1$ vertices), we cannot guarantee that a solution to Problem 5.1.2 will be found. However, if in some simplex a solution is not found for a particular exit facet, it is possible to "reroute" the path on the simplex graph so that all states that enter that simplex exit through a different exit facet. Furthermore, it is possible to build G_P and all G_S^k based on the existence of controllers which drive the system to particular exit facets.

5.1.1 Algorithm for multi-robot navigation

In summary, the algorithm for controller synthesis or the solution to Problem 3.1.7 (resp. 3.1.8) involves the following four steps:

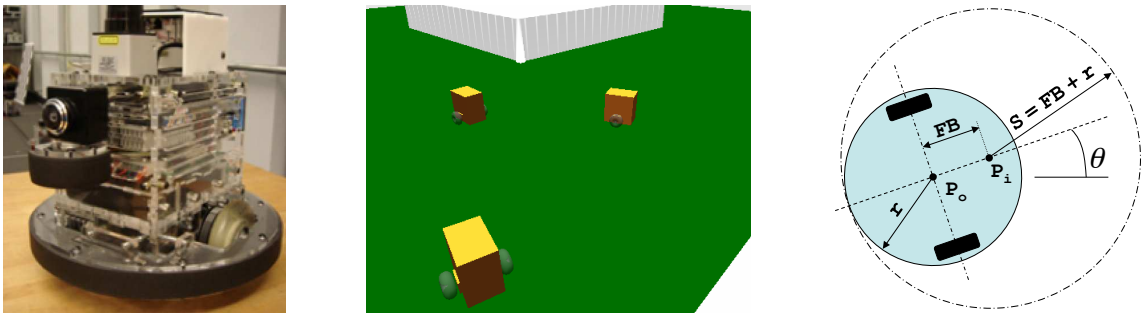
Algorithm 5.1.5.

1. *Construct the task configuration space $\mathcal{C}_T$ (Definition 3.1.10).*
2. *Find paths on the polytope graph G_P and all simplex graphs G_S^k.*
3. *Solve Problem 5.1.1 (resp. 5.1.2) on all simplices except s^g.*
4. *Solve Equation* (5.1) *in s^g.*

5.2 Simulations and experiments

In this section, we solve several multi-agent coordinated control problems to illustrate the application of the technique. Preprocessing of the controller is done in MATLAB using the Multi-Parametric Toolbox for polytope computations [61]. Three-dimensional dynamic simulation of the robots is done using GAZEBO, part of the PLAYER/STAGE/GAZEBO project [41]. GAZEBO is an open source multi-robot simulator, designed to accurately simulate a small population of robots with high fidelity. We use a MATLAB API [86] which interacts directly with GAZEBO and PLAYER to provide real-time control in Simulations and experiments, respectively.

The robot we use is the SCARAB robot, shown in Fig. 5.1a, which is a nonholonomic platform. Although the controller is designed for robots with the dynamics of (3.1), we use feedback linearization to provide inputs to the robots in (v, ω) form [52]. Fig. 5.1c shows the effect of feedback linearization on the proximity constraints and when calculating the free space. In the differential drive SCARABs, we track the reference point $\mathtt{P_i}$ which is offset from $\mathtt{P_o}$ by a feedback linearization distance, $\mathtt{FB}$. The entire robot lies within a circle centered at $\mathtt{P_i}$ of radius $\mathtt{S} = \mathtt{FB} + \mathtt{r}$, where $\mathtt{r}$ is the original radius of the robot. Since the radius enclosing the robot at the reference point is much larger than the radius of the robot itself, we must increase the size of the obstacles by the new radius $\mathtt{S}$ to prevent collisions, as well as decrease our



(a) (b) (c)

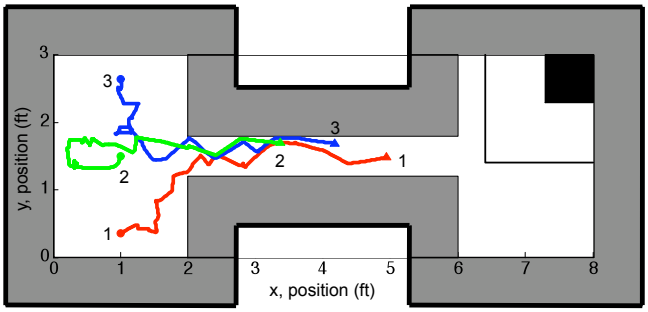
Figure 5.1: The SCARAB robot , the platform for simulations and demonstrations of the decentralized affine feedback controller. (a) A photograph of the SCARAB. (b) Screen capture of three SCARABs mid-simulation in the dynamic simulator GAZEBO. (c) Feedback linearization on the SCARAB.

maximum and increase our minimum proximity constraints. We translate the linear velocity commands which are output by the controller to linear and angular velocity by inverting the equations below (for $\texttt{FB} > 0$):

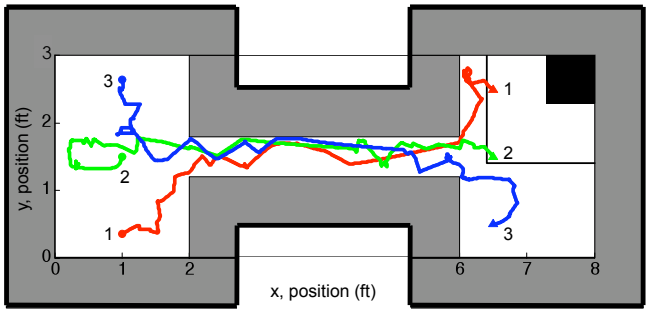
$$\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\texttt{FB}\sin\theta_i \\ \sin\theta_i & \texttt{FB}\cos\theta_i \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}. \tag{5.5}$$

5.2.1 Three agents negotiate passage through a corridor

Figure 5.2a and 5.2b show a simulation in which the agents successfully traverse a narrow corridor which requires the agents to traverse it in a single-file formation, making the proximity constraints difficult to preserve. Note that no formation is specified. Only the collision graph (complete), connectivity and information graphs (both as in Fig. 3.1c) are specified ($\delta_{\min} = 0.7$ft, $\delta_{\max} = 1.6$ft).



(a)

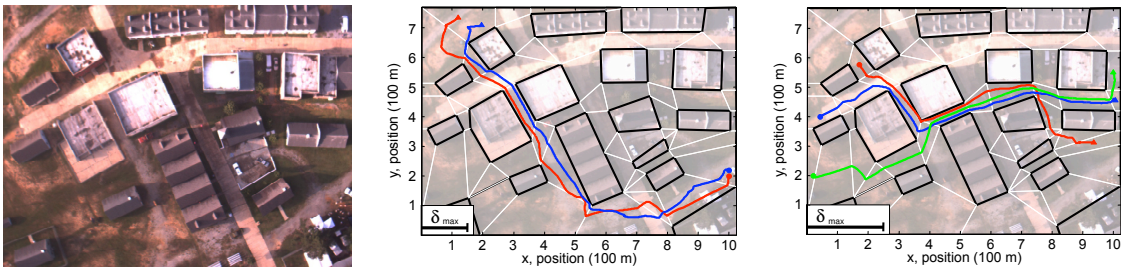


(b)

Figure 5.2: Three agents through a narrow corridor. The black boxes on the upper right show the symmetric proximity constraints, $\boldsymbol{\delta}_{\min} = 0.7$ft, $\boldsymbol{\delta}_{\max} = 1.6$ft. The thick black outline denotes the physical workspace, with the gray areas representing the areas of the configuration space which are within $\boldsymbol{\delta}_{\min}$ of the workspace boundaries. (a) Intermediate state. (b) Final State.

5.2.2 MATLAB simulations through a complex and real space

Figure 5.3 shows a simulation of a real world problem where vehicles with realistic ranges of connectivity (150m in Fig. 5.3b and 250m in Fig. 5.3c) navigate through an urban environment to their respective destinations while keeping within the specified range. Figure 5.3b has two ground vehicles with complete connectivity, collision, and information graphs. Figure 5.3c shows two ground vehicles with one aerial vehicle with a very large connectivity range (greater than 1000m).



(a) (b) (c)

Figure 5.3: A team of unmanned vehicles navigates an urban environment. (a) An aerial view of the urban environment used in the simulation. (b) Results with a team of 2 ground agents. (c) Results with two ground agents and one UAV. In (b),(c) buildings are enclosed by black polygons. The bar on the bottom left shows $\boldsymbol{\delta}_{\max}$ for each simulation. In 5.3c, one agent is a UAV, which has a long range of connectivity.

5.2.3 Three agents in MATLAB and GAZEBO

In a space which is multiply connected, agents have more than one route to the goal; depending on initial conditions and constraints, the agents will choose different paths. A simple multiply connected space is shown in Fig. 5.4a and 5.4c. The thick black outline and black box in the center denotes the physical workspace, with the gray areas representing the areas of the configuration space which are within $\boldsymbol{\delta}_{\min}$ of the workspace boundaries. The black boxes in the corners denote proximity constraints $\boldsymbol{\delta}_{\min} = 0.7\text{m}$ and $\boldsymbol{\delta}_{\max} = 2.5\text{m}$. In these simulations, the agents share the same configuration space, collision graph (complete), start and goal configurations, and proximity constraints. In the centralized case (Fig. 5.4a and 5.4b), the connectivity and information graphs are complete; in the decentralized case (Figures 5.4c and 5.4d) they are not complete (both as in Figure 3.1c, with agent 1 red, agent 2 green, and agent 3 blue). In Fig. 5.4a and 5.4c, solid lines show the position of the robot and dotted lines show the position of the feedback linearization point. The distances between the solid lines and the black dotted lines at the starting point are due to

feedback linearization distances (centralized: 0.3m, decentralized: 0.4m). The agents take different routes to the goal since the feedback linearization points for the two cases are initially in different polytopes.

Figures 5.4b and 5.4d show distance between agents in each case: red depicts inter-robot position, green depicts distance between feedback linearization points, and dotted blue marks the maximum allowable distance. The first row corresponds to agents 1 and 2, the second to agents 2 and 3, and the third to agents 1 and 3. The left column shows Euclidean distance between pairs of agents; center and right columns show distance in the x-direction and y-direction, respectively. The plots show that proximity constraints are maintained. In the center plot in Fig. 5.4d, the red line depicting inter-robot distance dips below the dashed blue line, indicating that the inter-robots distance has exceeded the limit; the distance between the feedback linearization points, however, does not. Since sufficient security margin was built into the inter-robot distances, the robots do not collide. This shows the importance of building in sufficient security margin of at least the robot radius plus the feedback linearization distance, as shown in Fig. 5.1c.

5.2.4 Experiments with two agents

For experiments, we have used two Scarab robots in the environment pictured in Fig. 5.5 and 5.6. Overhead cameras, accurate within 3cm, were used for robot tracking. Obstacles were padded (Fig. 5.5a) according to the feedback linearization distance of 18cm.

Figures 5.5a and 5.5b show the results of one experiment. The thick black outline of the workspace and the thick black lines within the workspace denote the physical workspace, with the gray areas representing the areas of the configuration space

Table 5.1: Critical values in our simulations

	$P_{\max}$	$P = \lvert\mathcal{C}_T\rvert$	$\sum_{i=1}^{P} V_i/P$	$\sum_{i=1}^{P} F_i/P$
Fig. 5.3b	$12,996$	2572	18	9
Fig. 5.4a	4096	464	92	15

which are within $\boldsymbol{\delta}_{\min} = 0.5$m of the workspace boundaries and obstacles ($\boldsymbol{\delta}_{\max} =$ 3m). Figure 5.5b shows distance between the two robots and the distance between the feedback linearization points ($\boldsymbol{\delta}_{\min} = 0.5$m, $\boldsymbol{\delta}_{\max} = 3$m). Figure 5.6 shows sequential still frames of the experiment. This experiment demonstrates that the controller can successfully be applied to real robot navigation problems.

5.3 Computational complexity

Now we discuss the computational complexity of our method. The worst case complexity is mostly determined by the number of polytopes in $\mathcal{C}_T$, which scales exponentially with n, as discussed in Section 3.3. However, actual numbers observed are much lower.

Table 5.1 presents critical values from our simulations. Although $P_{\max}$ is exponential, the actual number of polytopes in $\mathcal{C}_T$ is much lower. The table also presents average number of vertices and facets per polytope.

There are several ways to reduce complexity in the controller synthesis phase.

5.3.1 Distant pairs of polytopes

Given a connectivity graph, it is possible to rule out some combinations of polytopes before taking the Cartesian product. An example was shown in Fig. 3.8. This does not decrease the number of polytopes in $\mathcal{C}_T$, however, it decreases the computation time it takes to generate $\mathcal{C}_T$.

5.3.2 Lazy evaluation

If an initial configuration is given, and the agents cannot initiate in another polytope, find a controller only in the polytopes which are on the path from the initial to final polytope. Although this will not reduce the number of calculations to find $\mathcal{C}_T$, it will *significantly* decrease the number of polytopes for which controllers must be found. More generally, given a limited number of polytopes which contain all possible initial configurations of the agents, solve only in those polytopes which they will pass through.

5.4 Remarks

We presented a method for synthesizing decentralized affine feedback controllers on polytopes. The feedback controllers obtained by this method provide guarantees of convergence to a goal in any known polygonal environment. When combined with the configuration space model constructed in Chapter 3, it guarantees communication maintenance, collision avoidance, and compliance with other set constraints.

We demonstrated the method on groups of multiple heterogeneous agents navigating a known environment with obstacles, including navigating a narrow corridor as well as an urban environment. In experiments as well as 3D dynamic GAZEBO simulations, the controllers, although not specifically designed for nonholonomic robots, successfully drive agents with limited system state information to goal sets while avoiding collisions and maintaining specified proximity constraints. Additionally, we have shown in experiments that the controllers can be successfully applied to real robot navigation problems.

One limitation of this algorithm is that the complexity is exponential in the number of agents. However, we can reduce the complexity of controller synthesis in

two ways: (1) constructing the task configuration space by considering the distance between polygons in the free space (Section 5.3.1); (2) by using lazy evaluation for controller synthesis (Section 5.3.2). Furthermore, by using an informed graph search method in configuration space construction along with lazy evaluation in controller synthesis, we can significantly reduce computation.

Although our algorithm does not require exact knowledge of the position of non-neighbor robots, all robots must have knowledge of the simplex the state is currently in, to determine which controller to apply. This may seem infeasible, however, some limited information about position can be passed through neighbors in the connectivity graph. This would require state information to flow across the robot network, but not at the bandwidth required for a complete information graph. Since exact position need not be known, latency in the network will likely not result in a safety violation.

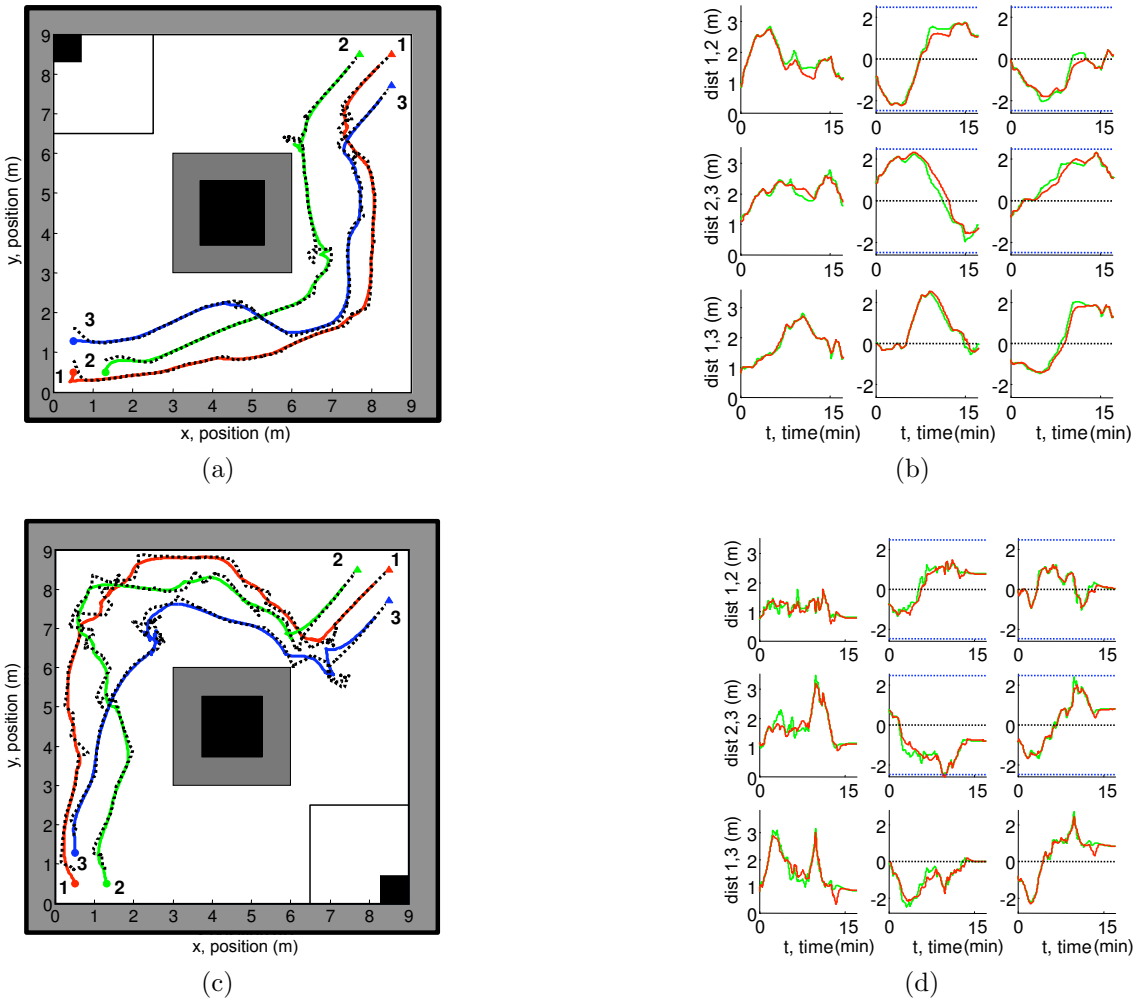
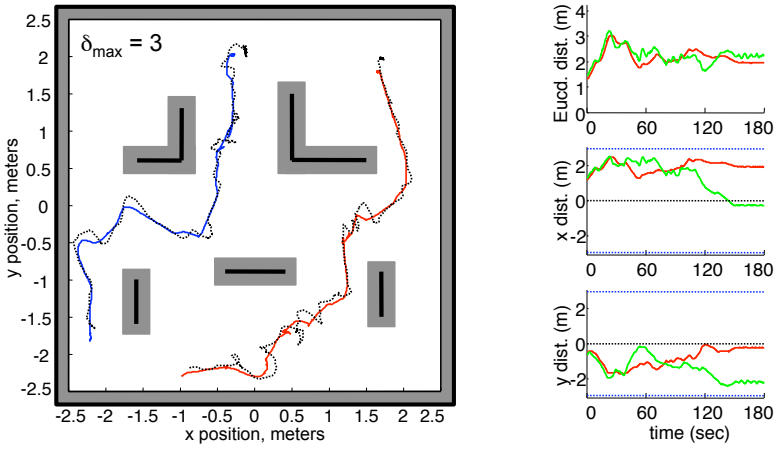


Figure 5.4: GAZEBO simulations in a multiply connected space with identical proximity constraints (agent pair (1,3) not connected) with centralized and decentralized controllers. (a) Results of centralized simulation. (b) Distances between agents in centralized simulation. (c) Results of decentralized simulation. (d) Distances between agents in decentralized simulation.



(a) (b)

Figure 5.5: An experiment with 2 SCARAB robots in a multiply connected space. (5.5a) Experimental results. Solid lines show the position of the robot and dotted lines show position of the feedback linearization point. (5.5b) Distances between robots. Red depicts robot position, green depicts the feedback linearization point, dotted blue marks the maximum allowable distance.

(a) (b) (c)

Figure 5.6: Sequential still frames of the experiment on two SCARAB robots. (a) Start configuration. (b) Intermediate configuration. (c) Goal configuration.

Chapter 6

Feedback Control for Dynamical Systems

Now that we've solved Problems 3.1.7 and 3.1.8 by synthesizing feedback controllers for limited-information kinematic systems, we switch to dynamical systems and Problem 3.1.9. In this chapter, we synthesize controllers to drive a second-order system to the desired coordinates by translating the discrete path on the polytope graph into state trajectories using local cell-based controllers.

Without any loss of generality, let the polytopes be numbered such that the global path planned by the discrete planner is $\{P^0, P^1, \cdots, P^g\}$ and P^g is the goal polytope. For $m \in \{0, 1, \ldots, g-1\}$, P^m is in the discrete plan and it has a successor polytope in the global plan, P^{m+1}, such that P^m and P^{m+1} share a common facet. Hence the feedback controller in P^m should prepare the system for the controller in P^{m+1} so that the robot is driven successively to the goal cell. In the goal cell, P^g, the controller should drive the system to the goal position. Below we develop smooth controllers for both of these situations that are based on navigation functions on star-worlds [102].

6.1 Local polytope controllers

Definition 6.1.1. [24, 54, 102] *Let $\mathcal{Q}$ be a D-dimensional compact, simply connected manifold with boundary and let $\mathbf{q}_g \in \mathcal{Q}$ be a unique point. A function $\varphi : \mathcal{Q} \mapsto [0, 1]$ is a navigation function if it*

- *is twice differentiable on $\mathcal{Q}$;*
- *achieves a unique minimum of 0 at $\mathbf{q}_g \in \mathcal{Q}$;*
- *is uniformly maximal on the boundary, i.e. evaluates to 1 on the boundary; and*
- *is Morse.*

Navigation functions defined above have been shown, in [102], to be useful in kinematic systems ($\dot{\mathbf{x}} = \mathbf{u}$) for generating controllers to drive the system to the goal position $\mathbf{q}_g$ by choosing the control law

$$\mathbf{u} = -\nabla\varphi(\mathbf{x})$$

while staying inside the manifold $\mathcal{Q}$ at all times. Convergence of this control law can be shown by taking $V(q) = \varphi(q)$ as a Lyapunov function.

Let us first consider the case of kinematic agents, where the equations of motion of each robot is given by (3.1).

6.1.1 Controllers for kinematic robots

Let the system currently lie in the polytope P^1 and let the next polytope in the discrete plan be P^2. The whole group dynamics, given by concatenating the equations

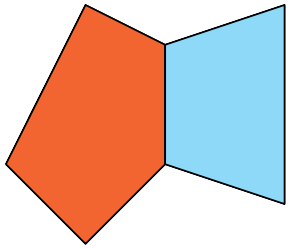


Figure 6.1: Adjacent polytopes P^1 (orange, left) and P^2 (right, blue), which form star S^{12}.

of motion of individual agents, is

$$\dot{\mathbf{x}} = \mathbf{u},\ \mathbf{x} \in \mathcal{C}_T \subset \mathbb{R}^D. \tag{6.1}$$

Corollary 6.1.2. *There exists a navigation function based controller that drives the system in* (6.1) *from a polytope* P^1 *to its adjoining polytope* P^2*.*

Proof. The two adjoining polytopes, P^1 and P^2 share a common facet. The union of these two polytopes is a star-shaped object[1] whose center can be any point on the interior of the common facet. Let this star be denoted by S^{12}. For example, in two dimensions, if in Figure 6.1 the orange-colored polygon on the left-hand side is P^1 and blue-colored polygon is P^2, then the union is of the these is the star, S^{12}.

Now, pick as local goal position, $\mathbf{x}^l \in S^{12}$ that lies in the polytope P^2. Following the proof by Rimon and Koditschek [102], there exists a navigation function, φ, with $\mathbf{x}^l$ as the unique minimal point. As in navigation function literature, the controller

$$\mathbf{u} = -\nabla\varphi(\mathbf{x}) \tag{6.2}$$

will drive the system given by (6.1) to the local goal position $\mathbf{x}^l$ which lies in P^2.

[1]Star shaped sets are closed sets that consists of a "center point" from where any ray crosses the boundary of the set once and only once. They are topologically equivalent to spheres of the same dimension. Common examples of such sets include convex polygons, star polygons, and spheres.

This means that the system has to cross the common facet between polytopes P^1 and P^2 implying that this controller will asymptotically drive the system from P^1 to the adjoining polytope P^2. □

6.1.2 Controllers for robots with second order dynamics

Now let us consider the case of dynamic agents, where the equation of motion for each robot is given by (3.2). As before, let the system currently lie in polytope P^1 with P^2 as the next polytope in the discrete plan. The whole group dynamics is given by concatenating the equations of motion of individual agents and is given by

$$\ddot{\mathbf{x}} = \tau,\ \mathbf{x} \in \mathcal{C}_T \subset \mathbb{R}^D. \tag{6.3}$$

Given the construction of the star S^{12} and a choice of local goal position $\mathbf{x}^l \in P^2 \subset S^{12}$ as in the kinematic agents' case, the same navigation function is also valid for the second order case. The control law

$$\tau = -\nabla\varphi(\mathbf{x}) - \Gamma\dot{\mathbf{x}}, \tag{6.4}$$

where $\Gamma \in \mathbb{R}^{D\times D}$ is a symmetric positive definite matrix, drives the system to the goal $\mathbf{x}^l$ while remaining inside S^{12} at all times. Convergence of this controller can be shown by using $V(\mathbf{x}, \dot{\mathbf{x}}) = \varphi(\mathbf{x}) + \frac{1}{2}\dot{\mathbf{x}}^{\mathrm{T}}\dot{\mathbf{x}}$ as a Lyapunov function.

The above controller drives the second order system to the goal location, but the velocity profile will be very different than that obtained in the corresponding first order system, $\dot{\mathbf{x}} = -\nabla\varphi(\mathbf{x})$. If a similar velocity profile as the first order case is

desired for the second order system, the following controller can be used,

$$\tau = -\Gamma\left(\dot{\mathbf{x}} + \nabla\varphi(\mathbf{x})\right) - \nabla\varphi(\mathbf{x}) - \frac{\partial^2\varphi(\mathbf{x})}{\partial\mathbf{x}^2}\,\dot{\mathbf{x}}, \tag{6.5}$$

where $\Gamma \in \mathbb{R}^{D\times D}$ is a symmetric positive definite matrix. Convergence of this controller can be shown by using $V(\mathbf{x},\dot{\mathbf{x}}) = \varphi(\mathbf{x}) + \frac{1}{2}\left(\dot{\mathbf{x}} + \nabla\varphi(\mathbf{x})\right)^{\mathrm{T}}\left(\dot{\mathbf{x}} + \nabla\varphi(\mathbf{x})\right)$ as a Lyapunov function [55]. Such a requirement is desired in highly dynamic systems such as quadrotors. Since the quadrotor is a 12-dimensional system, we use a second-order approximation instead of taking multiple products of 12-dimensional state spaces. Using the graph embedding (6.5) on top of this approximation allows the quadrotor to "act" more like a first order system, providing better response. We demonstrate the application of this embedding in Section 6.3.5.

6.1.3 Goal-polytope controller

Given the polygonal workspace, polygonal obstacles and proximity constraints described above, the controllers of Sections 6.1.1 and 6.1.2 can be used to drive the robots to the polytope containing the goal state. Once the goal polytope is reached, the controllers can be used with the star being the goal polytope itself and with the local goal as the desired goal configuration of the entire group.

6.2 Construction of navigation functions

The efficient construction of navigation functions in large dimensions is an important aspect of using the controllers of Section 6.1. Here we propose one method which we have succesfully used in our implementations on a variety of simulations described in Section 6.3. Note that other methods such as those described in [24, 103] can be

used for navigation function construction.

To begin, consider a convex polytope with r facets, with a goal location $\mathbf{x}^l$ inside it. Let each of the facets be given by $c_i \equiv k_i - \mathbf{h}_i\mathbf{x} = 0$ for $i = 1, 2, \cdots, r$ where $\mathbf{h}_i\mathbf{x} \leq k_i$ is the i^{th} row of the halfspace representation of the polytope, so that $c_i > 0$ inside the polytope. The boundary of the polytope is given by $c_1 c_2 \cdots c_r = 0$. Note that for small $\epsilon > 0$, the set given by $\beta = c_1 c_2 \cdots c_r - \epsilon > 0$ is always contained inside the polytope and approximates the polytope increasingle better as $\epsilon > 0$ becomes smaller and smaller. This allows us to choose the navigation function

$$\varphi = \frac{||\mathbf{x} - \mathbf{x}^l||^2}{(||\mathbf{x} - \mathbf{x}^l||^{2\mu} + \beta)^{\frac{1}{\mu}}} \tag{6.6}$$

for some $\mu > 0,\ \mu \in \mathbb{R}$, to drive the system to $\mathbf{x}^l$. This construction enables the navigation of the system to reach a desired goal location in the convex polytope while staying inside the convex polytope.

For the purpose of local controllers in Section 6.1, we need to construct navigation functions on the union of two adjoining convex polytopes, which is a star and need not be convex. However, since the polytopes have a matching facet at the interface between them, we can construct a convex polytope extension whose intersection with one of the polytopes is exactly that polytope. Consider adjacent polytopes P^m and P^{m+1} that share a matching facet. In other words, there exists a hyperplane that supports both polytopes, and at the interface, P^m and P^{m+1} share the exact same vertices. Let the halfspace representation of P^m and P^{m+1} be $H^m\mathbf{x} \leq K^m$ and $H^{m+1}\mathbf{x} \leq K^{m+1}$, respectively. Without loss of generality, assume the shared hyperplane is the first row in both H^m, K^m and H^{m+1}, K^{m+1}, so that $\mathbf{h}_1^m\mathbf{x} \leq k_1^m$ and $\mathbf{h}_1^{m+1}\mathbf{x} \leq k_1^{m+1}$. Then $\mathbf{h}_1^m = -\mathbf{h}_1^{m+1}$ and $k_1^m = -k_1^{m+1}$. Let H^m, K^m contain r^m

rows, and H^{m+1}, K^{m+1} contain s^{m+1} rows. Construct the transitional polytope

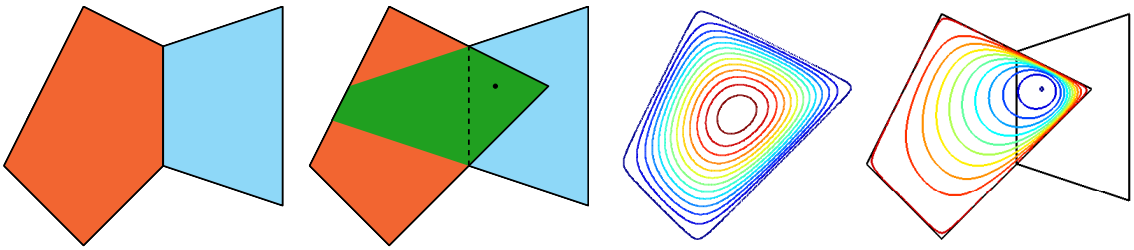
$$P^m_{m+1} = \{\mathbf{x} \quad | \quad \mathbf{h}^m_r \mathbf{x} \leq k^m_r, r = 2, \ldots, r^m, \\ \mathbf{h}^{m+1}_s \mathbf{x} \leq k^{m+1}_s, s = 2, \ldots, s^{m+1}\}.$$

Note that P^m_{m+1} is a convex polytope such that $P^m_{m+1} \subset (P^m \cup P^{m+1})$ and $(P^{m+1} \cap P^m_{m+1}) \setminus \partial(P^{m+1} \cap P^m_{m+1}) \neq \emptyset$. Now construct the polytope P^m_E, which is a convex extension of polytope P^m into polytope P^{m+1}, by

$$P^m_E \equiv P^m \cup P^m_{m+1}. \tag{6.7}$$

We synthesize a navigation function on P^m_E, and by placing the local goal at $\mathbf{x}^l$ in the interior of $P^{m+1} \cap P^m_{m+1}$, we drive the system to the next polytope (the centroid or the center of the Chebyshev ball of the polytope $P^{m+1} \cap P^m_{m+1}$ are good choices). It is important to ensure that the local goal $\mathbf{x}^l$ is within the boundary of the approximation of the next polytope.

An example of constructing the extended polytope P^m_E, as well as the navigation function is shown in Fig. 6.2. This construction makes the regions of attractions of the individual controllers of the sequential composition overlap; this is an advantage over other non-overlapping sequential composition methods used in multi-robot problems as the velocity of the system will decrease as it approaches the goal. In dynamic systems, this lowers the risk of crossing the interface at high speed and overshooting the next polytope. These benefits are most strongly exhibited in microprocessor systems where control is not truly continuous and a slight lag in control input can result in violation of constraints.



(a) (b) (c) (d)

Figure 6.2: An example of constructing convex extensions of polytopes. (a) Adjacent polytopes P^m (orange, left) and P^{m+1} (right, blue). (b) The transitional polytope P^m_{m+1} in green, and the local goal $\mathbf{x}^l$(black-dot). The convex extension P^m_E of P^m is the union of the orange and green polytopes. (c) A smooth approximation of the boundary of P^m_E. (d) A contour plot of the navigation function (6.6) of P^m_E.

6.2.1 Complete algorithm for multi-robot navigation

Algorithm 6.2.1. (Construction of piecewise smooth controllers for kinematic and dynamic systems)

1. *Find a path on G_P and construct $\mathcal{C}_T$ (Definition 3.1.10) simultaneously using heuristic-based graph search.*
2. *Construct polytopes P^m_E as in* (6.7) *for each polytope on the path except the goal polytope.*
3. *Synthesize navigation functions* (6.6) *in each polytope on the path.*

Theorem 6.2.2. *Algorithm 6.2.1 is complete based on the constraints* (3.3).

Proof. For each pair of adjacent polytopes on the path, an extended polytope can be found, as in (6.7). Since $P^m_{m+1} \cap P^{m+1} \neq \emptyset$, there exists a point in P^{m+1} which can be chosen as the local goal $\mathbf{x}^l$. Since $\mathbf{x}^l \in (P^m_{m+1} - \partial P^m_{m+1})$, $\exists\ \epsilon > 0$ such that $\mathbf{x}^l$ is guaranteed to be within the boundary of the approximation of both polygons. By definition, since there are no obstacles in the polytope P^m_E, a navigation function can be constructed without local minima which drives the state to the next polytope. In the goal polytope, a navigation function can be constructed without local minima which drives the state to the goal configuration $\mathbf{x}^g$. Therefore, if a path exists, we

are guaranteed to find a controller to drive the system to the goal configuration. By Theorem 3.2.6, we are guaranteed to find a path if one exists given the constraints (3.3). Therefore, Algorithm 6.2.1 is complete. □

6.3 Simulation results

Now we present several illustrative examples of the successful application of the controller to groups of unmanned ground vehicles (UGVs), unmanned aerial vehicles (UAVs) and quadrotors. Simulations are conducted in MATLAB, using second order models for all robots. For UGVs and UAVs, we use the model

$$\ddot{\mathbf{x}}_i = \mathbf{u}_i, \ x_i \in \mathbb{R}^{d_i}, \tag{6.8}$$

where $d_i = 2$ for UGVs and $d_i = 3$ for UAVs and quadrotors. Since a quadrotor is a 12-state dynamical system (three each of positions, linear velocities, orientation, and angular velocities) with four inputs (rotor velocities) [84], using the full model would be prohibitively complex for multiple quadrotor navigation. Therefore, the control for quadrotors is designed for (6.8), an abstraction of the full 12-state dynamical model. The abstraction is obtained by linearization of the full 12-state model around the nominal hover state, generating the control input τ of (6.4) or (6.5), as is desired. The input τ is then converted into motor thrusts and moments following [84], where the reader is referred for further details on the model. Because the abstraction is not an exact model of the full system, it is important for the control algorithm to be robust to modeling errors. As can be observed in the simulations in Sections 6.3.3, 6.3.4, and 6.3.5, the controller is robust to these errors due to its cell-based nature as well as the second-order lift using Lyapunov functions.

Proximity constraints for all simulations are $\boldsymbol{\delta}_{\min} = 0.5$m in the x, y plane, while

on the z axis, the minimum clearance above is 0.5m and below is 1m, to compensate for air flow between quadrotors. We run the control loop at 100Hz. All simulations were done on a MacBook Pro, with 2.53 GHz Intel Core 2 Duo processor, 4GB 1067 MHz RAM, running MATLAB R2009b. Polytope computations were done using the MultiParametric Toolbox [61] for MATLAB. To achieve faster convergence to the goal, once inside the next polytope, the controller is switched after ensuring ground robots were sufficiently close to the local goal and quadrotor velocity was below a specified threshold.

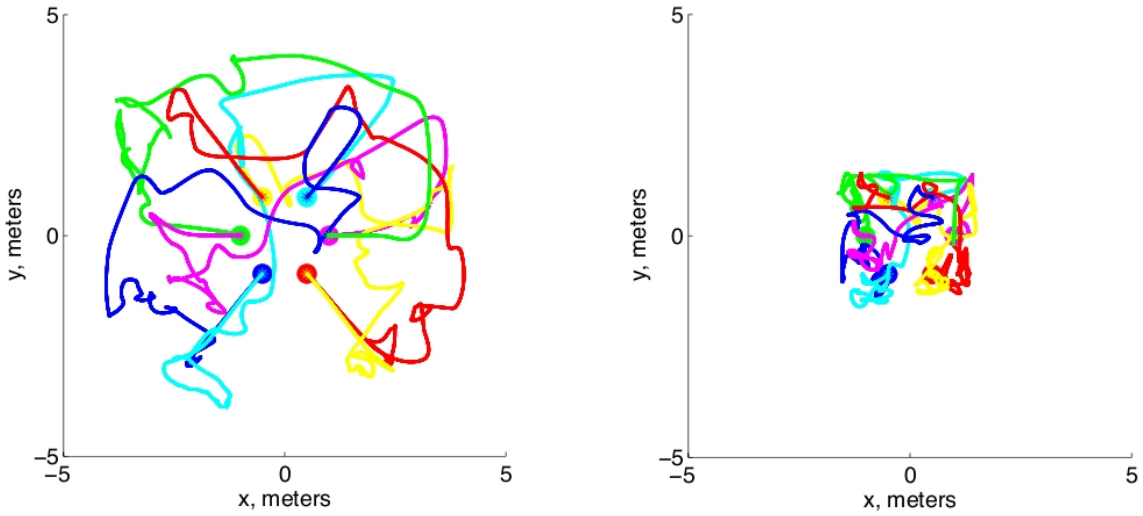
6.3.1 Six robots in a circle

Figure 6.3 shows six ground robots in a circle switch to a position directly across from their current position on the circle while avoiding collisions. The simulation in Fig. 6.3a is not communication constrained, while that in Fig. 6.3b has a maximum communication distance of 3m for each pair of robots. Each color denotes a single robot, which starts at the large dot and ends at the smaller dot. In these simulations, we favor large polytopes in the A* search by considering the transition cost

$$1 + L_{\max}/\rho L_{\min} \tag{6.9}$$

for each polytope, where $L_{\max}$ and $L_{\min}$ are the largest and smallest lengths, respectively, of the smallest hyperrectangle containing the next polytope, and ρ is the radius of the largest ball inscribed in the next polytope. Note that while this cost penalizes small "skinny" polytopes, it does not remove any polytopes from $\mathcal{C}_T$, therefore connectivity is preserved. For the heuristic we use the cost of 1 per transition, which is admissible since each transition has at least a cost of 1, increasing precomputation time (146s and 194s for the simulations without and with communications

constraints, respectively) because of heuristic accuracy. A more accurate heuristic would lead to faster computations.



(a) (b)

Figure 6.3: Six ground robots (2D) in a circle switch to opposite positions. (a) No communication constraints enforced. (b) Maximum distance of 3m communication constraint enforced.

6.3.2 Four UAVs in 3D with obstacles

Figure 6.4 presents the results of an illustrative simulation with four UAVs in an environment with obstacles (depicted in yellow, note that the obstacles meet in the center). Directional changes are due to the fact that we plan a path on the polytope graph, and not on the workspace itself. Although we forfeit some control over the path of the robots, this simplifies the planning process, and allows us to plan more easily for larger groups of robots. Each color represents a UAV, and colors brighten as time increases. Figure 6.4a shows a 3D view of the simulation, and Fig. 6.4b shows a top-down view. Starting positions are depicted by large dots and final positions by smaller dots. Precomputation time for this simulation was 15s.

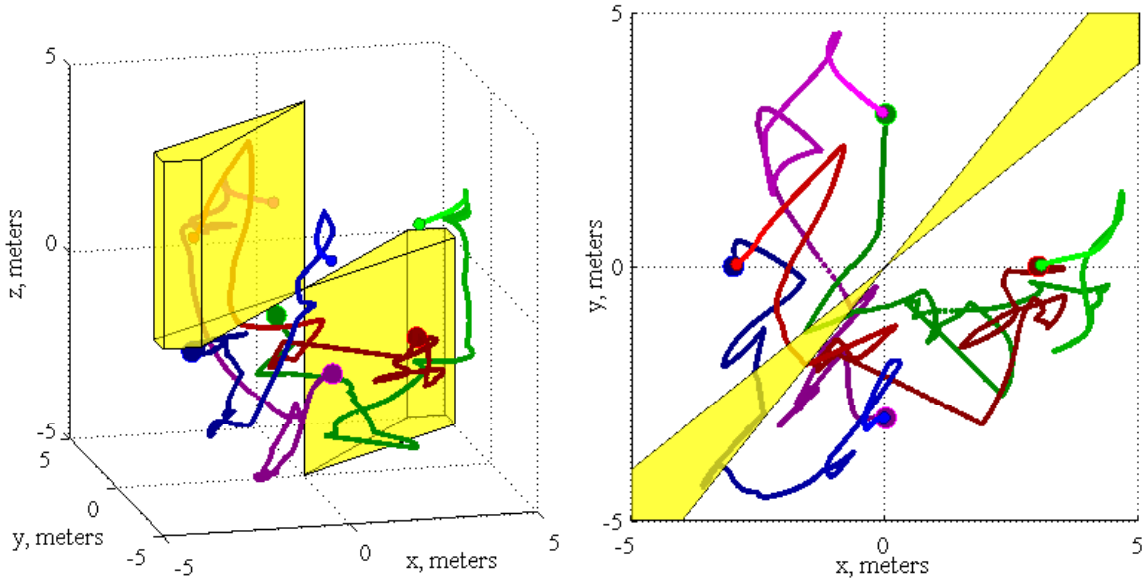


Figure 6.4: Four UAVs in an environment with obstacles (yellow). Each color represents a single UAV, and the colors brighten as time increases. (a) 3D view of the simulation. (b) Top down view of the simulation.

6.3.3 Single quadrotor among buildings

In Fig. 6.5, a single quadrotor navigates through an obstacle filled environment. The quadrotor takes off from the top of the yellow building and hovers at the intersection by the blue building. The top-down view of the quadrotor trajectory is shown for clarity in Fig. 6.5b. Figures 6.5c and 6.5d show the x, y, z position and velocity, respectively, while 6.5e shows the angular coordinates in time. Precomputation time for this simulation was 11.3s.

6.3.4 Two ground robots and one quadrotor

Figure 6.6 shows two examples of the application of our controller to simulations on groups of cooperating heterogeneous robots in an urban environment, with and without communication constraints. Precomputation time was about 11s for both

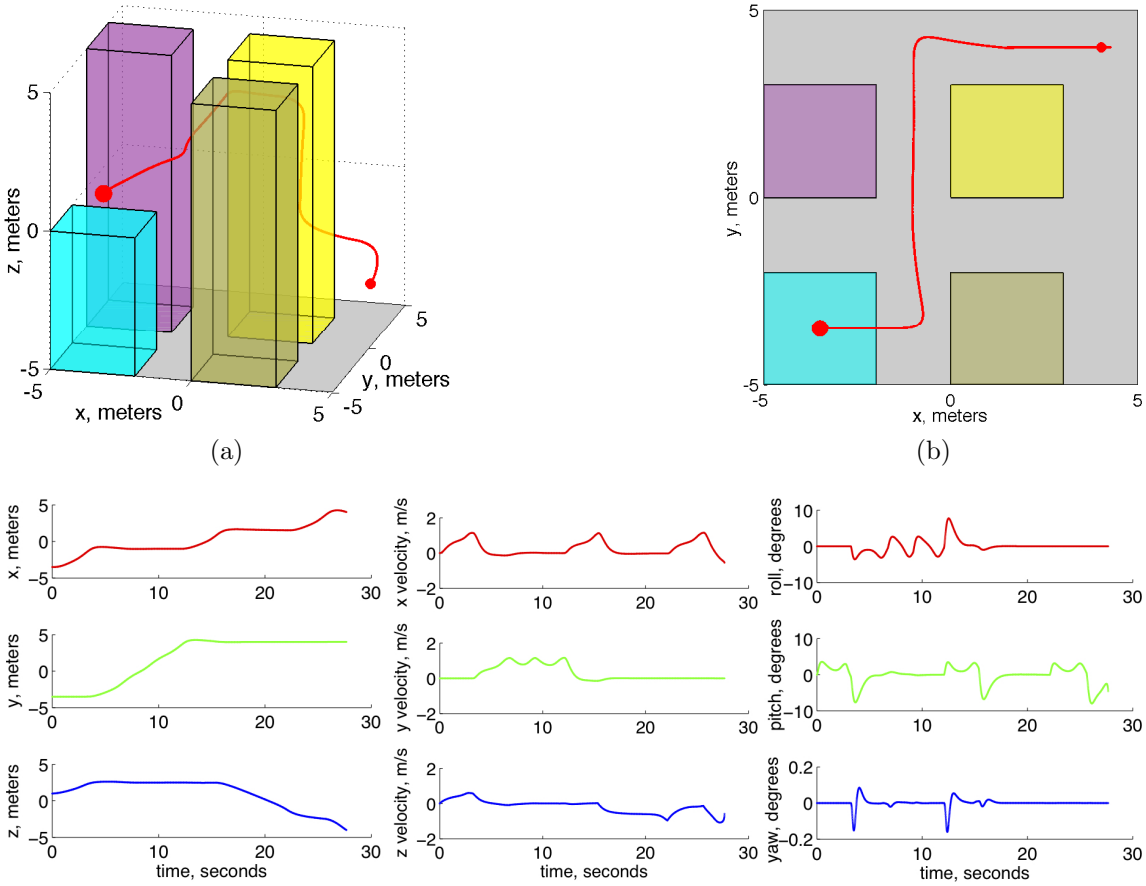
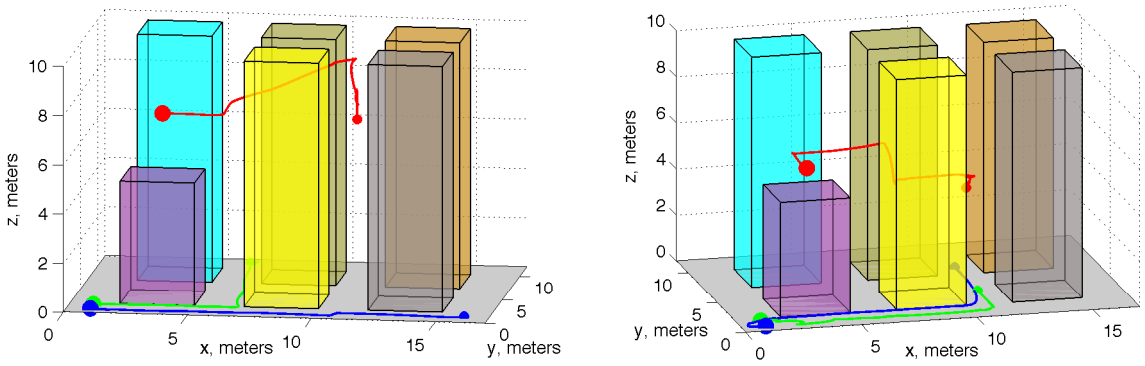


Figure 6.5: A single quadrotor simulation in an obstacle filled environment. The quadrotor takes off from the top of the cyan building, and hovers at the intersection by the yellow building. (b) The x, y, and z position (meters) in time. (c) Velocity in x, y, and z directions, in m/s. (d) Roll, pitch, and yaw in degrees.



(a) (b)

Figure 6.6: A quadrotor (red) and UGV (green, blue) simulation in an urban environment. The robots are deployed from the left front corner in both simulations. (a) The robots are deployed to different intersections without communication constraints. (b) The robots navigate to the same intersection while maintaining a maximum horizontal distance of 6.5m between all pairs of robots. UGVs are in green and blue, and the quadrotor is in red.

simulations. In Fig. 6.6a the robots deploy to different intersections without communication constraints. Figure 6.6b corresponds to a simulation with communication constraints enforced. Robots must maintain a maximum distance of 6.5m in x, y, and z. The trajectory of the 2 UGVs are in green and blue and that of the quadrotor is in red.

6.3.5 Three quadrotors through a window

Figure 6.7 shows the application of our controller in simulation to three cooperating quadrotors that must switch to opposite sides of a window. We enforce collision constraints, however no communication constraints are considered in this simulation. Because quadrotors are highly dynamic and the process model has uncertainties, we again favor large polytopes in the A* search using the cost (6.9), which leads to an increased precomputation time of 50s.

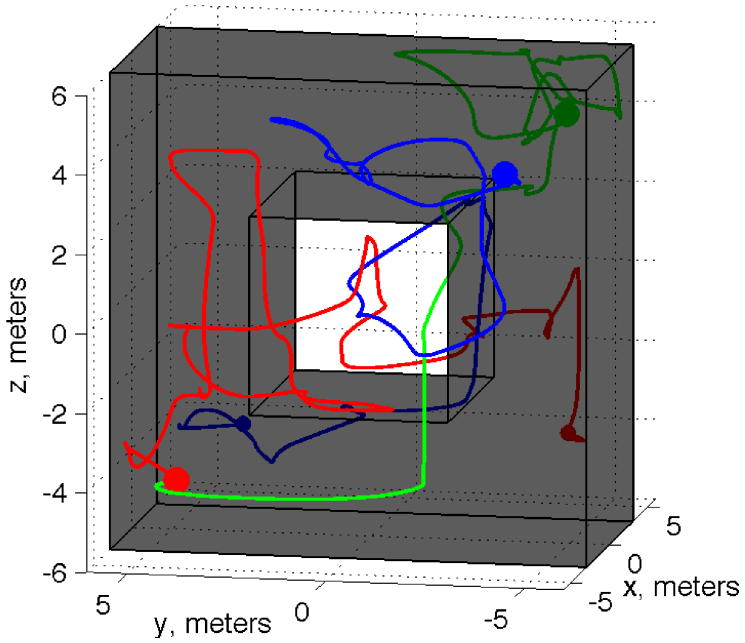


Figure 6.7: Three quadrotors must switch to opposite sides of a window. Each quadrotor is represented by a different color in green, blue and red.

6.4 Complexity

The complexity of the resulting controller depends on the number of robots and the complexity of the free space, as in (3.5). There are $\prod_{i=1}^{n} p_i$ polytopes in $\mathcal{C}_{all}$. In 2 dimensions, the number of proximity regions is $4^{n(n-1)}$, while in 3D, it is $6^{n(n-1)}$. Therefore the maximum number of polytopes in $\mathcal{C}_T$ is $4^{n(n-1)} \prod_{i=1}^{n} p_i$ for 2D systems, and $6^{n(n-1)} \prod_{i=1}^{n} p_i$ for 3D systems. However, we need not explore all of these polytopes. By using the heuristic derived in Section 3.2, we selectively calculate the space, thus significantly reducing the computational burden of the decomposition method.

6.5 Remarks

This chapter presented the development of a novel navigation function-based feedback controller that drives a group of robots with second order dynamics in an obstacle filled environment to a goal location. The controller is the result of the sequential composition of navigation functions in polytopes. Because the polytopes are a tessellation of the task configuration space of the robots, the resulting controller is guaranteed to be free of local minima. Furthermore, the method we have proposed is complete: it is guaranteed to find a solution if one exists.

The controller has been successfully applied in simulation to teams of second order fully actuated robots in both 2D and 3D environments. We have shown simulations on multiple quadrotors in an environment with obstacles, and a simulation on a heterogeneous group of robots, a quadrotor and two UGVs, with and without communication constraints. Since the controller is free of local minima, it does not require tuning for continuous systems, however, tuning improves performance in discrete implementations of control loops.

Chapter 7

Intra-group Controllers

The controllers introduced in Chapters 5 and 6 solve Problems 3.1.7, 3.1.8, and 3.1.9. They can also be used to solve Problem 4.3.2, although there is little reason to use the decentralized controller in Chapter 5 on the virtual boundary. In this chapter we introduce controllers for intra-group control within the virtual boundary and an alternate controller for virtual boundary control. We demonstrate these controllers on group navigation using an abstraction. We begin by introducing an alternate polytope-based controller developed by Lindemann and LaValle.

The vector field approach developed by Lindemann and LaValle in [71] results in smooth feedback control on polytopes. The general idea of the controller is to automatically generate a vector field which behaves like $\nabla\varphi$ instead of constructing φ then differentiating. The approach is to assign a vector field to each facet of the polytope which points inward on invariant facets and points outward on the exit facet. On the Generalized Voronoi Diagram (GVD) of the polytope, assign a vector field which points out of the exit facet. An example of such a field is one directed toward a point in the interior of the exit facet. The fields on the facets and on the GVD are then smoothly blended using a bump function.

Unlike the decentralized feedback controller presented in Chapter 5, this controller is smooth across the interfaces. It requires tessellating each polytope into a Generalized Voronoi Diagram (GVD) which is less computationally intensive than triangulation, which is required in the decentralized controller. In contrast, the navigation function approach in Chapter 6 does not require tessellation of the polytopes. In addition, this controller cannot be decentralized to accommodate for missing edges in the information graph (this is also true of the potential-function approach for second-order systems). Although this approach results in navigation-function-like trajectories, it is impossible to use the graph embedding (6.5) for dynamical systems such as quadrotors since it is not analytical.

We choose this controller since it is easy to generate a constant velocity vector field using this approach, which may be desirable in situations such as the group navigation examples we present below.

7.1 Group control using an abstraction

Once the paths on the upper- and lower- adjacency graphs have been determined, i.e. once Problem 4.3.5 (Discrete abstraction path) has been solved, we translate that path into feedback controllers.

Theorem 7.1.1 (Necessary and Sufficient condition)**.** *Problem 4.3.2* (*Control for group abstraction*) *has a solution iff Problem 4.3.5 has a solution.*

Proof: $\mathcal{C}_A^{free}$ contains every allowable configuration $\mathbf{x}_A$ in our polytopic world model. G_U and G_L together contain all the information about the connectivity of $\mathcal{C}_A^{free}$. Thus, if there is a solution to Problem 4.3.2, there must exist a path from the start node in $G_L^{k^0,m^0}$ to the goal node in $G_L^{k^g,m^g}$. Conversely, if there is no path on the

graph between the start node and the goal node, there is no solution to Problem 4.3.2.

Corollary 7.1.2 (Completeness)**.** *Problem 4.3.5, and therefore Problem 4.3.2, has a solution if the start and goal nodes on the polytope graph G_U are connected.*

7.1.1 Inter-robot constraints

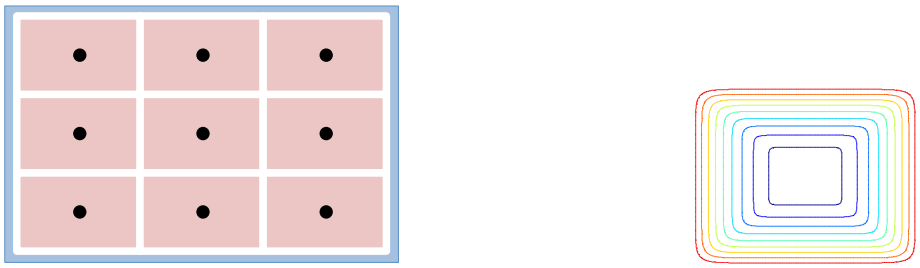
Since the boundary restricts the allowable space for the robots, the robots' configuration space is local and decoupled from the cluttered physical workspace. The input to each robot a_i in the local reference frame is a function

$$\mathbf{u}_i = \mathbf{u}_i \left(\mathbf{x}_1^l, \mathbf{x}_2^l, \ldots, \mathbf{x}_n^l, \mathbf{s} \right),$$

where $\mathbf{x}_i^l$ is the position of a_i in the local reference frame (fixed to and rotating with the center of the group abstraction) and $\mathbf{s}$ is the shape vector.

To solve the local robot navigation problem within the virtual boundary, we can impose interrobot constraints much like those introduced in Chapter 3.1 and use the controllers introduced in Chapters 5 and 6. Alternatively, we can use methods which are reactive to the changing workspace boundaries.

We demonstrate the use of a flexible formation strategy in Sec. 7.2.1. In a flexible formation, we would like the robots to maintain a specific structure or lattice, as in the 3×3 formation in Figure 7.1a. The robots must remain in this formation for the duration of the group navigation. To do so, we use a navigation function within each sub-rectangle which will draw the robots to their assigned location, and keep them from escaping their respective regions by increasing the feedback infinitely as they approach the boundaries. In other words, each robot has a copy of the navigation



(a) (b)

Figure 7.1: A flexible formation within a rectangular virtual boundary. (a) A 3×3 flexible formation. (b) A contour plot of a navigation function for a sub-rectangle of the formation.

function in Figure 7.1b which keeps it in position, and which scales with the size of the virtual boundary.

A Voronoi coverage type controller [22, 97, 112] would be useful if the group was being used for sensing or surveillance of different spaces (demonstrated in Sec. 7.2.2. If maintaining a specific shape is required we can parametrize the controller in [45] to stabilize the robots on an orbit in that shape. For stricter formations, [29, 67] and Chapter 8 provide a more structured organization of robots.

It is critical that the group dynamics evolve on a sufficiently faster time scale than abstraction dynamics to guarantee safety. Summing controllers without sufficient time scale separation could result in local minima.

7.1.2 Setting shape constraints

Choosing shape constraints (4.3) is critical to ensure there is enough space in the abstraction to maintain the desired formation. Knowing the shape of the abstraction, we can determine the minimum size of the abstraction so that it is large enough to contain the number of robots in the group. The minimum size of the abstraction will depend on the number of robots in the group, n, the desired formation, and a

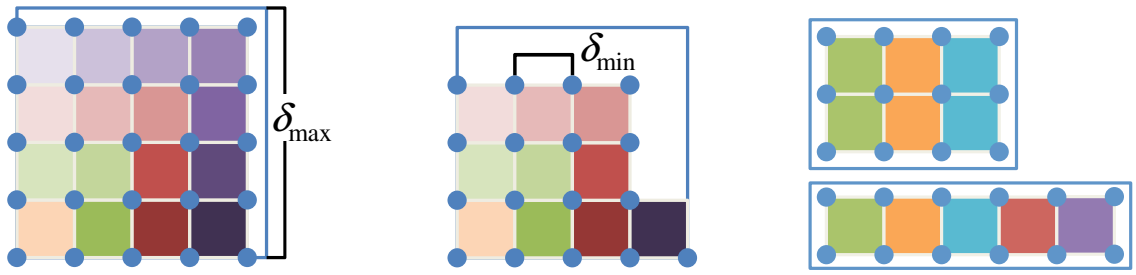


Figure 7.2: Abstraction shape constraints. (a) Maximum number of agents which fit into the abstraction with communication and collision constraints. (b) Minimum size of a rectangle is a function of $\boldsymbol{\delta}_{\min}$ and n. (c) Two formations with the same number of robots require different shape constraints.

minimum distance collision constraint $\boldsymbol{\delta}_{\min}$, if desired. It is indeed possible that two formations with the same number of robots require different shape constraints.

For a rectangular abstraction and using the infinity norm for collision constraints,

$$n \leq \left(\lfloor s_w(n)/\delta_{\min} \rfloor + 1\right)\left(\lfloor s_h(n)/\delta_{\min} \rfloor + 1\right). \tag{7.1}$$

(The delimiters $\lfloor \cdot \rfloor$ denote the floor.) If we would like to additionally ensure graph completeness in the abstraction, we can enforce $\{s_w, s_h\} \leq \boldsymbol{\delta}_{\max}$, where $\boldsymbol{\delta}_{\max}$ is the maximum distance at which communication can occur. An example of the maximum number of robots in a group is shown in Fig. 7.2a. Here, $\boldsymbol{\delta}_{\max}/\boldsymbol{\delta}_{\min} = 4$, so that $n^{\max} = (4+1)^2 = 25$ is the maximum number of robots in the group. An example of the minimum of $s_w(n)$ is shown in Fig. 7.2b. Here, $\lfloor \sqrt{n} \rfloor = 4$, so the minimum width is $4\boldsymbol{\delta}_{\min}$. Although these are general guidelines, choosing the shape constraints relies heavily on the robot formation.

7.2 Simulations

In this section we demonstrate two lower level controllers, a flexible formation and voronoi coverage, in the same workspace. In both simulations, $\Delta\theta = 20^\circ$, the start location is in district D^3 at $A = (\mathbf{x}_A, \theta, \mathbf{s}) = \left([2.5\ 2.5]^{\mathrm{T}}, 0, [3.3\ 3.8]^{\mathrm{T}}\right)$, and the goal location is in district D^7, at $A = (\mathbf{x}_A, \theta, \mathbf{s}) = \left([13\ 4]^{\mathrm{T}}, 0, [2.5\ 4]^{\mathrm{T}}\right)$ (units in meters).

7.2.1 Flexible formation simulation

In the flexible formation each robot is assigned part of a uniform 3×3 grid in the boundary, parametrized by s_w and s_h. To prevent inter-robot collision, we enforce $2.1\mathrm{m} \leq \{s_w, s_h\}$ and $4.2\mathrm{m} \leq s_w + s_h \leq 8\mathrm{m}$, so that the distance between robots would ideally be a minimum of 0.7m. (Maintaining this inter-robot distance is not guaranteed using this robot controller. However, using other controllers, such as those presented in the previous chapters, it is possible to guarantee inter-robot distances.) The navigation functions, coupled with the faster time scale at the robot level, ensure the robots do not escape the boundary. The simulation is shown in Fig. 7.3. Each edge on the upper adjacency graph has a cost of 1; therefore the path with the minimum transitions is chosen.

7.2.2 Voronoi coverage simulation

In this simulation, we use navigation functions to drive robots to the centroid of their Voronoi regions. With this type of controller, the bearing of the robots relative to each other will change as the abstraction boundary changes. The constraints on this simulation are $1\mathrm{m} \leq \{s_w, s_h\}$, $4\mathrm{m} \leq s_w + s_h \leq 8\mathrm{m}$. The Voronoi coverage type control allows us to set a much lower minimum on width and height of the boundary, since the formation of the robots will adapt as the group navigates the space, as shown

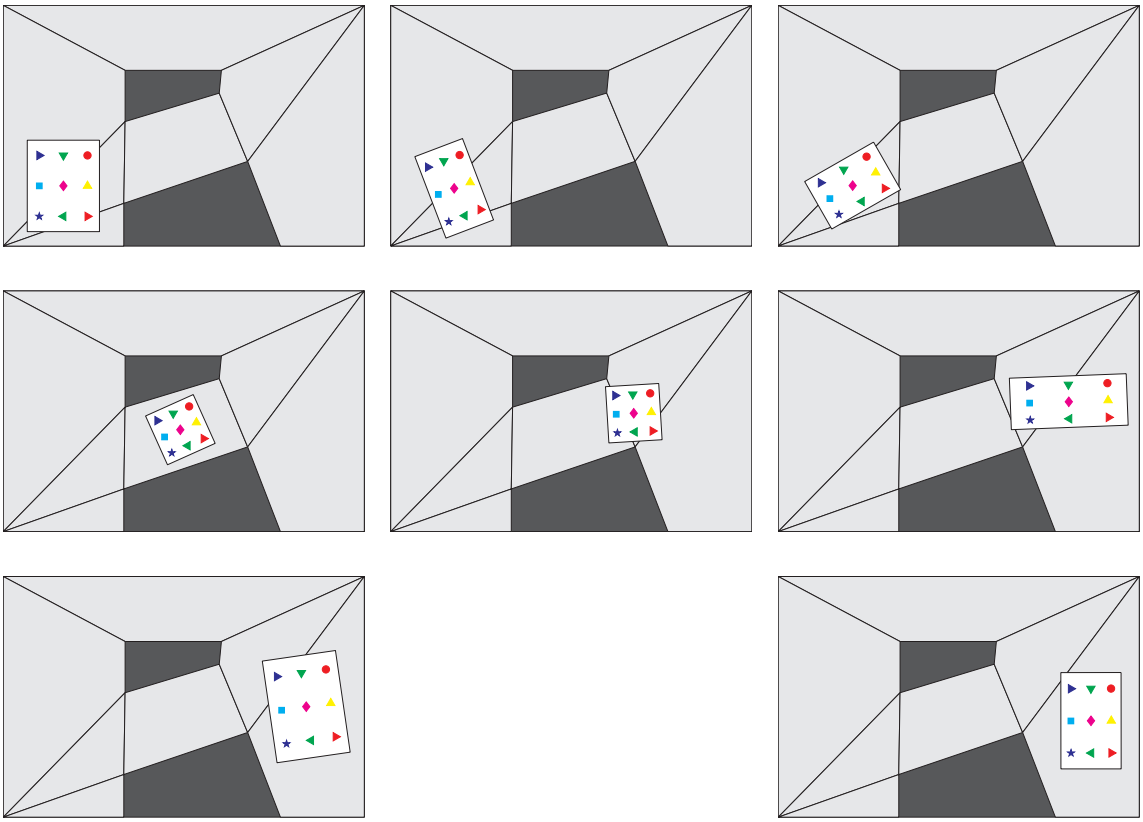


Figure 7.3: A simulation of 9 robots navigating a complex space while maintaining a flexible formation inside the abstraction.

in Fig. 7.4. Here we double the cost of changing theta slices so that the lowest cost path travels through more rooms.

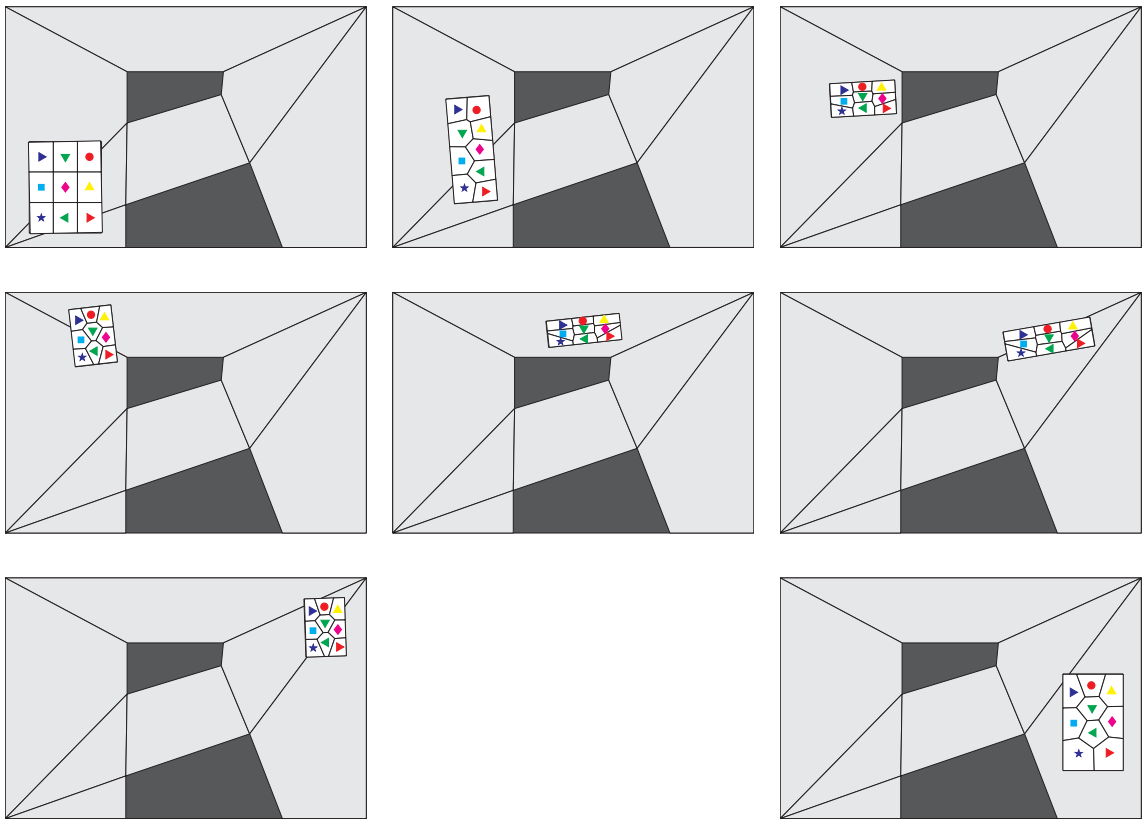


Figure 7.4: Simulation of 9 robots navigating a complex space while using Voronoi region coverage methods inside the abstraction.

Part III

Applications

Chapter 8

Formation Control

In this chapter we apply the polytope-based feedback controllers to a *team* of robots, enabling *automatic* creation of desired labelled formations with constraints on collision and communication and without local minima. The method provides global guarantees on shapes, communication topology, and relative positions of individual robots. We also use formation control to merge and split groups, forming groups of desired size and shape for a specific task.

We use the hierarchical approach introduced in Chapter 4 to decouple the group navigation and internal formation control. The abstraction models the extent of the formation, while the controllers on the robot level ensure that the bounds of the abstraction are satisfied. Therefore, a group of robots can reconfigure from one formation to another knowing only the limits of the abstraction, decoupling the agents from the physical space. Since the multi-group navigation problem is identical to the multi-robot navigation problem, we focus in this chapter on merging groups of robots into groups of arbitrary numbers of robots and constructing desired formation shapes. We use path planning to navigate the groups to their rendezvous point and then to their goal locations.

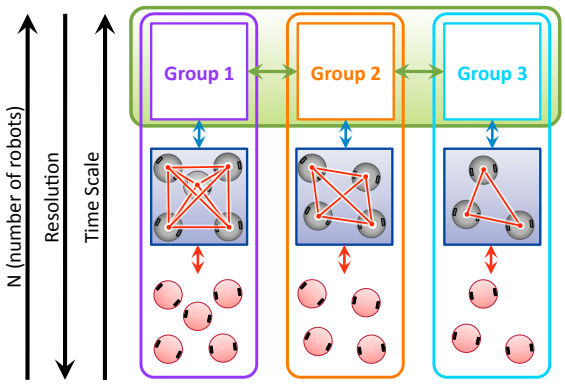


Figure 8.1: Hierarchical structure. At the top level, groups interact with limited knowledge about other groups. At the lowest level, individual robots implement the continuous controllers.

8.1 Problem formulation

Consider a team with multiple groups, $\mathcal{G}^\zeta, \zeta = \{1, \ldots, \xi\}$, of n^ζ kinematic agents $V_A^\zeta = \{a_i^\zeta | i = 1, \cdots, n^\zeta\}$. The team must form a group of $n^g \leq \sum_{\zeta=1}^{\xi} n^\zeta$ agents to accomplish a large task. Each agent has the configuration or state $\mathbf{x}_i^\zeta \in \mathbb{R}^2$ with the dynamics:

$$\dot{\mathbf{x}}_i^\zeta = \mathbf{U}_i^\zeta, \ \mathbf{x}_i^\zeta \in X_i^\zeta \subset \mathbb{R}^2, \ i = 1, \ldots, n^\zeta, \ \zeta = \{1, \ldots, \xi\}. \tag{8.1}$$

The input to each agent is (4.4).

Figure 8.1 is a graphical representation of the hierarchichal structure of our approach. At the top level, groups interact with limited knowledge of other groups. At the middle level, there is interaction between individual robots in order to maintain the formation. At the lowest level, individual robots execute the continuous controller. In the examples we present, we assume that there are no obstacles within the group boundary. However, should an obstacle appear within a group boundary, the group can split appropriately, then rejoin in a location without obstacles.

The number of agents required for the task, the goal formation shape, and the environment are known to all agents. We assume each agent is capable of synthesizing controllers (both for group navigation and reconfiguration), its group's abstraction, and whether certain criteria are satisfied (such merging criteria). This information propagates through the group rapidly through explicit communication, therefore we are not concerned with which agent is responsible for these calculations. Agents observe relative state of their neighbors and exchange this information with other neighbors to construct a complete group configuration. Groups are capable of long-range communication in short bursts to determine a rendezvous point; once the rendezvous point is determined, they no longer use long-range communication.

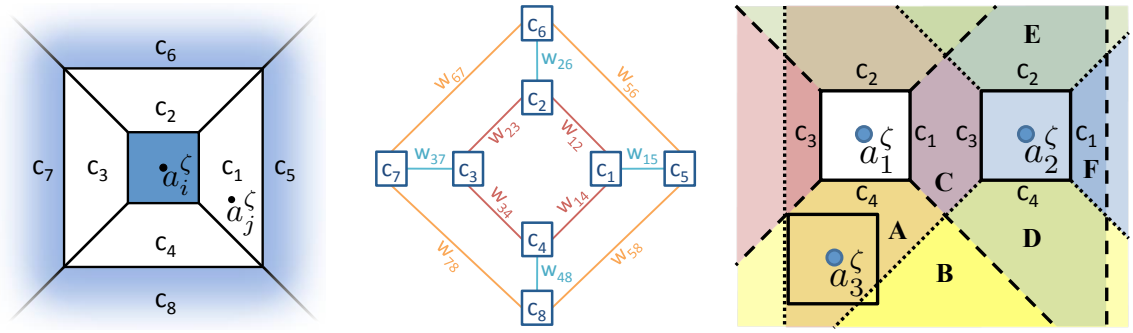
8.2 Formation shape controllers

In this section we develop the formation shape controllers, which both reconfigure the robots and maintain the formation once it is achieved.

Let the set of all agents be $V_A \equiv V_A^1 \cup V_A^2 \cup \cdots \cup V_A^\xi$. (Hereafter, for simplicity, where we describe a property for all agents $\cup_{\zeta=1}^{\xi} \cup_{i=1}^{n^\zeta} a_i^\zeta$, we will drop the superscript ζ.) Connectivity between all agents V_A is modeled by a single connectivity graph across all groups, while the collision graph is defined on individual groups. We assume that within groups, the information graph is complete.

To accomplish the task, a specific formation shape is required of the new group. The formation shape can be provided in continuous (exact) form, or discrete (approximate) form. In defining the discrete robot formation shape, we desire a non-overlapping partition of the square annulus whose union is the entire region. The description must also contain information about connectivity.

Definition 8.2.1. *A* robot formation shape $\mathcal{F}$ *of* n *robots describes the relative*



(a) (b) (c)

Figure 8.2: The shape descriptors (regions) for discrete robot formation shapes. (a) For pair (a_i^ζ, a_j^ζ) the shape descriptor represents the location of a_j^ζ with respect to a_i^ζ. Here, the region is c_1. (b) We can use this discrete system to build an adjacency graph. (c) Overlapping proximity regions of a group of three robots. Dashed (dotted) lines correspond to boundaries of proximity regions for a_1^ζ (a_2^ζ). Letters A-F correspond to possible polytopes through which a_3^ζ would pass to get to $\mathcal{F}_d^\zeta = [\mathsf{c}_1\ \mathsf{c}_1\ \mathsf{c}_1]$.

locations of the set of robots, specified exactly using continuous shape variables. $\mathcal{F}$ *is a* $\binom{n}{2}$*-tuple of vectors* $\mathcal{F} = \left[\mathbf{r}_{(1,2)}\ \mathbf{r}_{(1,3)}\ \ldots\ \mathbf{r}_{(n-1,n)}\right]$ *where* $\mathbf{r}_{(i,j)} = x_j - x_i$*. We use a superscript* ($\mathcal{F}^\zeta$) *to refer to the subset corresponding to group* $\mathcal{G}^\zeta$*.*

Definition 8.2.2. *The* discrete robot formation shape $\mathcal{F}_d$ *of* n *robots describes approximate relative locations of the set of robots using discrete shape descriptors. The shape descriptors are node names corresponding to regions of the tessellation of the annulus.* $\mathcal{F}_d$ *is a* $\binom{n}{2}$*-tuple of these node names,* $\mathcal{F}_d = \left[f_{(1,2)}\ f_{(1,3)}\ \ldots\ f_{(n-1,n)}\right]$*. For each pair of robots* (a_i, a_j)*,* $i < j$*,* $f_{(i,j)}$ *describes the position of* a_j *with respect to* a_i*, according to Fig. 8.2a. Regions* c_1 *to* c_4 *correspond to communication between the pair (i.e.* $(a_i, a_j) \in E_N$*). Regions* c_5 *to* c_8 *correspond to no direct communication between the pair* ($(a_i, a_j) \notin E_N$*). We use a superscript* ($\mathcal{F}_d^\zeta$) *to refer to the subset corresponding to group* $\mathcal{G}^\zeta$*.*

An example discrete formation shape for a group of three robots is shown in

Fig. 8.2c. Here, $f^{\zeta}_{(1,2)} = \mathtt{c}_1$, $f^{\zeta}_{(1,3)} = \mathtt{c}_4$, and $f^{\zeta}_{(2,3)} = \mathtt{c}_3$, so that $\mathcal{F}^{\zeta}_d = [\mathtt{c}_1\ \mathtt{c}_4\ \mathtt{c}_3]$. This is the same as the discrete pose introduced in Section 3.2, except here we do not include the location of each agent in the workspace, since they are all within the obstacle-free virtual boundary.

As discussed in Chapter 4, the free configuration spaces of the robots are simply the area contained within the abstraction boundary in the local abstraction frame. We generate the task configuration space for each group, $\mathcal{C}^{\zeta}_T$, identically to Section 3.1, by taking into account the proximity constraints. Note that each group's task configuration space is independent of other groups; that is, robots in a group rely only on each other's positions. Thus, reconfiguration of a group from one formation to another is entirely decoupled from other groups. Each discrete group formation corresponds to a unique polytope in $\mathcal{C}^{\zeta}_T$. By planning and synthesizing controllers on $\mathcal{C}^{\zeta}_T$, we drive the robots to the desired formation and keep them there as the group navigates the space.

Problem 8.2.3 (Formation Control)**.** *For any initial group formation $\mathcal{F}^{\zeta,0}$, consider the system (8.1) on $\mathbb{R}^{2n^{\zeta}}$, with goal formation $\mathcal{F}^{\zeta,g}$ and metric ρ. Find an input function $\mathbf{u}^{\zeta} : [0, T_{\zeta}] \to \mathcal{U}$ for any $\mathbf{x}^{\zeta,0} \in \mathcal{F}^{\zeta,0} \subset \mathcal{C}^{\zeta}_T$ such that*

1. *for all time $t \in [0, T_{\zeta}]$, $\mathbf{x}^{\zeta} \in \mathcal{C}^{\zeta}_T$ and $\mathcal{F}^{\zeta}|_{t=T_{\zeta}} = \mathcal{F}^{\zeta,g}$,*
2. *$\dot{\mathbf{x}}^{\zeta}_i = \mathbf{u}^{\zeta}_i$,*
3. *$x^{\zeta}(t) \in \mathcal{L}^{\zeta} \cap \mathcal{N}$, $\forall t \in [0, T_{\zeta}]$.*

Problem 8.2.4 (Discrete Formation Path)**.** *For the initial discrete group formation shape $\mathcal{F}^{\zeta,0}_d$, find a path to the goal discrete formation shape $\mathcal{F}^{\zeta,g}_d$, such that we minimize the cost*

$$h\left(\mathcal{F}^0_d, \mathcal{F}^g_d\right) = \sum_{\kappa=1}^{\binom{n}{2}} \mathtt{COST}(\mathtt{c}^0_{\kappa}, \mathtt{c}^g_{\kappa}).$$

Theorem 8.2.5 (Necessary and Sufficient condition)**.** *Problem 8.2.3 has a solution iff Problem 8.2.4 has a solution.*

Proof. $\mathcal{C}_T^\zeta$ contains every allowable configuration $\mathbf{x}^\zeta$ in our polytopic world model. G_P^ζ contains all the information about the connectivity of $\mathcal{C}_T^\zeta$. Thus, if there is a solution to Problem 8.2.3, there must exist a path from the start node in G_P^ζ to the goal node. Conversely, if there is no path on the graph G_P^ζ between the start node and the goal node, there is no solution to Problem 8.2.3. □

Corollary 8.2.6 (Completeness)**.** *Problem 8.2.4, and therefore Problem 8.2.3, has a solution if the start and goal nodes on the polytope graph G_P^ζ are connected.*

8.2.1 Controller synthesis

After a path to the goal is determined, we want to be able to synthesize feedback controllers to drive the system through those polytopes to the goal. We choose to use the controller developed by Lindemann and Lavalle which we discussed in Chapter 7. We set the vector field on each facet except the exit facet to be a unit inward normal; on the exit facet, we set the vector field to the unit normal pointing outward. For each polytope on the path to the goal formation, we implement the controller using the Chebyshev center of the exit facet as the attractor for the vector field on the GVD. In the goal polytope, if an exact formation shape is prescribed, we decompose the polytope so that the vertices of each polytope in the decomposition are the vertices of a facet along with the goal point. Then we set all facet vector fields to be pointing inward, and the field on the faces of the decomposition always pointing to the goal.

If a discrete formation shape is prescribed, we set the attractor field in the goal

polytope to point to the Chebyshev center of the polytope, satisfying the requirements set in [71] to guarantee convergence. There are several advantages to using the Chebyshev center. First, it allows a minimum radius around the attractor, providing some robustness (for disturbances or feedback linearization for nonholonomic robots). Second, the Chebyshev center lies on the GVD, so we do not need to use a separate decomposition that would force us to enumerate the vertices of the polytope, which is computationally expensive. To smoothly stabilize the system at the Chebyshev center, we normalize the attractor field until it is within the radius of the Chebyshev ball. Once within the Chebyshev ball, we use a second bump function to drive the input to zero as we approach the center.

We use the controller for the goal polytope to maintain the desired group formation as the group moves through the space, inside the bounds of the abstraction. In these examples, we do not allow the abstraction to rotate, and once a size has been determined for a group's abstraction, we do not allow the size to change. Other than during the merging process, we let $s^\zeta \equiv s_w^\zeta(n^\zeta) \equiv s_h^\zeta(n^\zeta)$, so that the boundary during group navigation is a square. During merging, we let the boundary be the smallest rectangle enclosing the boundaries of the individual groups. This ensures that there is enough space for all robots while requiring less space than a square.

To emulate the cost of sharing information across large spaces, we place restrictions on communication between groups of robots on three levels based on inter-group distances. The lowest level of communication occurs at the largest distances, above the threshold $\mathtt{t}^{\max}$,

$$|\mathbf{x}_A^\zeta - \mathbf{x}_A^\eta|_\infty > \mathtt{t}^{\max}.$$

At this level, groups communicate as necessary to negotiate rendezvous points.

The mid-level of inter-group communication occurs below the threshold $\mathtt{t}^{\max}$,

$$|\mathbf{x}_A^\zeta - \mathbf{x}_A^\eta|_\infty \leq \mathtt{t}^{\max},$$

where groups perceive position relative to each other ($\mathbf{x}_A^\zeta - \mathbf{x}_A^\eta$).

Once two groups are close enough that explicit inter-group communication is established (i.e. a member from one group is able to communicate directly with a member of the other group), groups can share both the number and position of each group's agents by passing this information through the robot formation graph. Once the group is within a pre-specified distance of the other groups,

$$|\mathbf{x}_A^\zeta - \mathbf{x}_A^\eta|_\infty \leq \mathtt{t}^{\min}, \;\; \zeta, \eta \in \{1, \cdots, \xi\}, \; \zeta \neq \eta \tag{8.2}$$

the groups are able to commence the merging process.

We treat the abstraction as a single robot. Because multiple abstractions share one workspace, we need a multi-robot controller to ensure they do not collide. As discussed in Chapter 2, there are many controllers applicable to multi-robot problems, including those we have presented in Chapters 5, 6, and 7. However, the inter-group communication restrictions as well as the real-time nature of this problem require a decentralized controller to bring the groups to the rendezvous area. (In our simulations, we have used path-planning to determine a path for each group to the rendezvous point.) Once the groups are close enough to know relative position ($|\mathbf{x}_A^\zeta - \mathbf{x}_A^\eta| \leq \mathtt{t}^{\max}$), a more demanding controller may be used to drive them close enough to communicate and merge (until they satisfy (8.2)). Then, they must reconfigure into the desired formation, and continue to the task location while maintaining that formation.

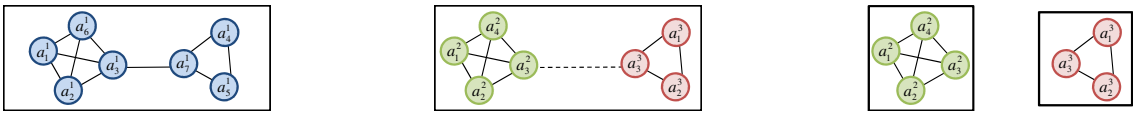
8.3 Merging and splitting groups

In this section we describe the process of merging groups. Once the groups have satisfied (8.2), they combine their boundaries into the smallest boundary of the specified shape that contains all of the groups' boundaries. This will now be the boundary for the reconfiguration discussed in Sec. 8.2.

The desired formation can be either connected or disconnected. If we have just the right number of robots, the resulting graph is connected. If we have more than needed for the task, the formation graph is disconnected, and a group of robots break away.

If we have the exact number of robots required for the task, once they have reconfigured into the desired formation, the boundary size must be adjusted. The boundary is resized to within some small ϵ of the smallest rectangle centered at the centroid of the group and enclosing all the robots. If it is possible to resize directly to the desired size then we are finished, and the group can continue to the task location. If not, we allow the robots to stabilize to the Chebyshev center of the formation using the new boundary size. Then, we re-evaluate the boundary size, and iterate until we get to the desired size. Because we are decreasing the size of the virtual boundary, the formation will get tighter until the group can be enclosed as desired.

If we have more robots than required, the group reconfigures into a formation shape such that the required robots' subgraph is the one required for the task, and the rest of the robots are connected to that subgraph by one edge. An example of this is shown in Fig. 8.3. The dotted line in the example depicts where the group will split. The desired formation is achieved after reconfiguration in Fig. 8.3a. The discrete formation shape node relating a_3^1 to a_7^1 is $f_{3,7}^1 = \mathsf{c}_1$. In Fig. 8.3b, the groups



(a) Desired formation shape (b) Preparing to split (c) After splitting

Figure 8.3: An example of a desired robot formation shape and the splitting process when the task requires less robots than the total number in all groups.

separate, until $f^1_{3,7} = \mathsf{c}_5$, meaning there is no direct communication between a^1_3 and a^1_7. This is when the group splits into two as in Fig. 8.3c.

In summary, the algorithm for our approach involves the following six steps.

Algorithm 8.3.1.

1. *Construct the goal controller for formation maintenance in $\mathcal{C}^\zeta_T$ for each group.*
2. *Drive the groups toward each other in the space while using the formation maintenance controller.*
3. *When (8.2) is satisfied, solve Problem 8.2.4 while selectively constructing the task configuration space for the joint group of robots.*
4. *Solve Problem 8.2.3 on all polytopes on the path, and solve for a goal controller in the goal polytope.*
5. *If $\sum_{i=1}^{\xi} n^\zeta > n^g$, break the team into two separate groups and construct the task configuration space and goal polytope controller for the new groups.*
6. *Drive the newly formed group(s) to the task location while using the new goal polytope controller to maintain the formation.*

8.4 Simulations

We simulate a two-group example where the groups merge into the correct number of robots required for the task, and a three-group example where there are more robots than necessary to complete the task. The simulations run on MATLAB, using the Multi-Parametric Toolbox for polytope computations [61].

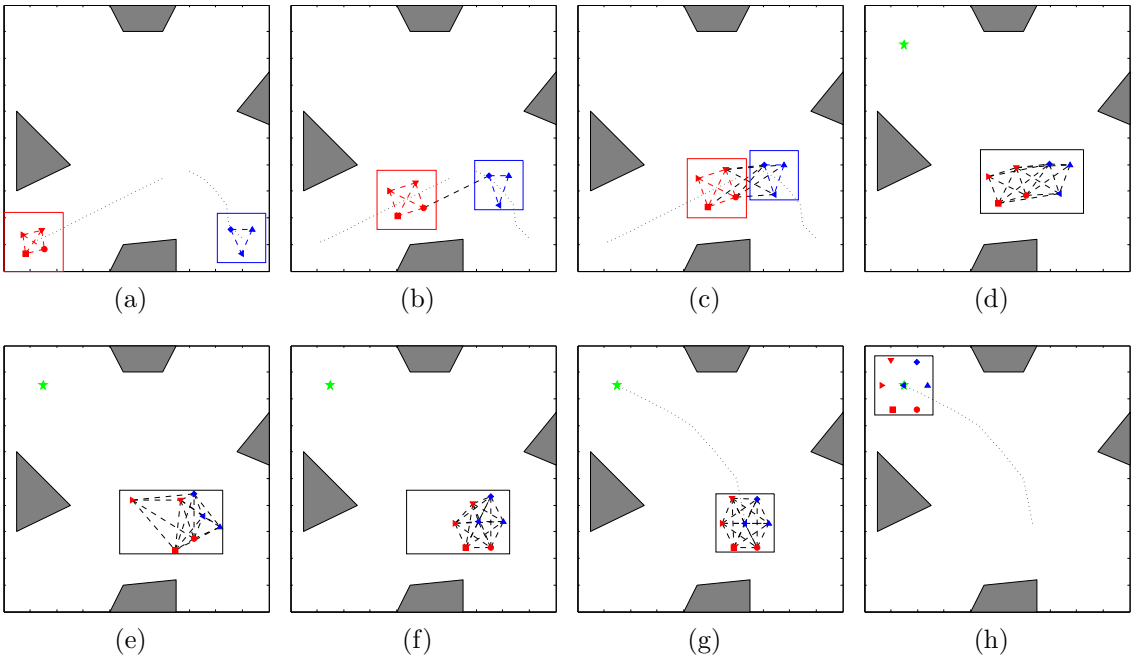


Figure 8.4: A two group simulation. Dashed lines represent communication links (omitted in (h)), dotted lines represent the path, and the star represents the task location. (a) Initial condition. (b) Direct inter-group communication established. (c) Merging criterion satisfied. (d) A single group is formed. (e) Mid-reconfiguration. (f) At the desired formation. (g) The boundary is reshaped. (h) At the task location.

8.4.1 Two groups merging and continuing to task location

In this example (Fig. 8.4) a group of four robots and a group of three robots join to form a single group for a task which requires seven robots. Once the groups are merged and in the desired boundary shape and size, they proceed to the task location. We used as parameters $\boldsymbol{\delta}_{\max} = 2.5$, $\boldsymbol{\delta}_{\min} = 0.2$, $\mathtt{t}^{\max} = 5$, $\mathtt{t}^{\min} = 0.2$, $\epsilon = 0.1$, and $s_h^\zeta = s_w^\zeta = 2.5 - 2/n^\zeta$.

8.4.2 Three groups merging and splitting

In this example (Fig. 8.5), two tasks require seven and two robots each. Nine robots are available across three groups of three. The three groups merge, then split into a

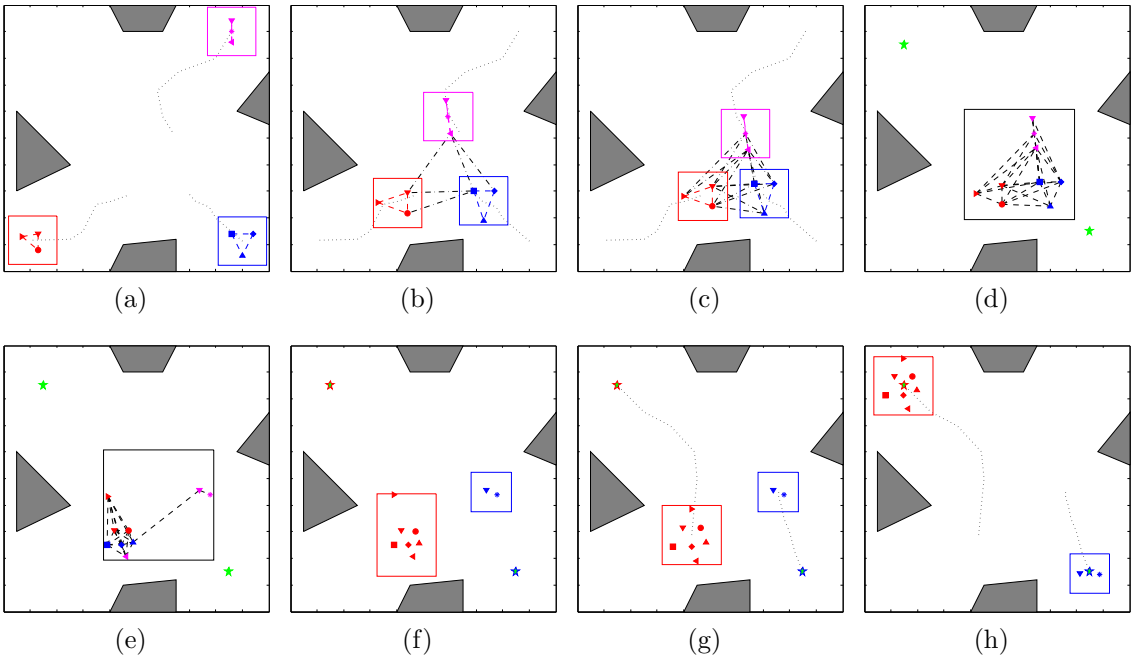


Figure 8.5: A three group simulation. Dashed lines represent communication links (omitted (g)-(j) where graphs are complete), dotted lines represent the path, and stars represent task locations. (a) Initial conditions. (b) Inter-group direct communication established. (c) Merging criterion satisfied. (d) One group is formed. (e) Prior to disconnection. (f) Groups split. (g) Boundaries are resized. (h) At the task locations.

group of seven and a group of two robots. The groups then proceed to their respective task locations. Here we used the same parameters as above (except $\mathtt{t}^{\min} = 0.5$).

8.5 Complexity

In this section we discuss the complexity of our method. For each pair of agents with collision constraints we have one annulus with eight regions, resulting in a maximum of

$$P_{max} = 8^{n^\zeta(n^\zeta-1)/2}$$

polytopes in $\mathcal{C}_T^\zeta$. Although this scales exponentially with the number of robots in the group, we only construct the polytopes as we expand nodes in the polytope graph.

To solve Problem 8.2.4, we use an A^* algorithm. In an A^* algorithm, the number of nodes expanded is exponential in the actual path length, unless the error of the heuristic grows no faster than the logarithm of the actual cost [106]

$$\left| h\left(\mathcal{F}_d^{\zeta,0}, \mathcal{F}_d^{\zeta,g}\right) - h^*\left(\mathcal{F}_d^{\zeta,0}, \mathcal{F}_d^{\zeta,g}\right)\right| \leq O(\log h^*(n^\zeta)).$$

Although we do not have a bound for our heuristic error, empirically we have found that there exists a path to the goal of the heuristic cost. If there exists a path to the goal, then there likely exist other paths of the same length to the goal (though in the case of differently weighted transitions, equal length may not correspond to equal cost). For example, consider Fig. 8.2. If the start formation is $\mathcal{F}_d^{\zeta,0} = [\mathsf{c}_1\ \mathsf{c}_4\ \mathsf{c}_3]$ and the goal formation $\mathcal{F}_d^{\zeta,g} = [\mathsf{c}_1\ \mathsf{c}_1\ \mathsf{c}_1]$, then $h(\mathcal{F}_d^{\zeta,0}, \mathcal{F}_d^{\zeta,g}) = 3$. There exist three paths with cost $h^*(\mathcal{F}_d^{\zeta,0}, \mathcal{F}_d^{\zeta,g}) = 3$: **ABDF**, **ACDF**, and **ACEF** in Fig. 8.2c. In general, since the graph is cyclical, it is likely that a path exists with the exact cost of the heuristic. If the graph is weighted such that each edge is not of equal cost, this is not generally the case.

We synthesize a controller in each polytope on the path to solve Problem 8.2.3. The vector field can be computed in $O(\sum_{d=0}^{2N^i} \Psi_d)$ time, where Ψ_d is the total number of d-dimensional cells in the GVD [71]. For bounds on Ψ_d for a certain class of polytopes, see [53].

The complexity increases linearly in the number of concurrent merging and re-configuring processes, since each group computes its own controllers.

8.5.1 The effects of group size and congestion on complexity

We have already discussed that the complexity of the method is exponential in the number of robots. However, we have not yet explored the effects of congestion on the graph search. One can imagine that a large number of robots in a small space may cause difficulty in finding a solution, since some discrete formation shapes may not be achievable. To analyze this, we ran numerous trials to test the effect of robot group size and the ratio of size of the individual configuration space ($\mathcal{C}_i$) to the size of the robots.

In these trials, the configuration space is square and obstacle free. We calculate the size ratio as the length of the side of the square to the robots' radius (the group of robots here is homogeneous). For robot group size ranging from four to twelve, and size ratio also from four to twelve, we ran trials with randomly generated initial and final configurations. For each pair of group size and size ratio, 30 trials were run and averaged. The most congested of the trials is that with twelve robots with a size ratio of four. Here, 12 robots maneuver in a space that can only accommodate 16 robots when fully packed. The results indicate that congestion in the space has little discernible effect on the time spent searching and constructing the graph.

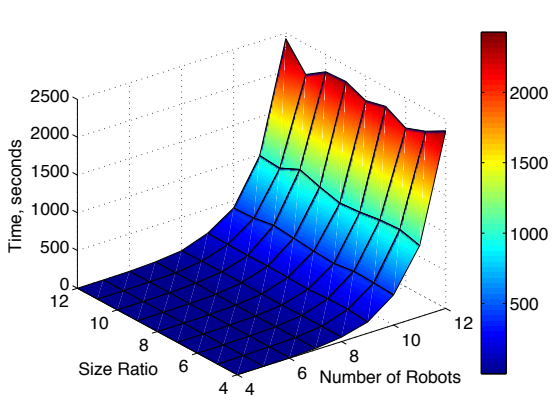
Figure 8.6 presents the results of our trials. Figure 8.6a depicts the average time, in seconds, spent searching and constructing the graph for each pair of size ratio and number of robots. While the time spent increased markedly as the number of robots increased, the effect of congestion was quite small. The average path length determined by the A* search is presented in Figure 8.6b. The increase in path length is not as dramatic as the increase in computation time, although it does increase, as expected, as the number of robots increases. We can also see that the path length does not change much as the congestion increases. In fact, in the most congested

trial of 12 robots with a size ratio of 4, the path length is comparable to the rest of the trials with 12 robots. Finally, Fig. 8.6c compares the number of nodes which are expanded in the graph search (left) with the number of those nodes which are actually valid and hence added to the graph (right). We can see from these plots that results are quite comparable along the axis of size ratio. However, in a congested space, the resulting polytopes will be quite small in comparison to those in a less congested space, so they may be less desirable for approximate models, like our model of the quadrotors.

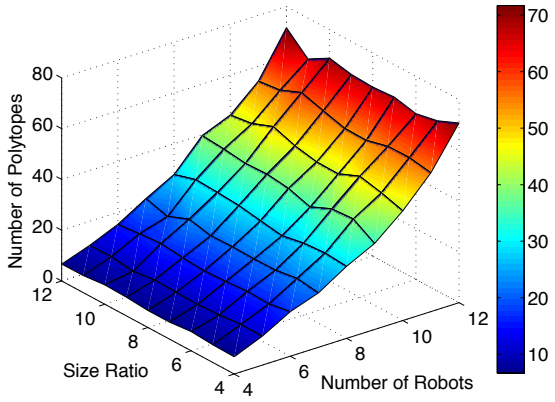
8.6 Remarks

In this chapter, we have presented a method for controlling multiple groups of robots to create, reconfigure, and maintain formations under communication constraints. We provide guarantees of safety, preventing inter-robot collisions and collisions with obstacles in the workspace. Our controller is entirely automatic, and requires information about the space, the desired formation, and the task location. We have discussed briefly the complexity of our approach, which in the worst case scales exponentially in n^{ζ} and $h^*\left(\mathcal{F}_d^{\zeta,0}, \mathcal{F}_d^{\zeta,g}\right)$.

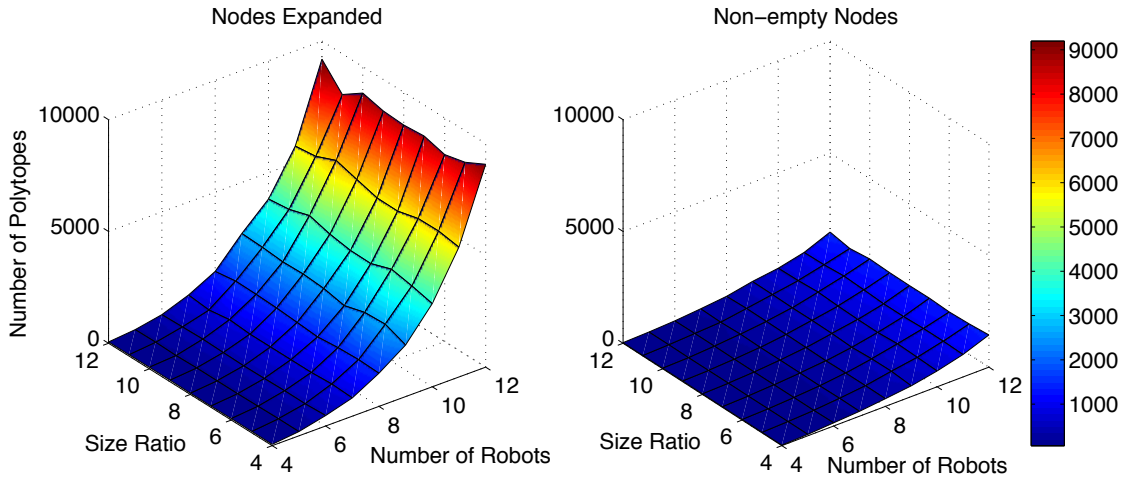
The algorithm is complete based on the choice of abstraction boundary. Since the abstraction is an overestimate of the area occupied by the robots, it is possible that some solutions will be lost. This is especially the case when fixing the boundary of the abstraction while the group is navigating through the space.



(a) Computation time



(b) Path length



(c) Comparison of nodes expanded and those which are valid

Figure 8.6: Effects of congestion and group size on the graph search. (a) The average time spent to find a path to the goal configuration. (b) The average resulting path length. (c) The left panel shows the number of nodes expanded in the graph search, while the right panel shows the number of those nodes which are valid and hence added to the graph.

Chapter 9

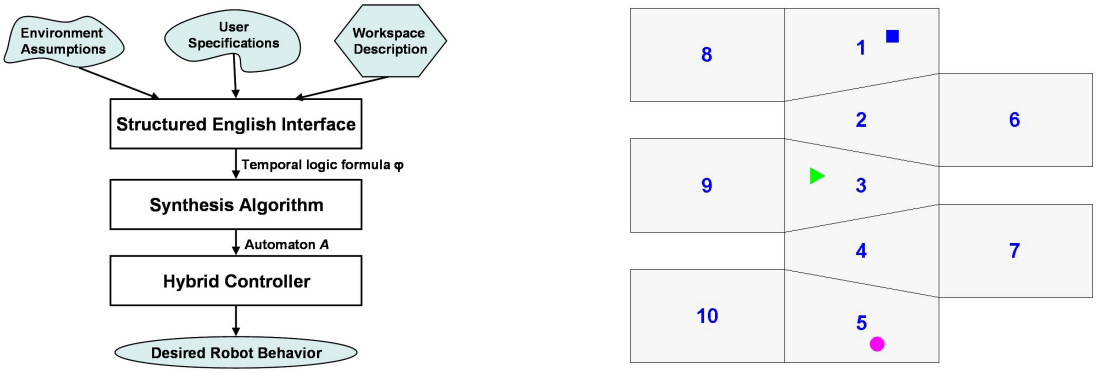
Natural Language Specifications

In this chapter we consider a group of n kinematic robots in a polygonal workspace. The robots have a set of sensors which capture high-level information about the space, and actions, such as transmitting messages or sounding alarms. A high level task is given as a set of Structured English sentences which describe the desired behavior of the robots and the assumptions on sensor information.

Each robot has a set of sensors $Sen = \{s_{i_{\mathtt{S}}} | i = 1, \ldots, n; \mathtt{S} = 1, \ldots, \mathtt{m}_i\}$ that capture high level information about the world (e.g. whether a person is seen or a fire is detected). The robots may also have a set of actions $Act = \{act_{i_{\mathtt{A}}} | i = 1, \ldots, n; \mathtt{A} = 1, \ldots, \mathtt{l}_i\}$ such as picking up objects, transmitting messages, or sounding alarms. In this paper we assume such actions do not have explicit time constraints (minimal or maximal duration).

In addition, we consider a high level task σ given as a set of Structured English sentences that the team must achieve. This task describes the desired behavior of the robots and assumptions on the sensor information.

Problem 9.0.1. *Consider a group of robots in $\mathbb{R}^D$, $D = \sum_{i=1}^{n} d_i$ with dynamics* (3.1)*, sensors Sen and actions Act and a high level specification σ. For any possible*



(a) (b)

Figure 9.1: Overview of method (a) and workspace (b).

initial state $\{\mathbf{x}_0, Sen_0, Act_0\}$ *such that* $\{\mathbf{x}_0, Sen_0, Act_0\} \models \sigma$ *find a control law* $\mathbf{u}$ *and an action activation policy* $\pi : t \to 2^{Act}$ *that for each time* t *specifies which actions should be active, such that*

1. $\dot{\mathbf{x}}_i = \mathbf{u}_i$;

2. *an action* act_{i_A} *is activated at time* t *if and only if* $act_{i_A} \in \pi(t)$;

3. *if* $s_{i_S}(t) \models \sigma$, $\forall t \geq 0$, i_S *(the sensors satisfy the assumptions on their behavior) then* $\{\mathbf{x}_i(t), act_{i_A}(t)\} \models \sigma$, $\forall t \geq 0$, i, i_A *(the robots satisfy the task);*

if such a system exists.

9.1 Task controller

Now we must transform a multi-robot high-level task, captured by Structured English instructions and the discrete representation of the workspace, into a hybrid controller guaranteed to drive the robots according to the desired behavior. The method is demonstrated with a recycling example.

Figure 9.1a shows the three main steps of the approach. First, the user specification and assumptions regarding the environment (the behavior of the sensor inputs, *Sen*) are captured using Structured English sentences. These are then translated automatically into linear temporal logic (LTL) formulas [30] and combined with a discrete abstraction of the workspace to create the formula σ which belongs to a specific fragment of LTL [59, 98]. Next, an automaton A is automatically synthesized such that every execution of A satisfies σ. Finally, a hybrid controller based on the the automaton A is created.

We illustrate these steps with the following scenario. Three robots, a_1, a_2 and a_3, share a free configuration space $\mathcal{C}_i^{free} \subset \mathbb{R}^2$ with ten districts, shown in Fig 9.1b. Initially a_1 (blue square) is in District 1, a_2 (green triangle) is in District 3 and a_3 (magenta circle) is in District 5. The high-level task requires robots to pick up different items from predesignated locations in District 6 and 7, which can be metal, glass, or paper. The items must be deposited, according to composition, in the correct location (metal in District 8, glass in District 9, paper in District 10) while avoiding collisions and deadlocks. Additionally, we impose that there be at most one robot in District 6 and 7 at a time, for the recyclers' peace of mind.

The first step, translation, builds upon [58]. There, the user must first define two sets of binary propositions. One set, *Sen* in Problem 9.0.1, represents information robots gather through sensors and communication. The other represents the state of the robots, controlled by the system, including locations and possible actions *Act*. All these propositions are then used to write the task using Structured English sentences that are automatically translated to an LTL formula.

A task description can be divided into three components, *initial conditions*, *goals*, and *transitions*. The initial conditions capture the state of the environment and system the moment the system is turned on. Goals include assumptions about

the environment and desired behavior for the system. Transitions contain assumed constraints on the changes in sensor information from one time step to the next and constrain possible moves the system can make.

The tasks here involve continuous robot motion. To capture the motion of the robots using the discrete LTL formalism, we partition the workspace into district and create propositions that relate the location of the robots to these districts. For example, a proposition [2 3 8] is true if a_1 is in District 2, a_2 is in District 3 and a_3 is in District 8 and false otherwise. Then, based on adjacency of the regions and allowable room combinations we restrict the changes in these propositions, constraining robot motion to a feasible behavior. Given the decomposition, adding these restrictions to the transitions component of the LTL formula is automatic.

Here the sensor propositions, Sen, are: **pu6, pu7** there is an available item in Districts 6 and 7, respectively; **m1,g1,p1,m2,g2,p2,m3,g3,p3** composition of the item a_i just picked up (metal, glass, paper, respectively). The system propositions relate to different robot actions Act: **a1PU, a2PU, a3PU** a_i should pick up an item; **a1Carry, a2Carry, a3Carry** a_i is carrying an item; **a1D, a2D, a3D** a_i should deposit the item it has been carrying; as well as **robot motion** (locations). The latter correspond to all district combinations: for example, [1 3 5] is true when a_1 is in District 1, a_2 is in District 3 and a_3 is in District 5. Our workspace contains ten districts and three robots; thus there are 1000 possible combinations in general.

Once the propositions are defined, the task must be specified using Structured English sentences. The sentences capture assuptions about sensor or system behavior. We present two example sentences here; the first ($S3$) captures assumptions about sensor behavior, and the second ($S7$) which captures desired system behavior.

S3 "if you activated $a1Carry$ and you sensed $m1$ then always $m1$" (same for g and

p): Once the material type of a carried item is determined, it does not change.

S7 "activate $a1PU$ if and only if you did not activate $a1Carry$ and (you are in [6 X X] and you are sensing $pu6$ or you are in [7 X X] and you are sensing $pu7$)" - If the robot is not carrying an item and it is in a district with an available item, it should pick it up.

These sentences refer to a_1 but the full specification contains the same sentences for a_2 and a_3. In all, for our example there are 14 sentences to define the behavior of the system for each robot, five relating to sensor assumptions and nine relating to system behavior [57].

The next two steps (automaton synthesis and hybrid controller creation) follow the work in [59]. The synthesis algorithm [98] automatically generates an automaton A that implements the desired behavior, *if this behavior can be achieved*. The states of this automaton contain the truth values of the system propositions while the truth values of the sensor propositions guard its transitions. Every execution of the automaton, based on sensor information, is guaranteed to satisfy the desired system behavior.

A portion of the automaton is shown in Fig. 9.2. Circles represent the automaton states; robot propositions written inside each circle are those that are true in that state. Edges are labeled with all sensor propositions that are true when that transition is enabled. In the top most state, the robots are in district combination [6 3 7], a_1 is picking up an item, and a_3 is carrying an item. If there are no more items to pick up (left and right branches, in both $pu6$ and $pu7$ are false) the robots proceed to the drop off location (in the right branch, a_3 drops the paper item in District 10, as required). If there are more items (middle branch, $pu7$ is true) a_2 proceeds to pick up the item.

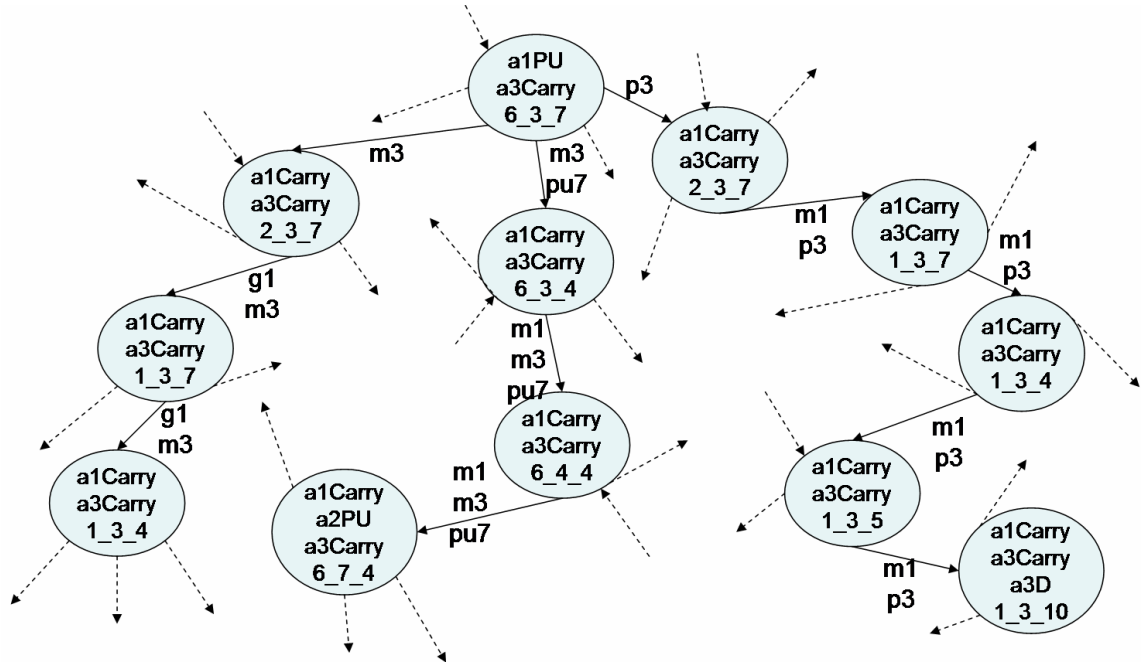


Figure 9.2: A portion of the automaton for our example.

The final step is to construct the hybrid controller that continuously executes A, based on the sensor inputs. Recall, from Problem 9.0.1, that we need to construct a motion control law $\mathbf{u}$ as well as an action activation policy π.

The continuous motion control $\mathbf{u}$ is generated by creating a database of controllers for switching from each room combination to every adjacent room combination. This database of controllers is then used, according to the sensor inputs and the automaton states, to drive the system to the next room combination. As for the actions, for each time t the action policy $\pi(t)$ is the set of all system propositions that are true at the current automaton state.

It is important to note about automata and hybrid controllers created using this method that goals are satisfied cyclically, that is, the first goal written is reached, then the second, and finally after the last goal is achieved the automaton satisfies the first goal again and so on. In our example, this results in the robots first picking

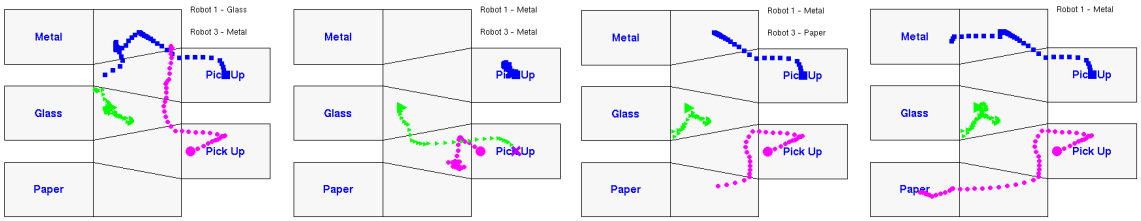


Figure 9.3: Simulation of the automaton segment of Fig. 9.2. (a) Left branch. (b) Middle branch. (c) Right branch, first snapshot. (d) Right branch, second snapshot.

available items until either all robots are carrying something or there are no more items, and only then the robots deposit the items.

Figure 9.3 depicts part of a simulation run and illustrates both the continuous execution of the automaton segment shown in Fig. 9.2 and that using the same automaton, the behavior of the system varies significantly based on events in the environment.

9.2 Feedback controllers

This section addresses the synthesis of multi-robot feedback controllers that drive robots from one location to another while guaranteeing *safety* (collision avoidance) and other specified inter-robot constraints. The controller we use is the centralized version of the controller presented in Chapter 5, in other words, we do not limit communication between agents.

Consider the team of n robots V_A with dynamics (3.1) in some district combination $\mathcal{D} = [\mathtt{p}_1\ \mathtt{p}_2\ \ldots\ \mathtt{p}_n]$, where $\mathtt{p}_i$ denotes the district in which robot a_i is located. One robot, the *active* robot, must transition to a new room without collisions or without any other robots transitioning to a new room (the reason for this will become clear in Section 9.2.1).

The configuration spaces are defined exactly as in Chapter 3, with symmetric collision constraints and a communication range at least as large as the workspace. Therefore the proximity constraints correspond to an infinite square annulus in the relative space of pairs of agents, as in Figure 3.2c. Thus, $\mathcal{C}_T = \mathcal{C}_{all} \cap \mathcal{L}$ (i.e. we consider only collision constraints).

9.2.1 Feedback controllers on the task configuration space

We now consider a subproblem of Problem 9.0.1.

Problem 9.2.1 (Controller Synthesis)**.** *For any initial configuration* $\mathbf{x}_0 \in \mathcal{D}^0 \subset \mathcal{C}_T$*, consider the system* (3.1) *on* $\mathbb{R}^D$*, where* $D = \sum_{i=1}^{n} d_i$*, with goal configuration* $\mathbf{x}^g \in \mathcal{D}^g \subset \mathcal{C}_T$*, and* $\mathcal{D}^0$ *is adjacent to* $\mathcal{D}^g$*. Find a piecewise affine input function* $\mathbf{u} : [0, T_0] \to \mathbf{U}$ *for any* $\mathbf{x}_0 \in \mathcal{D}^0$ *such that*

1. $\forall t \in [0, T_0]$, $x \in \mathcal{D}^0 \cup \mathcal{D}^g$, $\mathbf{x}(T_0)$ *arbitrarily close to* $\mathbf{x}^g$,

2. $\dot{\mathbf{x}}_i = \mathbf{u}_i$,

3. $\mathbf{x}(t) \in \mathcal{L}$, $\forall t \in [0, T_0]$.

Note that Problem 9.2.1 is for driving the state from one district combination to an adjacent one, and not for generating a path on all of $\mathcal{C}_T$. Since the controllers developed in earlier chapters direct states through a facet, not an edge, only one robot will cross a room threshold at any time. Thus, we restrict the automaton to commands which result in room change for only one robot, limiting the path on the polytope graph to the polytopes in the initial and final room combination. Once in the polytope containing the goal configuration we steer states to the goal configuration.

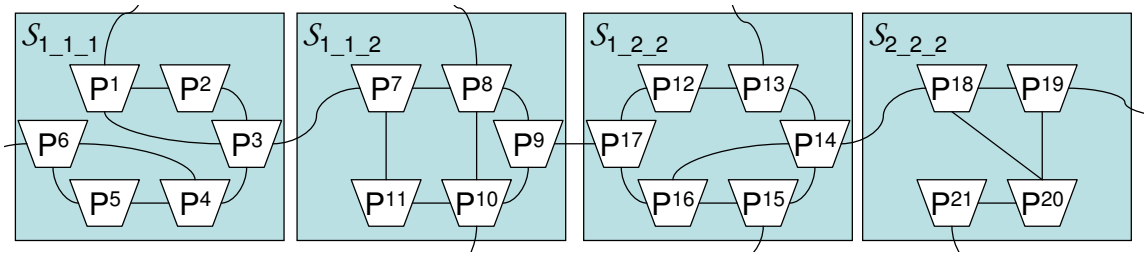


Figure 9.4: A partial view of a polytope graph for three robots.

To solve Problem 9.2.1, we first build a hierarchical discrete representation of the task configuration space and find paths in this discrete representation. Second, we translate these paths into feedback controllers.

In the first stage, we associate each polytope with a district combination, $\mathcal{S}_{\varpi_1\_\cdots\_\varpi_n} = \{P^k | \mathbf{x} \in P^k \Rightarrow \mathbf{x}_1 \in \varpi_1, \cdots, \mathbf{x}_n \in \varpi_n\}$, where P^k is a polytope in $\mathcal{C}_T$. We define the upper adjacency graph on these district combinations and we define the polytope graph, $\mathcal{G}_P$ as usual. A sample polytope graph is shown in Fig. 9.4. $\mathcal{G}_P$ is used in the creation of the automaton; thus, the automaton cannot give instructions that violate the collision constraints (and communication constraints, if included).

For all transitions from one district combination to another, we determine a discrete path from each polytope in the original district combination to the next district combination. For example, referring to Fig. 9.4, the paths from $\mathcal{S}_{2\_2\_2}$ to $\mathcal{S}_{1\_2\_2}$ may be $P^{19} \rightarrow P^{18} \rightarrow P^{14}$ and $P^{21} \rightarrow P^{20} \rightarrow P^{18} \rightarrow P^{14}$. If the active robot must stay in a district (to pick up/deposit) we determine a discrete path from each polytope in the current room combination to the goal polytope. Inactive robots stay in their current rooms and go to a goal position, (the goal position is described for every robot). We are not concerned with whether the inactive robots reach their goal position.

We use an algorithm such as Dijkstra to choose a path which minimizes the

number of polytopes visited, which minimizes the number of transitions between polytopes. Since we do not limit the state information available to the robots, we can guarantee that all paths on the graph can be translated into feedback controllers to solve Problem 9.2.1 by using either the centralized version of the controller in Chapter 5, the potential function controllers in Chapter 6, or the vector field controller by Lindemann and LaValle introduced in Chapter 7.

In summary, the algorithm for controller synthesis or the solution to Problem 9.2.1 involves the following steps:

Algorithm 9.2.2 (Controller Synthesis)**.**

1. *Construct task configuration space $\mathcal{C}_T$, taking into account collision and other desired constraints.*
2. *Construct the polytope graph, $\mathcal{G}_P$, associating each polytope to a district combination.*
3. *For each district combination, find paths from every polytope P^k in $\mathcal{D}^0$ to every adjacent district combination.*
4. *For each possible exit facet for each polytope P^k, synthesize a controller which drives every state inside P^k to the exit facet.*
5. *For each P^k which contains a goal state, synthesize a controller which drives every state inside P^k to the goal configuration.*

This algorithm will create a database of controllers for all possible commands from the automaton.

9.3 Simulation

In this section we show a MATLAB simulation and demonstrate how different tasks can easily be accommodated using the same controllers but a different automaton. The atomic controllers were designed in MATLAB using the Multi-Parametric

Toolbox for polytope computations [61]. The automata were synthesized using a prototype of the JTLV system [107].

Figure 9.5 depicts a sample simulation. In this scenario, there is always something to pick up (denoted as a purple X) in both locations. The robots a_1 (blue square), a_2 (green triangle), and a_3 (magenta circle) start in Districts 1, 3, and 5 respectively. First a_3 goes to District 7 and picks up an object (Fig. 9.5a), then a_2 picks up in District 7 (Fig. 9.5b) then a_1 picks up in District 6. Note that as discussed in Section 9.2.1, for every discrete transition in the automaton only one robot is changing the region it is in.

Once all robots have identified their carried item (Figs. 9.5b and 9.5c) they drop it off appropriately: a_3 drops off the paper item (Fig. 9.5d), a_1 (a_2) drops off a glass (metal) item (Fig. 9.5e). Since there are more items to pick up, the robots move towards the pickup rooms and a_3 picks up more paper in District 7 (Fig. 9.5f).

Adding robot motion constraints can be done in two ways. One is to explicitly state such constraints in the user specifications. The other is to remove nodes and transitions from the discrete representation of $\mathcal{C}_T$.

Example 9.3.1. *Add a "baby sister" constraint requiring robot a_2 to always follow robot a_1, i.e. they must always be either in the same or adjacent rooms.*

Adding this constraint reduces the number of lower-level controllers from 944 to 256 and the resulting automaton contains 6,874 states.

Sensor inputs as well as system outputs can be added in a very flexible way as long as the added specification does not create a logical contradiction with the previously specified task or results in an infeasible motion request.

Figure 9.5: Simulation of recycling example. (a) a_3 picks up in Room 7. (b) a_2 picks up in District 7. (c) a_1 picks up in District 6. (d) a_3 drops off a paper object. (e) a_1 and a_2 drop off a glass and a metal object respectively. (f) a_3 picks up in District 7 again .

9.4 Remarks

We have presented an application of the methods described in Chapters 3 and 5 to designing provably-correct control policies for robot teams that achieve complex high-level tasks which are described in Structured English while providing low-level guarantees of collision and deadlock avoidance. This involves creating a discrete automaton satisfying the task and a database of controllers which can continuously implement every possible transition in the automaton.

Given a workspace decomposition and the robots' capabilities, the method is entirely automatic and "recyclable" with minimal additional computation. Furthermore, changing the specification and adding more sensor information or different robot actions is simple. This results in an extremely flexible system that allows non-experts to design complex systems that perform a large variety of interesting tasks.

Although this method requires an initial preprocessing stage to create the low-level controllers (which can be computationally expensive) the method requires only up-front user input (the space, number of robots, proximity constraints and the high-level specification) and no hand-tuning. Furthermore, the controllers are reused to accommodate a wide variety of high-level tasks.

By using a different, less computationally expensive controller, such as the non-linear controller presented in Chapter 6, it is possible to calculate controllers online, which significantly reduces the overhead necessary for this application. Since the controller is guaranteed to have a solution for every possible exit facet, we still avoid deadlock situations. Furthermore, it is not necessary to calculate more than the free configuration space of each robot and the associated adjacency graph in advance. This we present in the next chapter.

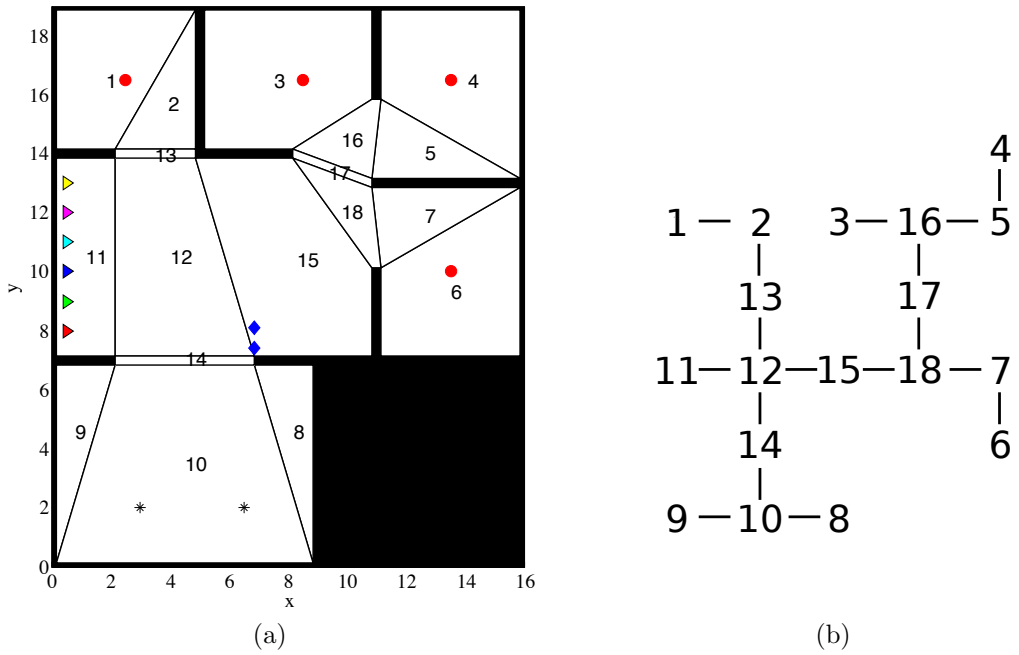
Chapter 10

Dynamic Conditions and Object Manipulation

In this chapter, we present a simulation which involves dynamic team organization and communication graph, manipulating objects in the environment, ongoing task assignment and re-assignment, and stigmergic interactions. We do this using the tools developed in Part II, but with feedback controllers synthesized online instead of *a priori*. This is preliminary work; it is included to demonstrate the versatility of the work presented in this thesis and its potential to be applied without precomputation, in partially unknown environments, and to object manipulation.

Six robots, $\mathcal{V}_a = \{a_1, \ldots, a_6\}$ with dynamics (3.1) and configuration $\mathbf{x} = [\mathbf{x}_1^{\mathrm{T}} \cdots \mathbf{x}_6^{\mathrm{T}}]^{\mathrm{T}}$ are tasked with finding and transporting two circular objects into designated positions. We set the minimum distance between robots as $\boldsymbol{\delta} = 0.5$m. The environment as well as the object to be moved are padded by 0.15m.

We assume that the initial tesselation of the padded workspace, shown in Fig. 10.1a, is given. We also assume that initially no robots are in districts 8, 9, 10, or 14, and that the robots are capable of calculating the adjacency graph shown in Figure 10.1b.



(a) (b)

Figure 10.1: Workspace for six robot simulation. (a) Robots are shown at the start configuration in district 11, and possible locations of objects to be moved are shown as red dots in districts 1,3,4, and 6. (b) The adjacency graph of the workspace.

Each circular object requires four robots to move. The robots are aware of four possible locations for the two objects, in districts 1, 3, 4, and 6, marked with red dots. The initial task is to explore these four districts to find the objects. Next, two robots must push open the door to district 10 while four robots cage the object to move it to one of the goal locations in district 10, marked with black stars in Figure 10.1a. Finally, four robots move the second object while the two remaining robots return to the start position.

The robots do not know which of the four possible districts will contain the two objects which must be moved. Additionally, it is not known in advance which robots will explore which areas; task assignment is done online based on the state of the system, and can be done in a decentralized way. We assume each robot is capable of calculating their cost-to-go for each task location. The robot with the lowest cost for

any of the tasks is assigned first, followed by the lowest cost among the remaining untasked robots, and so on. This assignment algorithm lends itself well to auctioning methods.

Once the initial task assignment is complete, the remaining tasks are assigned by combining stigmergic interactions (which we discuss in more detail in Section 10.2) and cost-to-go calculations. The cost-to-go is calculated as a weighted combination of distance on the graph, where the cost of each edge is the distance between the centroids of each cell plus the Euclidean distance (without taking into account obstacles) to the task location. Although this is how tasks are assigned in this simulation, any task assignment algorithm will do.

10.1 Dynamic communication graph

We assume that robots can only communicate with each other if they are in adjacent districts. This means that the communication graph changes as the agents move throughout the space. We also assume that robots can pass information through the communication graph. For example, if a_1 is neighbors with a_2 which is neighbors with a_3, then a_1 can receive state information about a_3 and vice versa. If there is no chain of adjacent agents between a pair of agents, then they do not have access to each others' state. In this way, robots can avoid collisions while reducing complexity. However, there do exist configurations such that two robots are not in adjacent cells but can collide. For example, near the shared vertex of districts 16 and 4, two robots can be within collision distance. This situation can be avoided by using a different decomposition or by employing a rule which allows robots to communicate based on the specific cells they are in rather than a blanket rule of adjacent cells. (Note that the rules for communication cannot change once motion has commenced, as that

could cause cycling and instabilities.)

Agents explore a district by entering and navigating to the potential object location. Once they are within a specified threshold of the where the object would be located (here, 0.8m from the object center), they can sense whether the object is there (by bump sensors, sonar, vision, etc.). Although this is how objects are detected in this simulation, there is no reason why other methods could not be used, such as sweeping the district until either the object is found or the entire district has been covered.

10.2 Stigmergic interactions

To lessen the communication burden, we include stigmergic interactions. Recall that stigmergy is a mechanism by which agents modify their local environment, thereby affecting the actions of other agents. Each of districts 1, 3, 4, and 6 are equipped with indicator lights which are visible from the main hall (districts 11, 12, 15); every agent has access to network nodes which can modify the status of these indicator lights by sending packets of data to the nodes (i.e. pinging the node). The indicator light nodes control the color of an indicator light according to the number of packets received, which determines the status of the district. The indicator lights have one of four states: *off*, *green*, *yellow*, or *red*.

All lights start in the *off* position. When an object is found in a district, the agent pings the indicator node, which changes the light to *green*, indicating that the district contains one of the objects to be moved, and has no robots yet assigned to it. Once a single robot has assigned itself to an object, it pings the indicator, which turns the light *yellow*. The indicator remains yellow until four robots have assigned themselves to it (i.e. five packets received). Once four robots have committed to

moving an object, the indicator is changed to *red*, indicating that there are a sufficient number of robots committed to moving the object. The remaining robots must open the door to the storage districts 8, 9, and 10.

The indicator lights determine how robots in the main hall, which can observe the lights, will behave. If a robot is waiting in the main hall and senses a green light, it assigns itself to that object by pinging the indicator light node. If a robot senses both green and yellow lights, it favors the object in the room with the yellow light, which already has at least one robot assigned to it. Should a robot sense a red light, that indicates that four robots are already assigned to an object, therefore, it assigns itself to open the door to the storage area, since the door must be opened before objects can be moved into districts 8, 9, and 10. If our simulation included more than six robots, it would be possible for a robot to assign itself to move another object; however, with four robots needed to move an object and two robots needed to open the door, the only option is to open the door once four robots have been assigned to an object. The first three robots which sense the yellow indicator light assign themselves to moving the object in that district.

Once an agent explores a district, it sets the indicator light state, then returns to the main hall. Though we assume each robot has the ability to ping an indicator light node, the robots could also navigate to a light switch inside each district and activate it. It is important to return to the main hall even if an object is found in that district since other agents might already have assigned themselves to another object. Any agents which currently have no assignment and are waiting in the main hall can ping an indicator light to signal that they have chosen that task. Agents choose tasks based on cost-to-go; therefore, if two indicator lights turn green at exactly the same moment or if an agent senses two green lights at the same moment, it chooses the task with the lowest cost-to-go.

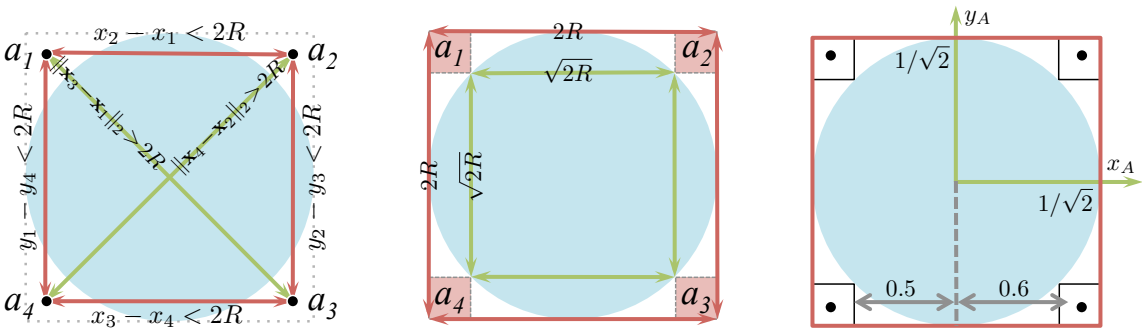
The door to the storage area has a handle which extends in the y-direction when the two robots are in position and requires two robots to push open. The start positions to open the door are shown as blue diamonds in Figure 10.1. To maintain contact with the door handle, the robots must stay within a specified distance of the door. Once open, the door stays open, and the handle is returned to its original position. The agents are then free to assign themselves to other tasks.

10.3 Abstractions and object manipulation

To move the circular object, the four agents must surround it so that it cannot escape, or *cage* the object. To get around the object without collision, the agents must subdivide the districts near the object, and remove from the object from the free space. Since the object is circular, it is overestimated as a square rotated to 45^o. We assume that the size of the objects are known, although their extents could be perceived by robots equipped with the proper vision capabilities. We also assume that the robots are capable of tessellating the workspace around the object.

To properly cage the circular object, interrobot distances must comply with the constraints shown in Figure 10.2a. We linearize the constraints to those in Figure 10.2b, where the outer red boundary is the virtual boundary for the group, and each robot must stay within its respective red-shaded square within the virtual boundary.

Once the object is caged, the group must navigate together to the deposit point in district 10, marked with stars in Figure 10.1a. The robots in the group must calculate the abstract configuration space, shown in Figure 10.3b. Here the original free configuration space boundaries are shown in grey, while the black border shows the reduced area for group navigation. The shape vector for the virtual boundary is



(a) (b) (c)

Figure 10.2: Interrobot constraints required to cage the object with four robots. (a) The full set of constraints. (b) The reduced set of constraints used. (c) The local configuration space inside the abstraction boundary, lengths in meters.

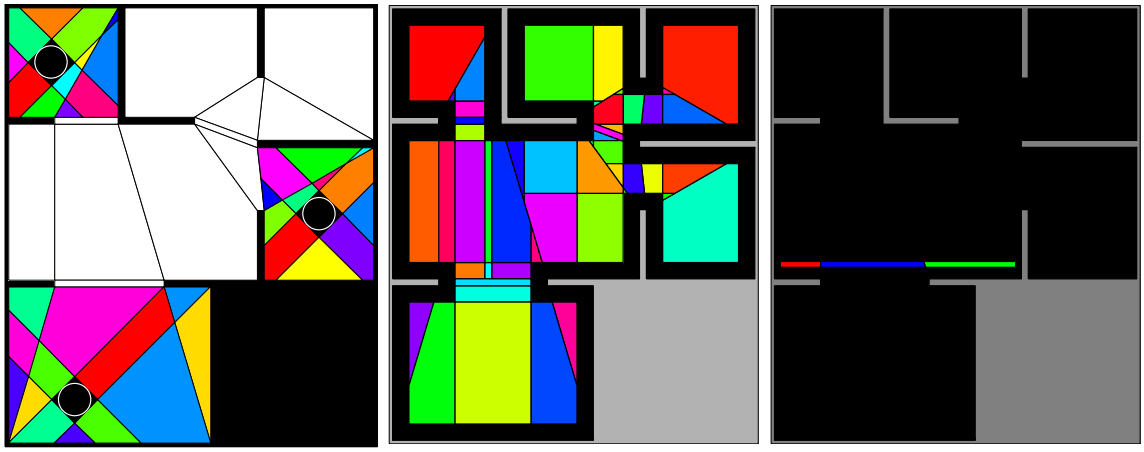
$s_{\text{cage}} = (s_w, s_h) = (1.4\text{m}, 1.4\text{m})$, and the abstract configuration space takes approximately 8s to compute.

To open the door to the storage area, a pair of robots must move along the horizon until the door is fully open. Since the maneuver is very constrained, the size of the virtual boundary is constant, at $s_{\text{door}} = (0.5\text{m}, 1\text{m})$, and the center of the agents' virtual boundary must stay within the colored areas in Figure 10.3c. The group motion to open the door is shown in Figures 10.4i and 10.5a.

Due to the constrained nature of the tasks in this simulation, we do not consider deformable or rotating virtual boundaries, however, the boundaries can be made deformable or rotatable for other tasks.

10.4 Discrete path planning

At any point in time, the graph for the team may be connected or disconnected. Any robots which are connected on the communication graph together plan a joint path on their discrete task configuration space. Our strategy is to allow the robot with the longest discrete path to transition first. For example, in Figures 10.4a and



(a) (b) (c)

Figure 10.3: Further tessellations of the workspace. (a) To get around the circular object without collision, both when initially caging it and depositing it, the space must be subdivided. (b) The abstract configuration space for the group of robots while caging the object. (c) The abstract configuration space for the group of robots which must push open the door to the storage area.

10.4b, the cyan robot is assigned to check district 4 for an object, which corresponds to the longest discrete path; therefore it transitions to district 12 first. An example of a disconnected graph is shown in Figure 10.4c, where at the current time, the cyan, magenta, and blue robots are connected so they plan together, the red and green robots are connected so they plan together, and the yellow robot plans on its own.

Each connected subset of robots plans a path until the robot with the longest discrete path transitions to an adjacent district. The connected subset of robots replan when either (a) the designated robot has transitioned to an adjacent district, or (2) the communication graph changes. The communication graph can change if another robot (or group of robots) enters communication range.

Groups are treated as large robots that can communicate with any external robots in districts adjacent to any robot within the group. For example, in Figure 10.5a,

the group reference point in district 13 and the yellow and magenta robots are in district 11, yet since the cyan and green robots are in district 12, which is adjacent to district 11, they can communicate. Robots which can communicate with the group do not take into consideration how many robots are in the group or the configuration of robots within the group. They need only the extent and coordinates of the virtual boundary and any external connected robots to plan a path together.

We allow groups to have precedence over individual robots; however, the task of opening the door always has precedence over any other task. Although this works in our example, it may not make sense in other applications; this is for the user to decide.

10.4.1 Switching strategy

Switching strategy is extremely important to providing convergence guarantees. Since a path is planned for transitioning through districts and not individual polytopes, we can guarantee that at each switch of the path and hence the underlying controller, the team is consistently making forward progress toward the goal. Since we measure progress by the same measure for all subgroups of robots (that an agent or group has crossed a district threshold in the correct direction), we can think of this progress measure as a discrete Lyapunov function which is always decreasing. If the Lyapunov function of a switched system decreases upon each switch, we can guarantee stability [14].

10.5 Controller synthesis

Control synthesis is done entirely online, using the navigation function approach presented in Chapter 6. This simulation presents a few challenges with respect to

control synthesis.

Firstly, in order to prevent many small polytopes from appearing in the configuration spaces and unnecessary increases in computation time, we have refrained from slicing the abstract configuration space and the subdivided configuration spaces around the object so that all polytopes have matching facets (see Figs. 10.3a and 10.3b). Instead, we take a hierarchical approach to constructing the adjacency graph, much like the upper and lower adjacency graphs introduced in Chapter 4. For the subdivided spaces around the objects, we ensure that (1) the object is contained in an obstacle which still allows for caging, and (2) there are matching facets inside the two districts that make up the room. For the abstract configuration space in Fig. 10.3b, we ensure that there are matching facets within each district, and drive the group to the interface. We plan a discrete path on the districts, then a secondary path through these further tessellated polytopes. Since we are using the navigation function controllers presented in Chapter 6, we must choose a goal point that is inside the next polytope. To do this, we consider the next polytope as the intersection of the next district with the subdivided cells (this can be a group of polytopes). We then extend the current polytope into this group of polytopes by removing the constraint at the interface as in Fig. 6.2, and choose a goal point within this overlapping area (again we choose the Chebyshev center).

We take a conservative approach to caging the object, by limiting the possible relative locations of the agents more than necessary. Figure 10.2c shows the local configuration space within the abstraction boundary, the goal locations of each robot within the boundary (black dots), as well as the limits of each robots possible location in the local configuration space (this is shown by the black boundaries).

10.6 Simulation

In this section we present the results of the simulation we have discussed. All computations were done on a MacBook Pro with 2.53 GHz Intel Core 2 Duo processor, with 4GB 1062 MHz DDR3 memory, running MATLAB 7.11.0. All polytope computations we done using Multiparametric Toolbox for Matlab [61].

Figures 10.4 and 10.5 present sequential frames of the simulation. For clarity, we switch notation so that $\mathcal{V}_a = \{a_r, a_g, a_b, a_c, a_m, a_y\}$ where subscripts correspond to the first letter of the agents' color. Figure 10.4a shows the initial configuration, and the intial assignment of robots to tasks. Task locations for each robot are shown with color-coded triangles. Each sequential screenshot shows the color-coded trajectory each robot has taken since the last screenshot, and the current location of the robot with a black circle. In Fig. 10.4b, a_c has discovered there is no object in district 4, and must return to the main hall. Each agent has a waiting location in district 15, and returns there when its task is complete in order to wait for a different task. In Fig. 10.4c, a_y discovers an object in district 1, pings the indicator light node to turn green, and since a_r and a_g are waiting with no task assignment in the main hall, they assign themselves to moving this object by pinging the indicator node which changes the light to yellow. Figure 10.4c also shows the waiting locations for a_y and a_c. In Fig. 10.4d, a_b has discovered an object in district 6, and pinged the indicator node for the district, turning the indicator light to green. Also, a_c has just entered the main hall, and sensing the yellow indicator for district 1, assigns itself to the task. Figure 10.4e shows that a_b is the next agent to return to the main hall and sense the yellow indicator for district 1, assigning itself. Now, since 4 agents have been assigned to district 1, the light is turned to red, and a_m assigns itself to opening the door.

Figure 10.4f shows the robots beginning to surround the object in district 1, and a_y assigning itself to opening the door. Figure 10.4g shows progress on caging the object in district 1. In Fig. 10.4h, the object is successfully caged, and the group of robots build the abstract configuration space. Figure 10.4i shows a_y and a_m have reached their goal location and form a group to open the door.

In Fig. 10.5a, a_y and a_m have opened the door and now assign themselves to moving the other object. Here the group which has caged the object takes precedence while a_y and a_m wait for the group to pass in Fig. 10.5b. Once the group is out of communication range, in Fig. 10.5c, a_y and a_m can now move freely to their task locations.

Once the first object has been deposited at the desired location as in Fig. 10.5d, a_r, a_g, a_b, and a_c dissolve their abstraction and move to their waiting areas in district 15 to check if another task is available. They move around the object via the tessellation of districts 8, 9, and 10, as shown in Fig. 10.5e. Once a_r and a_g have reached the main hall and sense the yellow indicator for district 6, they assign themselves to that task by pinging the node, turning the indicator to red as in Fig. 10.5f. Now, all that remains for a_c and a_b once entering the main hall is to return to their start location, while the remaining agents cage the last object in Fig. 10.5g. The group cages and moves the object, which enters into communication with a_c and a_b in Fig. 10.5h. This causes some motion for a_c and a_b, which is reversed once they are no longer in communication with the group as in Fig. 10.5i. Finally, in Fig. 10.5i, the group has transported the object to its designated location.

10.7 Remarks

In this chapter we presented a simulation which utilizes many of the tools presented in this thesis, as well as displaying the versatility of these tools by utilizing them in ways we have not previously presented. We have shown that task configuration spaces for multirobot problems can be generated online using an A* algorithm with a valid heuristic. Abstract configuration spaces can also be computed online, as we have computed the entire abstract configuration space for the group of four robots in approximately 8 seconds. Controller synthesis, when using the navigation function based method, can also be done online (the decentralized feedback control, since it requires triangulation, cannot be synthesized in real time). We have also shown that the communication graph need not be static as we have previously stated; in our simulation, not only did the communication graph change, but the number of agents changed, as robots came in and out of connectivity and groups of robots were represented as a single entity.

We have used stigmergic interactions to lessen the communication burden by signaling task locations and task status with indicator lights. We have demonstrated that these controllers can be used for caging and transporting objects in the environment. Additionally, we have shown that it is not necessary to have a database of controllers to use when a task is assigned as we had in Chapter 9, and that tasks can be assigned and groups can be formed on the fly.

Finally, all of this was entirely automatic, requiring no hand tuning.

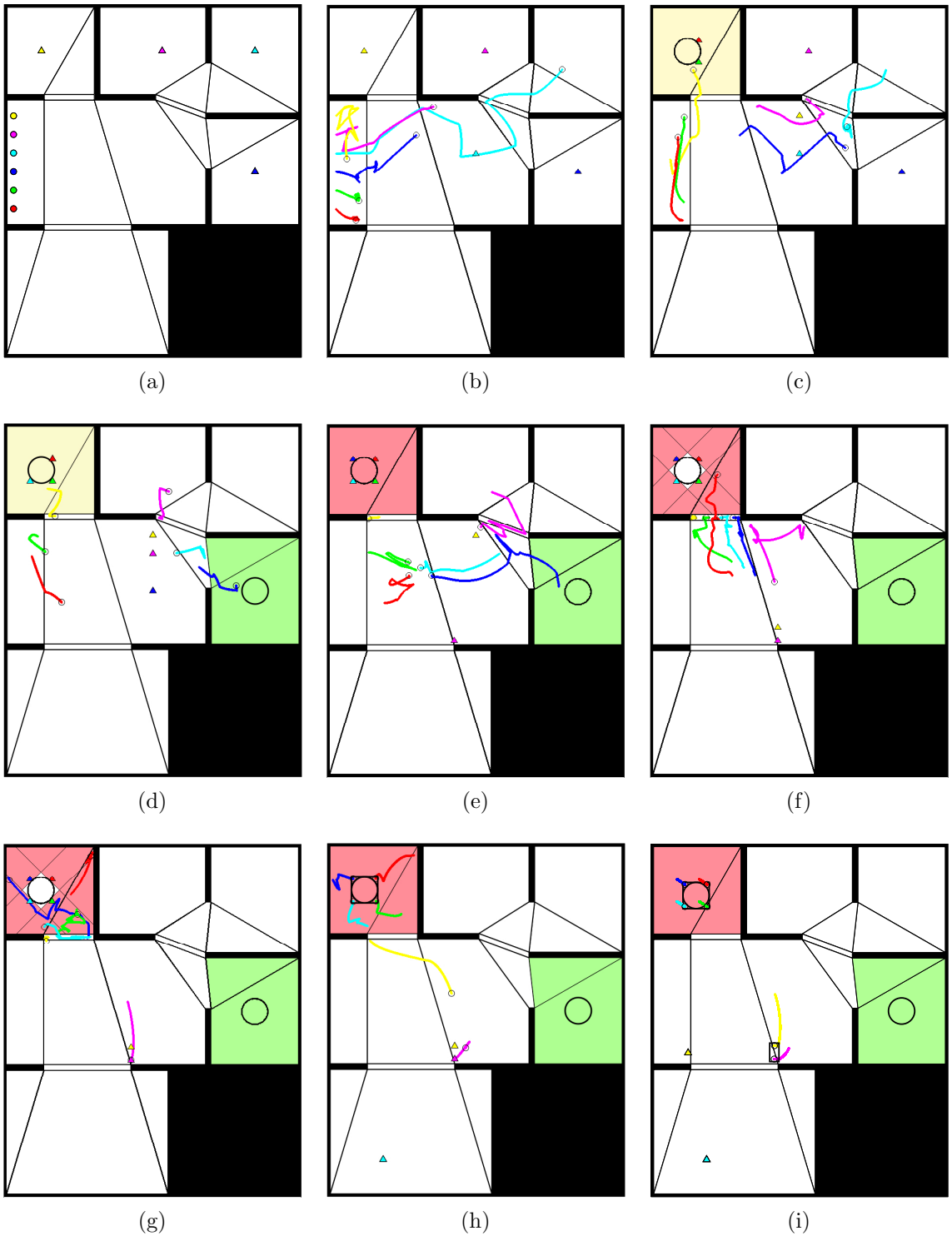
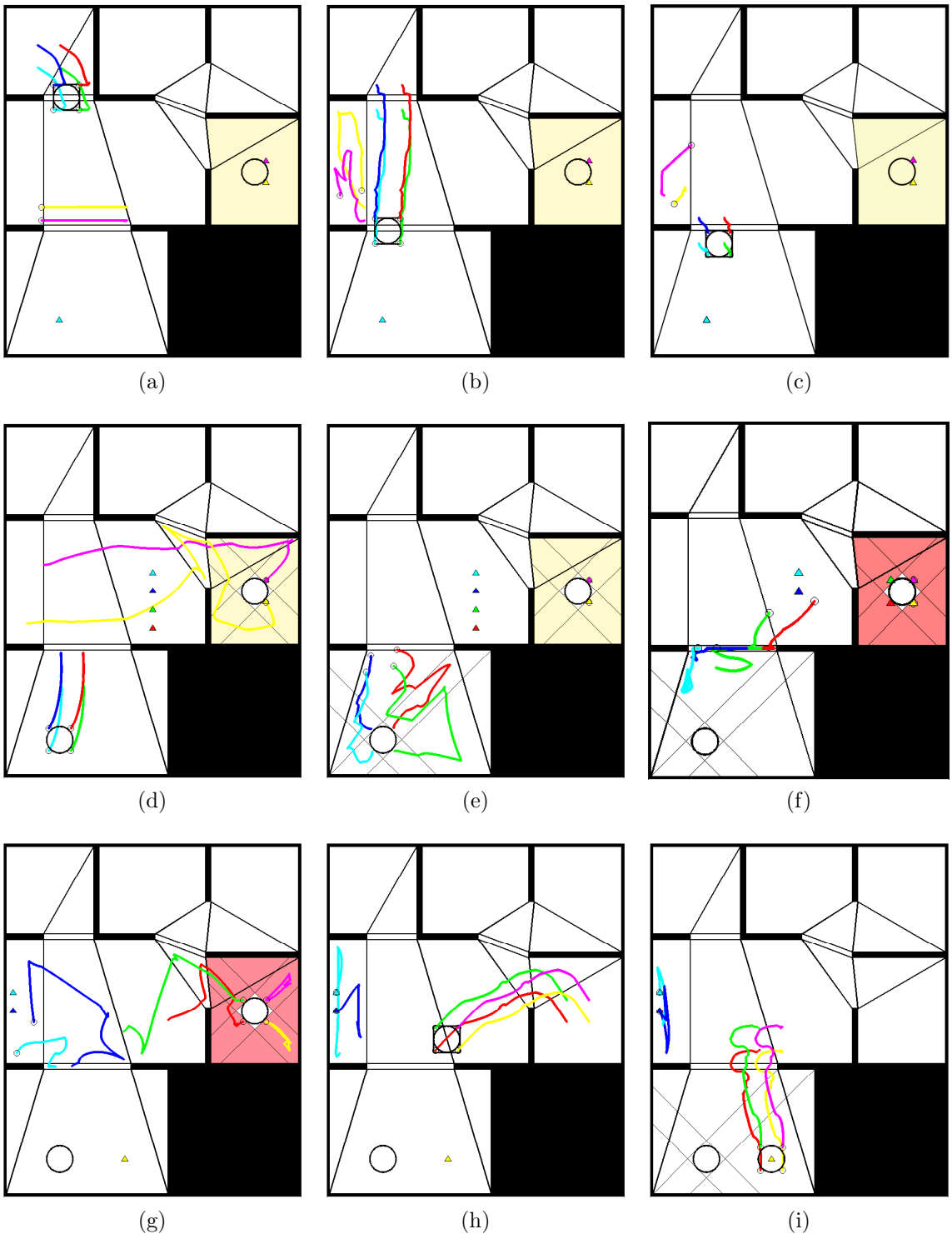


Figure 10.4: Selected sequential frames of the simulation. The robots' current locations are shown with small black circles, while their trajectory since the last frame is shown as a colored line. Objects are shown as large black circles.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 10.5: Selected sequential frames of the simulation, continued.

Chapter 11

Concluding Remarks

Autonomous multirobot coordination poses challenges which impede its expansion into real-world scenarios such as search and rescue, reconnaissance, and construction. We can address these challenges in two ways: by increasing the number of robots and decreasing their individual capabilities, and having provably correct, automatically synthesized feedback policies which provide global guarantees on convergence and safety. We have addressed the problem of automatic synthesis of feedback policies for multirobot problems with these guarantees. The controllers we have developed have been demonstrated in situations of limited communication, on dynamical systems such as quadrotors, and on heterogeneous teams of robots. Furthermore, we have addressed the complexity of group navigation by developing an abstraction on groups of robots, rendering the complexity of group navigation independent of the number of robots in the group.

While feedback policies which provide these types of guarantees are, in general, computationally complex, requiring Cartesian products of configuration spaces, we have demonstrated their ability to be used in real-time in Chapter 10. The ability to rapidly subdivide a space online and use the subdivision for navigation enables

these controllers to function in partially unknown environments while still providing the same guarantees as in known environments.

The feedback policies we developed are also extremely versatile. As we have demonstrated in Chapter 10, they can adapted to dynamic communication graphs, partially unknown environments, and object manipulation. Furthermore, we have shown that these traditionally communication-heavy algorithms can be lightened by using stigmergic interactions instead of direct communication, while still providing the same guarantees.

11.1 Contributions

The contributions of this thesis are three-fold:

Configuration space modeling for groups of robots

We present two methods of modeling configuration spaces for groups of robots. The first models the case when robots are assigned individual goals, and have geometric inter-robot constraints, such as collision, communication, and mutual exclusion constraints. Any constraints which can be encoded as halfspaces are admissible, which allows for a wide range of constraints and capabilities of this model. By exploiting the fact that we are taking Cartesian products of known graphs, we also generate a heuristic which can be used for an informed graph search, significantly reducing precomputation time.

Abstractions on groups of robots

The second models the case when the goal orientation is not finely specified, or the goals lend themselves to navigating as a group. In this case, we use an abstraction on the group to decouple the navigation and inter-robot coordination problems. The abstraction defines a virtual boundary for the group of robots, which deforms and

rotates as it navigates the environment. To accommodate for rotations and still result in a configuration space composed of polytopes, we slice the angular component and overestimate the virtual boundary in each slice. Within the virtual boundary, the robots coordinate to ensure the desired internal behavior is achieved.

Synthesis of feedback policies with convergence and safety guarantees

Finally, once we have modeled the configuration space, we build a discrete representation of it, on which we find a path. This path determines the flow through the polytopes, which is used for sequential composition of feedback controllers. We have proposed two feedback controllers. The first, decentralized affine feedback, is a vector field solution. We solve for inputs at the vertices of a simplex, then interpolate inside to generate the vector field. The controller is decentralized in a sense that if two agents are not communicating, their feedback does not depend on each others' state. The controller is solved by linear programs, but requires triangulation of the space, which in turn requires enumerating the vertices of the polytope, and hence is cost prohibitive in higher than 6 dimensions. The second is a nonlinear, analytical feedback for dynamical systems. This controller requires only the halfspace description of polytopes, which enables its use in higher dimensions (we have shown its use in as high as 12 dimensions). Unlike vector field approaches, the analytical approach lends itself to a graph embedding that causes second order systems to behave more like first order systems, making it desirable for highly dynamical systems such as quadrotors.

Our approach guarantees convergence to the goal. Furthermore, it guarantees safety: in other words, the system will satisfy all of the set constraints, including collision avoidance, communication maintenance, mutual exclusion, and any others.

While our approach is computationally complex when used to find global solutions to the navigation problem, we have demonstrated its use in real-time as local

navigation solutions in the final simulation. Furthermore, we have demonstrated the versatility of this approach, by demonstrating its use to manipulate objects in the environment, incorporating stigmergic interactions, creating and dissolving groups online, and modifying communication graphs based on location in the environment.

11.2 Combinatorics of task assignment

A major contributor to complexity of multirobot problems is task assignment. While algorithms to assign agents to specific tasks are beyond the scope of this thesis, this is still extremely important. The problem of efficient task assignment is critical in many applications. The more efficient the task assignment, both in computation and in end result, the faster all tasks can be completed. There are two main aspects to task assignment. First is the representation of the organization of robots. We seek an elegant representation of the breakdown of groups, so that we know the current organization and the desired organization. Second is determining the most efficient sequence of organizations to solve the problem.

11.2.1 Representating organizations

To represent organizations, we borrow notation from symmetric group theory [108]. The symmetric group, $\mathcal{S}_n$, is the set of all bijections from $\{1, 2, \ldots, n\}$ to itself. Elements in $\mathcal{S}_n$ are permutations π. A Young's tableau provides a visual organization of partitions of a team of robots λ. For example, the tableau

$$t_1 = \begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array}$$

has partition $\lambda = (2, 2)$ and is row equivalent ($t_1 \sim t_2$) to

$$t_2 = \begin{array}{cc} 2 & 1 \\ 3 & 4 \end{array},$$

since the corresponding rows contain the same elements. A set of all row equivalent tableaux are a Young's tabloid

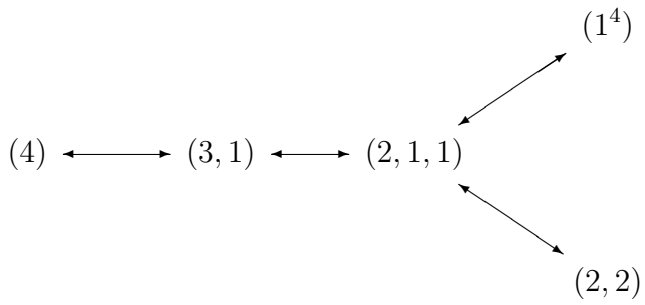
$$\{t\} = \left\{ \begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array}, \begin{array}{cc} 1 & 2 \\ 4 & 3 \end{array}, \begin{array}{cc} 2 & 1 \\ 3 & 4 \end{array}, \begin{array}{cc} 2 & 1 \\ 4 & 3 \end{array} \right\} = \begin{array}{|cc|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}.$$

The Young's tabloid provides an effective way of representing an organization of robots, where each group is contained in a row, without ordering within the group. The Hasse diagram for partitions allows us to order partitions by dominance. The Hasse diagram for partitions of 4 is

$$(4) \longleftarrow (3,1) \longleftarrow (2,2) \longleftarrow (2,1,1) \longleftarrow (1^4)$$

While the Hasse diagram provides a visual representation of some transitions from one organization to another, it does not describe effectively all possible transitions. For example, it is possible to transition $(2, 1, 1) \to (3, 1)$. However the strict ordering of Hasse diagrams does not allow this. Therefore, it may be desirable to create a

new diagram, representing an adjacency of partitions.



This diagram in conjunction with permutations on the tabloids can be used to determine optimal ways to assign robots to different groups for different tasks.

11.2.2 Task assignment in the literature

In the simulation presented in Chapter 10, task assignment was determined by minimizing a weighted sum of discrete and Euclidean distances to the task location. However, the example is not limited to this type of task assignment. One can optimize a utility function for the group, which could also take into account battery status, robot fitness for a particular task type, quality of network connectivity, etc.

Determining the most efficient assignment of robots to subtasks has been addressed in the literature for many types of problems [42, 80, 85, 88, 93, 113, 117, 121]. Gerkey and Mataric provide a taxonomy and formal analysis of task allocation for multirobot systems [42]. There exists a full spectrum of the level of communication required to determine the assignment, as demonstrated in the multiple algorithms in [80]. Perhaps the most desirable aspect of any task assignment algorithm is decentralization [85, 93, 113]. The type of task also makes a large impact on the most efficient assignment, such as specialization for cooperative transportation tasks [88].

11.3 Future work

We have already established the versatility of this methodology by synthesizing controllers online, transporting objects, incorporating stigmergic interations, and forming and dissolving groups online as necessary. However, we have only touched on the last three of these issues. These are directions which I would like to explore in the future, in addition to formalizing the preliminary work presented in Chapter 10.

11.3.1 Interaction with the environment

True interaction with the environment is much more than object manipulation, which is in turn more than transporting a circular object. Transporting a general geometric object is much more complex, since rotations of the object can mean the difference between successful caging and object escape. We can address rotations by ensuring that the robots maintain a formation which not only cages the object in its current orientation, but prevents the object from rotating beyond the capabilities of the current "cage". Interacting with the environment can also refer to constructing objects. The controllers presented in Chapter 6 are particularly well suited to quadrotor construction tasks, where objects must be deposited in specified locations. Since the controllers can be calculated online, and hence task assignment can be ongoing, as we have shown, we can construct complex objects which require building in a particular order, without predefining tasks for each of the robots.

11.3.2 Abstractions for multirobot control

By using abstractions, we can control the behavior of large groups without generating individual controllers for each agent. I would like to extend my work in Chapter 4 to a number of applications, including creating network topologies, controlling satellite

clusters, and automated warehouse systems.

- *Generating network topologies:* To generate a specific network topology we can design an abstraction such that it contains only the lattice structure of the network and let this propagate throughout the network, so that all agents create the desired lattice with their neighbors.

- *Satellite clusters:* With current research thrusts in switching from monolith satellites to clusters, there is a need to avoid damage from space debris while maintaining network topology and functionality. Including the network topology requirements in the abstraction and using the abstracted state to control the orbit of the entire cluster can provide an effective method of switching orbits and returning to the original after the threat has passed.

- *Automated warehouse systems:* The typical warehouse was not designed for the demand of small, unique drop shipments prompted by advent of e-commerce. Robots are being used in existing warehouses to bring shelves of items to packers, but digging out a shelf from behind hundreds of others can be challenging if we have to determine a path for each one. By moving them in groups, using an abstraction, the computation can be made much simpler.

11.4 Final remarks

In conclusion, we acknowledge the need for more flexible inter-robot and inter-group coordination. Environments may not always be fully known, communication will not be perfect inside a square annulus and drop to zero outside it, and robots will not always have perfect state information. However, we have taken steps toward alleviating the first two of these issues, by allowing obstacles in the free space and

enabling the robots to subdivide the space, and by modifying the communication graph so that it is determined by the agent pair's location in the environment instead of their relative position. It is a challenge to overcome the requirement for perfect state information, therefore it may be difficult to use this approach in cluttered outdoor environments. The key point, however, is that this approach can be used in real time in partially unknown environments, and still provide guarantees of convergence and safety.

Bibliography

[1] C. Anderson and N. R. Franks, "Teams in animal societies," *Behav Ecol*, vol. 12, no. 5, pp. 534–540, 2001.

[2] N. Ayanian and V. Kumar, "Decentralized feedback controllers for multi-agent teams in environments with obstacles," in *Proc. IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008, pp. 1936–1941.

[3] N. Ayanian, V. Kallem, and V. Kumar, "Synthesis of feedback controllers for multiple aerial robots with geometric constraints," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, San Francisco, CA, Sep. 2011.

[4] N. Ayanian and V. Kumar, "Abstractions and controllers for groups of robots in environments with obstacles," in *Proc. IEEE International Conference on Robotics and Automation*, Anchorage, AK, May 2010.

[5] ——, "Decentralized feedback controllers for multi-agent teams in environments with obstacles," *IEEE Trans. Robot.*, vol. 26, pp. 878–887, Oct. 2010.

[6] N. Ayanian, V. Kumar, and D. Koditschek, "Synthesis of controllers to create, maintain, and reconfigure robot formations with communication constraints," in *Proc. of International Symposium of Robotics Research*, Luzern, Switzerland, Aug. 2009, pp. 625–642.

[7] T. Balch and R. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Trans. Rob. Autom.*, vol. 14, no. 6, pp. 926–939, Dec 1998.

[8] T. Balch and M. Hybinette, "Social potentials for scalable multi-robot formations," in *Proc. IEEE Conf. Robotics and Automation*, vol. 1, Apr. 2000, pp. 73–80.

[9] M. A. Batalin and G. S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, Fukuoka, Japan, June 2002, pp. 373–382.

[10] R. Beard, J. Lawton, and F. Hadaegh, "A feedback architecture for formation control," in *Proc. of American Control Conference*, vol. 6, 2000, pp. 4087 – 4091.

[11] C. Belta and V. Kumar, "Abstraction and control for groups of robots," *IEEE Trans. Rob.*, vol. 20, no. 5, pp. 865–875, Oct. 2004.

[12] S. Berman, Q. Lindsey, M. Sakar, V. Kumar, and S. Pratt, "Study of group food retrieval by ants as a model for multi-robot collective transport strategies," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[13] E. Borzello and L. Merkle, "Multi-agent cooperation using the ant algorithm with variable pheromone placement," in *IEEE Congress on Evolutionary Computation*, vol. 2, Sep. 2005, pp. 1232 – 1237.

[14] M. Branicky, "Stability of switched and hybrid systems," *Proc. IEEE Conf. Decision Control, Lake Buena Vista, FL*, vol. 4, pp. 3498 – 3503, Dec. 1994.

[15] L. Chaimowicz, A. Cowley, D. Gomez-Ibanez, B. Grocholsky, M. Hsieh, H. Hsu, J. Keller, V. Kumar, R. Swaminathan, and C. Taylor, "Deploying air-ground multi-robot teams in urban environments," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, L. Parker, F. Schneider, and A. Schultz, Eds. Springer Netherlands, 2005, pp. 223–234.

[16] L. Chaimowicz, B. Grocholsky, J. Keller, V. Kumar, and C. Taylor, "Experiments in multirobot air-ground coordination," in *Proc. IEEE Conf. Robotics and Automation*, vol. 4, apr. 2004, pp. 4053 – 4058.

[17] P. Cheng, J. Fink, S. Kim, and V. Kumar, "Cooperative towing with multiple robots," in *Algorithmic Foundation of Robotics VIII*, ser. Springer Tracts in Advanced Robotics, G. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Springer Berlin / Heidelberg, 2009, vol. 57, pp. 101–116.

[18] D. Conner, H. Choset, and A. Rizzi, "Provably correct navigation and control for non-holonomic convex-bodied systems operating in cluttered environments," in *Robotics: Science and Systems*. Philadelphia, Pennsylvania: MIT Press, August 2006.

[19] D. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[21] J. Cortes, S. Martinez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1289 –1298, Aug. 2006.

[22] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 243–255, Apr. 2004.

[23] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, "Collective memory and spatial sorting in animal groups," *Journal of Theoretical Biology*, vol. 218, no. 1, pp. 1 – 11, 2002.

[24] N. J. Cowan, "Navigation functions on cross product spaces," *IEEE Trans. Autom. Control*, vol. 52, no. 7, pp. 1297–1302, 2007.

[25] J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario, "Self-organizing collection and transport of objects in unpredictable environments," in *Japan-U.S.A. Symposium on Flexible Automation*, 1990, pp. 1093–1098.

[26] J. Desai, J. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 17, no. 6, Dec. 2001.

[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[28] D. Dimarogonas, M. Zavlanos, S. Loizou, and K. Kyriakopoulos, "Decentralized motion control of multiple holonomic agents under input constraints," in *Proc. IEEE Conference on Decision and Control*, vol. 4, Dec. 2003, pp. 3390–3395.

[29] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Trans. Rob. Autom.*, vol. 17, no. 6, pp. 947–951, Dec. 2001.

[30] E. A. Emerson, "Temporal and modal logic," in *Handbook of theoretical computer science (vol. B): formal models and semantics*. Cambridge, MA, USA: MIT Press, 1990, pp. 995–1072.

[31] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Automat. Control*, vol. 49, no. 9, pp. 1465–1476, Sep. 2004.

[32] R. Fierro and A. Das, "Hybrid control of reconfigurable robot formations," in *Proc. of American Control Conference*, vol. 6, June 2003, pp. 4607 – 4612 vol.6.

[33] J. Fink, M. Hsieh, and V. Kumar, "Multi-robot manipulation via caging in environments with obstacles," in *Proc. IEEE Conf. Robotics and Automation*, Pasadena, CA, May 2008, pp. 1471 –1476.

[34] J. Fink, N. Michael, and V. Kumar, "Composition of vector fields for multi-robot manipulation via caging," in *Proceedings of Robotics: Science and Systems*, Atlanta, GA, June 2007.

[35] J. Fink, N. Michael, S. Kim, and V. Kumar, "Planning and control for cooperative manipulation and transportation with aerial robots," in *International Symposium on Robotics Research*, Lucerne, Switzerland, Aug. 2009.

[36] J. Fredslund and M. J. Mataric, "A general algorithm for robot formations using local sensing and minimal communication," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 837 – 846, Oct. 2002.

[37] M. Fujii, W. Inamura, H. Murakami, K. Tanaka, and K. Kosuge, "Cooperative control of multiple mobile robots transporting a single object with loose

handling," in *IEEE Intl Conf. Robotics and Biomimetics*, Sanya, China, Dec. 2007, pp. 816–822.

[38] K. Fukuda, "Frequently asked questions in polyhedral computation," 2000, http://www.ifor.math.ethz.ch/ fukuda/polyfaq/polyfaq.html.

[39] A. Ganguli, J. Cortés, and F. Bullo, "Multirobot rendezvous with visibility sensors in nonconvex environments," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 340–352, April 2009.

[40] M. C. D. Gennaro and A. Jadbabaie, "Formation control for a cooperative multi-agent system using decentralized navigation functions," in *Proceedings of the American Control Conference*, Minneapolis, Minnesota, June 2006.

[41] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proc. of Int. Conf. on Advanced Robotics*, Coimbra, June 2003.

[42] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, Sept. 2004.

[43] L. Habets and J. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, no. 1, Jan. 2004.

[44] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100 –107, July 1968.

[45] M. A. Hsieh, V. Kumar, and L. Chaimowicz, "Decentralized controllers for shape generation with robotic swarms," *Robotica*, vol. 26, no. 5, pp. 691–701, Sept. 2008.

[46] A. Jadbabaie, J. Lin, and A. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *Automatic Control, IEEE Transactions on*, vol. 48, no. 6, pp. 988 – 1001, June 2003.

[47] R. Johansson and A. Saffiotti, "Navigating by stigmergy: A realization on an RFID floor for minimalistic robots," in *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 245 –252.

[48] E. W. Justh and P. S. Krishnaprasad, "Equilibria and steering laws for planar formations," *Systems and Control Letters*, vol. 52, no. 1, pp. 25 – 38, 2004.

[49] W. Kang, N. Xi, and A. Sparks, "Formation control of autonomous agents in 3d workspace," in *IEEE International Conference on Robotics and Automation*, vol. 2, San Francisco, CA, Apr. 2000, pp. 1755 –1760.

[50] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[51] M. Keil and J. Snoeyink, "On the time bound for convex decomposition of simple polygons," in *Proceedings of the 10th Canadian Conference on Computational Geometry*, 1998, pp. 54–55.

[52] H. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[53] V. Klee, "On the complexity of d-dimensional Voronoi diagrams," *Archiv der Mathematik*, vol. 34, no. 1, pp. 75–80, December 1980.

[54] D. E. Koditschek, "The application of total energy as a Lyapunov function for mechanical control systems," in *Dynamics and control of multibody systems (Brunswick, ME, 1988)*. Providence, RI: Amer. Math. Soc., 1989, pp. 131–157.

[55] D. Koditschek, "Adaptive techniques for mechanical systems." in *Fifth Yale Workshop on Applications of Adaptive Systems Theory*, May 1987, pp. 259–265.

[56] E. Kranakis, D. Krizanc, and S. Rajsbaum, "Mobile agent rendezvous: A survey," in *Structural Information and Communication Complexity*, ser. Lecture Notes in Computer Science, P. Flocchini and L. Gasieniec, Eds. Springer Berlin / Heidelberg, 2006, vol. 4056, pp. 1–9.

[57] H. Kress-Gazit, N. Ayanian, G. J. Pappas, and V. Kumar, "Recycling controllers," in *IEEE International Conf. on Automation Science and Engineering*, Washington, DC, Aug. 2008, pp. 772–777.

[58] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "From structured english to robot motion," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, San Diego, CA, Oct. 2007, pp. 2717–2722.

[59] ——, "Where's waldo? sensor-based temporal logic motion planning," in *IEEE International Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, pp. 3116–3121.

[60] C. R. Kube and E. Bonabeau, "Cooperative transport by ants and robots," *Robotics and Autonomous Systems*, vol. 30, no. 1-2, pp. 85 – 101, Jan. 2000.

[61] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-parametric toolbox (MPT)," http://control.ee.ethz.ch/ mpt/ 2004.

[62] T. Lan, S. Liu, and S. Yang, "Collective cooperation inspired by stigmergy strategy," in *World Congress on Intelligent Control and Automation*, vol. 1, 2006, pp. 441 –445.

[63] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[64] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[65] J. Lawton, R. Beard, and B. Young, "A decentralized approach to formation maneuvers," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 6, pp. 933 – 941, Dec. 2003.

[66] D. Lee and M. Spong, "Stable flocking of multiple inertial agents on balanced graphs," *IEEE Transactions on Automatic Control*, vol. 52, no. 8, pp. 1469 –1475, Aug. 2007.

[67] N. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials and coordinated control of groups," in *Proc. of IEEE Conf. on Decision and Control*, vol. 3, Orlando, FL, Dec. 2001, pp. 2968–2973.

[68] M. A. Lewis and K.-H. Tan, "High precision formation control of mobile robots using virtual structures," *Autonomous Robots*, vol. 4, pp. 387–403, 1997.

[69] J. Lin, A. Morse, and B. Anderson, "The multi-agent rendezvous problem," in *Proc. IEEE Conference on Decision and Control*, vol. 2, dec. 2003, pp. 1508 – 1513.

[70] Z. Lin, M. Broucke, and B. Francis, "Local control strategies for groups of mobile autonomous agents," *Automatic Control, IEEE Transactions on*, vol. 49, no. 4, pp. 622 – 629, Apr. 2004.

[71] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *IEEE Conference on Decision and Control*, Seville, Spain, Dec. 2005, pp. 3553–3559.

[72] ——, "Computing smooth feedback plans over cylindrical algebraic decompositions," in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[73] ——, "Smooth feedback for car-like vehicles in polygonal environments," in *IEEE Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, pp. 3104 – 3109.

[74] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Real time feedback control for nonholonomic mobile robots with obstacles," in *IEEE Conference on Decision and Control*, San Diego, CA, Dec. 2006, pp. 2406 – 2411.

[75] S. Loizou, D. Dimarogonas, and K. Kyriakopoulos, "Decentralized feedback stabilization of multiple nonholonomic agents," in *Proc. of IEEE International Conf. on Robotics and Automation*, vol. 3, 2004, pp. 3012–3017.

[76] S. Loizou and K. Kyriakopoulos, "Closed loop navigation for multiple non-holonomic vehicles," in *Proc. of IEEE International Conference on Robotics and Automation*, vol. 3, Sept. 2003, pp. 4240–4245.

[77] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[78] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, Dec. 1996.

[79] J. Marshall, M. Broucke, and B. Francis, "Formations of vehicles in cyclic pursuit," *IEEE Transactions on Automatic Control*, vol. 49, no. 11, pp. 1963 – 1974, Nov. 2004.

[80] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms," in *Proceedings of Robotics: Science and Systems*, Cambridge, MA, June 2005.

[81] Y. Meng and J. Gan, "Self-adaptive distributed multi-task allocation in a multi-robot system," in *IEEE Congress on Evolutionary Computation*, Jun. 2008, pp. 398–404.

[82] M. Mesbahi and F. Y. Hadaegh, "Formation flying control of multiple spacecraft via graphs, matrix inequalities, and switching," *AIAA J. Guidance, Control, and Dynamics*, vol. 24, no. 2, pp. 369–377, Mar.–Apr. 2001.

[83] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[84] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robot. Autom. Mag.*, vol. 17, no. 3, pp. 56 –65, Sep. 2010.

[85] N. Michael, M. Zavlanos, V. Kumar, and G. Pappas, "Distributed multi-robot task assignment and formation control," in *Proc. IEEE Conf. Robotics and Automation*, Pasadena, CA, May 2008, pp. 128 –133.

[86] N. Michael, "Personal communication," 2007, university of Pennsylvania.

[87] N. Michael and V. Kumar, "Controlling shapes of ensembles of robots of finite size with nonholonomic constraints," in *Proceedings of Robotics: Science and Systems*, June 2008, pp. 157–162.

[88] N. Miyata, J. Ota, Y. Aiyama, and T. Arai, "Real-time task assignment for multi-agent robots-application to cooperative transportation task," in *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, vol. 4, 1999, pp. 761 –768.

[89] N. Miyata, J. Ota, T. Arai, and H. Asama, "Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 769 – 780, Oct. 2002.

[90] P. Ogren, M. Egerstedt, and X. Hu, "A control lyapunov function approach to multi-agent coordination," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 2, 2001, pp. 1150–1155 vol.2.

[91] R. Olfati-Saber and R. Murray, "Flocking with obstacle avoidance: cooperation with limited communication in mobile networks," in *IEEE Conf. on Decision and Control*, vol. 2, Dec. 2003, pp. 2022–2028.

[92] ——, "Distributed cooperative control of multiple vehicle formations using structural potential functions," in *Proc. of IFAC World Congress*, Barcelona, July 2002.

[93] E. Ostergaard, M. Mataric, and G. Sukhatme, "Distributed multi-robot task allocation for emergency handling," in *Proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, vol. 2, 2001, pp. 821–826.

[94] D. A. Paley, N. E. Leonard, and R. Sepulchre, "Stabilization of symmetric formations to motion around convex loops," *Systems and Control Letters*, vol. 57, no. 3, pp. 209 – 215, 2008.

[95] J. K. Parrish, S. V. Viscido, and D. Grunbaum, "Self-organized fish schools: An examination of emergent properties," *Biol. Bull.*, vol. 202, no. 3, pp. 296–305, Jun. 2002.

[96] G. Pereira, V. Kumar, J. Spletzer, C. Taylor, and M. Campos, "Cooperative transport of planar objects by multiple mobile robots using object closure," in *Experimental Robotics VIII*, ser. Springer Tracts in Advanced Robotics, B. Siciliano and P. Dario, Eds. Springer Berlin / Heidelberg, 2003, vol. 5, pp. 287–296.

[97] L. C. A. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. C. Mesquita, and G. A. S. Pereira, "Simultaneous coverage and tracking (scat) of moving targets with robot networks," in *Proc. Int. Workshop Algorithmic Found. Robot.*, Guanajuato, Mexico, Dec. 2008.

[98] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive designs," in *VMCAI*, Charleston, SC, Jan. 2006, pp. 364–380.

[99] W. Ren, "Consensus based formation control strategies for multi-vehicle systems," in *Proc. of American Control Conference*, Minneapolis, Minnesota, June 2006, p. 6 pp.

[100] W. Ren and R. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 655 – 661, May 2005.

[101] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Comput. Graph.*, vol. 21, no. 4, pp. 25–34, July 1987.

[102] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Trans. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, 1992.

[103] E. Rimon and D. Koditschek, "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," *Trans. of the American Mathematical Society*, vol. 327, no. 5, Sept. 1991.

[104] ——, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robotics and Automation*, vol. 8, no. 5, Oct. 1992.

[105] B. Roszak and M. E. Broucke, "Necessary and sufficient conditions for reachability on a simplex," *Automatica*, vol. 42, no. 11, Nov. 2006.

[106] S. Russel and P. Norvig, *Artificial Intelligence: a Modern Approach*, 1st ed. Englewood Cliffs, NJ: Prentice Hall, 1995.

[107] Y. Sa'ar, "Java temporal logic verifier (jtlv)," http://jtlv.sourceforge.net 2008.

[108] B. E. Sagan, *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*, 2nd ed., ser. Graduate Texts in Mathematics. New York: Springer-Verlag, 2001, vol. 203.

[109] J. Sanchez and R. Fierro, "Sliding mode control for robot formations," in *IEEE Intl Symposium on Intelligent Control*, Houston, TX, Oct. 2003, pp. 438 – 443.

[110] P. Santana, J. Barata, H. Cruz, A. Mestre, J. Lisboa, and L. Flores, "A multi-robot system for landmine detection," in *proceedings of IEEE Conf. Emerging Technologies and Factory Automation*, vol. 1, Sep. 2005, pp. 721–728.

[111] M. Schneider-Fontan and M. Mataric, "Territorial multi-robot task division," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 5, pp. 815 –822, Oct. 1998.

[112] M. Schwager, D. Rus, and J.-J. E. Slotine, "Decentralized, adaptive control for coverage with networked robots," *The International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[113] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, no. 1, pp. 165–200, May 1998.

[114] B. Smith, M. Egerstedt, and A. Howard, "Automatic deployment and formation control of decentralized multi-agent networks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2008, pp. 134–139.

[115] B. Smith, J. Wang, and M. Egerstedt, "Persistent formation control of multi-robot networks," in *Proceedings of the IEEE Conference on Decision and Control*, Dec. 2008, pp. 471–476.

[116] F. Steele, Jr. and G. Thomas, "Directed stigmergy-based control for multi-robot systems," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, Arlington, Virginia, USA, 2007, pp. 223 – 230.

[117] P. Stone and M. M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, no. 2, June 1999.

[118] A. Sudsang, F. Rothganger, and J. Ponce, "An implemented planner for manipulating a polygonal object in the plane with three disc-shaped mobile robots," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, vol. 3, 2001, pp. 1499 –1506.

[119] K. Sugihara and I. Suzuki, "Distributed algorithms for formation of geometric patterns with many mobile robots," *JOURNAL OF ROBOTIC SYSTEMS*, vol. 13, no. 3, pp. 127–139, Mar. 1996.

[120] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1347–1363, 1999.

[121] F. Tang and L. Parker, "A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation," in *Proc. IEEE Conf. Robotics and Automation*, Apr. 2007, pp. 3351 –3358.

[122] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Trans. Automat. Contr.*, vol. 52, no. 5, pp. 863–868, May 2007.

[123] H. G. Tanner and A. Kumar, "Formation stabilization of multiple agents using decentralized navigation functions," in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[124] H. Tanner, G. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Trans. Rob. Autom.*, vol. 20, no. 3, pp. 443–455, June 2004.

[125] R. Tedrake, I. R. Manchester, M. M. Tobenkin, and J. W. Roberts, "LQR-Trees: Feedback motion planning via sums of squares verification," *International Journal of Robotics Research*, vol. 29, pp. 1038–1052, July 2010.

[126] Z. Wang and V. Kumar, "Object closure and manipulation by multiple cooperating mobile robots," in *Proc. IEEE Conf. Robotics and Automation*, vol. 1, 2002, pp. 394–399.

[127] J. Werfel, D. Ingber, and R. Nagpal, "Collective construction of environmentally-adaptive structures," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, Oct. 2007, pp. 2345 –2352.

[128] J. Werfel and R. Nagpal, "Extended stigmergy in collective construction," *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 20 – 28, Mar. 2006.

[129] G. A. Xiaoping Yun and O. Albayrak, "Line and circle formation of distributed physical mobile robots," *Journal of Robotic Systems*, vol. 14, no. 2, pp. 63–76, Feb. 1997.

[130] P. Yang, R. Freeman, and K. Lynch, "Multi-agent coordination by decentralized estimation and control," *IEEE Trans. Automat. Contr.*, vol. 53, no. 11, pp. 2480 –2496, Dec. 2008.

[131] M. Zavlanos, A. Jadbabaie, and G. Pappas, "Flocking while preserving network connectivity," in *Proc. IEEE Conference on Decision and Control*, Dec. 2007, pp. 2919 –2924.

[132] M. Zavlanos and G. Pappas, "Distributed formation control with permutation symmetries," in *Proc. of IEEE Conf. on Decision and Control*, New Orleans, LA, Dec. 2007, pp. 2894 –2899.

[133] ——, "Dynamic assignment in distributed motion planning with local coordination," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 232 –242, Feb. 2008.

[134] P. Zebrowski, Y. Litus, and R. Vaughan, "Energy efficient robot rendezvous," in *Canadian Conference on Computer and Robot Vision*, May 2007, pp. 139 –148.

[135] Y. Zhang, M. Schervish, and H. Choset, "Probabilistic hierarchical spatial model for mine locations and its application in robotic landmine search," in *proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, vol. 1, Lausanne, Switzerland, 2002, pp. 681 – 689.